
Dynamic Discovery of Type Classes and Relations in Semantic Web Data

Serkan Ayvaz · Mehmet Aydar

Abstract The continuing development of Semantic Web technologies and the increasing user adoption in the recent years have accelerated the progress incorporating explicit semantics with data on the Web. With the rapidly growing RDF (Resource Description Framework) data on the Semantic Web, processing large semantic graph data have become more challenging. Constructing a summary graph structure from the raw RDF can help obtain semantic type relations and reduce the computational complexity for graph processing purposes. In this paper, we addressed the problem of graph summarization in RDF graphs, and we proposed an approach for building summary graph structures automatically from RDF graph data. Moreover, we introduced a measure to help discover optimum class dissimilarity thresholds and an effective method to discover the type classes automatically. In future work, we plan to investigate further improvement options on the scalability of the proposed method.

Keywords Semantic Web · RDF · Graph Summarization · Automatic Weight Generation

1 Introduction

The Web as the global information source is growing exponentially. In recent years, there has been significant developments in publishing data with expressed semantics in the Web. The Linking Open Data[6] and similar community projects have recommended the publication of large amount of globally useful datasets in machine-readable forms. Moreover, the utilization of

Serkan Ayvaz
Department of Software Engineering, Bahcesehir University, Besiktas 34353, Istanbul, Turkey.
E-mail: serkan.ayvaz@eng.bau.edu.tr

Mehmet Aydar
Department of Computer Science, Kent State University, Kent, Ohio 44240, USA.
E-mail: maydar@kent.edu

Resource Description Framework (RDF), along with other forms of semantic data in forms of RDFa [1] and microformats [22] in web pages has expanded.

As a standard data model for the Semantic Web, RDF is a graph-structured general purpose language for representing information in a way that the resources are described unambiguously using RDF statements. The RDF statements are in the form of subject-predicate-object triples. Every triple is a relationship between two entities.

RDF uses `rdf:type` property for stating class membership of entities. The entity type information is particularly useful for semantic searches in finding related entities and in traversal of the hierarchical structure of the RDF graph. However, the semantic data available on the Web today often don't have precise entity type information. It is partially due to (1) not containing the entity types owing to the flexibility of RDF model not forcing constraints on the schema, (2) representing data with non-standard vocabularies for typing as some data publishers do not use standard vocabularies such as `rdf:type` and `rdfs:subClassOf`, which is making it challenging to locate the type triples, furthermore, (3) defining the type information too generally that loosely coupled entities are represented in the same types.

Constructing a graph structure containing the entity type classes, class attributes and relations between the type classes can be instrumental for Semantic Search algorithms in terms of query time since the entire input data does not need to be completely processed at the query time. We call this structure as the Summary Graph [5,4]. Semantic search is a common graph processing task and it often requires a summary graph structure for effective and faster graph processing. For instance, the semantic search approach proposed by [41] uses a summary graph structure in the search mechanism. They generate the summary graph using a set of aggregation rules, which calculate the equivalence classes of all nodes belonging to one type class and project all edges to corresponding edges accordingly. In this approach, one needs to know what constitutes a type class in advance. However, this assumption may not be realistic for real-world RDF datasets, i.e., the RDF data may not be tied to a standard ontology or vocabulary. Our work attempts to address this issue by automatically building the summary graph structure from the data itself by utilizing graph node similarity scores.

There exists related methods to obtain a summary graph: (1) A summary graph can be obtained from the dataset ontology, if the dataset is already tied to an ontology. (2) Another way to obtain the summary graph is to locate the type triples (`rdf:type`) in the dataset and to organize the type classes and relations accordingly, if the data set is published using a standard vocabulary [14]. (3) Or the summary graph can be built automatically without relying on an ontology or a standard vocabulary. Our graph summarization approach is based on the latter method. `Rdf:type` is an optional property and it is often missing in commonly used datasets. Furthermore, it can potentially be inconsistent or erroneous in some cases [32]. Thus, the methods, which rely on `rdf:type` or existence of an ontology may have implications for general use. Therefore, there is a need for automatic generation of summary graphs from

RDF Data. In this regard, we describe an entity similarity metric and the methods used for automatically generating a summary graph from RDF Data. To the best of our knowledge, this is the first approach to attempt to generate summary graph of RDF graph automatically based on the entity similarities.

Contributions and Outline

In this study, we focus on the problem of efficiently building a summary graph structure automatically from underlying RDF data. We utilize the notion of entity similarity in an RDF dataset so that fundamentally similar entities could be associated with the same class, which we call it type class in this paper.

In our approach, the type classes, importance weight of each property and each string word for each of the referenced IRIs (Internationalized Resource Identifiers) are auto-generated. Furthermore, the weights in the pairwise similarity calculation are generated dynamically and applied during the summary graph generation. Our methodology is to utilize graph locality and neighborhood similarity. Our algorithm does not rely on the existence of a common vocabulary. We use the Jaccard measure context for entity similarity such that the properties of the entities are treated as the dimensions of the entities.

A stability measure, which represents the degree of confidence of a relation between classes in the summary graph, is proposed. From the input RDF data itself, we generate the summary graph along with the classes and class relations with the stability measure for each class relation. The main contributions of present study are

- We investigated the graph summary problem in RDF graphs and provided an effective approach for generating summary graphs automatically from RDF data.
- For automatic discovery of the summary graphs, we introduced a measure, which we call Favorability that helps discover optimal class dissimilarity thresholds and provided an effective method that discovers the type classes using the class threshold measure.
- To assess the effectiveness of our approach, we applied our methods to real-world datasets.

The rest of the paper is organized as follows. In section 2, we define the graph summarization problem in RDF graph data. Then, in section 3, we discuss our methods and the algorithms in detail. In section 4, the results of the evaluations assessing the efficiency of the proposed methods are presented. Finally, we review the related work in section 6 and follow this with our conclusion and future work in section 7.

2 Defining Graph Summarization Problem

RDF data consist of a collection of statements that intrinsically represent a labeled, directed multi-graph with which the resources are expressed unam-

biguously. RDF statements describe resources in the form of triples, consisting of subject-predicate-object expressions that describe a resource, the type of a resource (type triple), or a relationship between two resources [12].

To describe resources, each RDF node that corresponds to an RDF entity is represented with an IRI. The values such as strings, numbers and dates in RDF data are represented by literal nodes. A predicate in an RDF triple is also called a property of the RDF subject node. A predicate can be one of two types: a `DatatypeProperty` where the subject of the triple is an IRI and the object of the triple is a literal or an `ObjectProperty` where both the subject and object of the triple are IRIs. Each object of a subject node is called a neighbor of that subject node. The subject in an RDF triple is either an IRI or a blank node, the predicate is an IRI, and the object is either an IRI, a literal or a blank node. The subjects and objects of triples in the RDF graph form RDF nodes. As an example, figure 1 represents two sample university entities *Kent_State* and *Case_Western* and their properties.

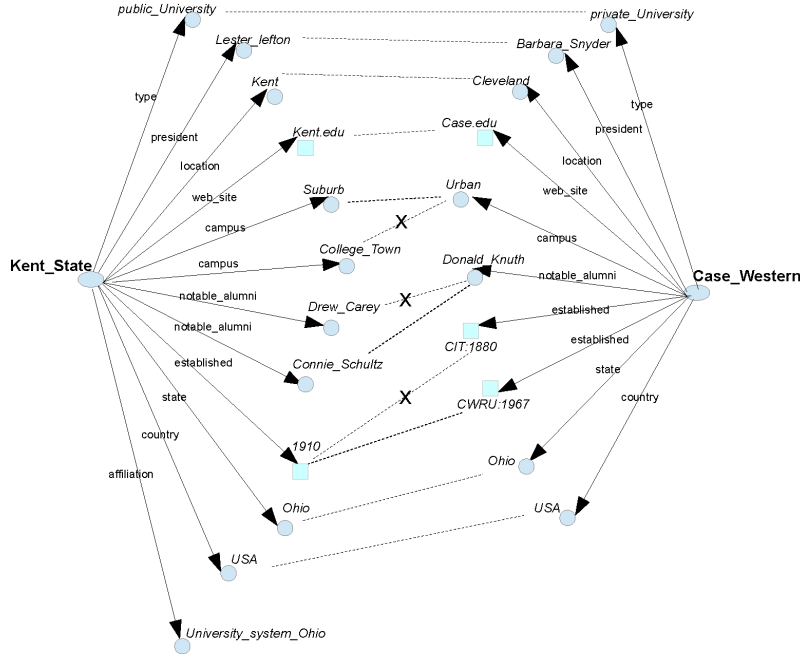


Fig. 1 Sample graph demonstrating two nodes

Formally, RDF graph is a directed labeled graph, which can be represented as $G = (V, L, E)$, where the set of vertices V represents entities (resources), the set of directed edges E of the form $l(u, v)$, with $u \in V$, $v \in V$ and $l \in L$,

denote predicates (properties) between entities, and the labels L are predicate names or labels. Note that an edge $l(u, v)$ represents the RDF triple (u, l, v) .

A summary graph of a data graph is the directed graph such that each node in the summary graph is a subset of the original graph nodes of the same type. Thus, we define the Graph Summarization Problem as finding the corresponding summary graph $G' = (V', L', E')$ of G , such that each element of V' is a subset of V containing all elements of the same type. For $v \in V$, we let $[v]$ denote the subset of V containing all elements in V with the same type class as v . The vertices V' in the summary graph G' are equivalence classes over the original graph G , and the vertices in V' are disjoint subsets of V . E' and L' are, respectively, the sets of edges and labels in the graph G' . Hence, $L' \subset L$, and the elements of E' are defined by the elements in the equivalence classes in V' and the edges in E . Let $u, v \in V$; then $[u], [v] \in V'$. There is an edge $l([u], [v]) \in E'$ if and only if there exist $s \in [u] \subseteq V$ and $t \in [v] \subseteq V$ such that $l(s, t) \in E$.

3 Methods

The Graph Summarization Problem can be considered as a problem of identifying entity type classes. The set of entity type classes can be inferred from RDF data such that each type class in the set of entity types contains the same or very similar entities only. In our method, the entity type classes are derived from the entity similarities. The discovery of the type classes, i.e., the elements v in V' can be also seen as clustering problem. Using the calculated similarity measurements of the entity pairs, the entities that are the same or very similar, satisfying a similarity threshold, are combined in the same type class. Our entity similarity measurement approach is based on the intuition that the graph nodes that have similar relations to similar neighbors tend themselves to be similar nodes.

Previously, we investigated methods for computing entity similarities effectively. We then developed a framework for building a summary graph structure in RDF data [5, 4]. This current study extends our summary graph generation approach [5] and enhances it by incorporating a measure to help discover optimum class dissimilarity thresholds and an effective method to discover the type classes automatically.

Similarity of IRI Nodes

Intuitively, the characteristics of an RDF graph node are defined by its properties and the neighboring entities which are connected and related by similar properties. Based on the intuition that similar IRI nodes tend to have similar properties and interact with similar neighbor nodes, the similarities of entities in our method are calculated using the predicates of the IRI nodes, in addition to the neighbor nodes that they interact with common predicates. By neighbor we mean that a neighbor of a graph node is another node which is “connected”

by a predicate. More formally, a node u is connected to node v , i.e., u is a neighbor of v , if there is a label $l \in L$ such that $l(u, v) \in E$. Therefore, a node and its neighbors are connected by a property. Thus, the similarity calculation may yield more accurate results with the addition of neighborhood similarity.

Similarity of Literal Neighbor Nodes

The similarity of literal nodes indirectly impact the similarity of IRI nodes in the calculation of neighborhood similarity. The neighbors of IRI nodes can be either other IRI nodes or literals. Incorporating neighboring literals in the computation of the similarity of pairs can be beneficial, especially, in datasets where the entities are commonly described using literals. Therefore, the similarity of literal neighbor nodes are taken into account in our approach.

A literal node can consist of two or three elements: a lexical form, a datatype IRI and a language tag. The language tag in a literal node is included if and only if the datatype IRI of the literal node corresponds to `rdf:langString` [7]. It is important to note that the literals should be in the same language while incorporating literals in the computation of the similarity of IRI node pairs. As the same literals may have totally different meanings in different languages, we are assuming that all the literals are in the same language. If present, the `rdf:langString` component of the literal nodes is expected to have only one value. When calculating the similarity, the lexical form and the data type URI components a pair of literal nodes are considered. As comparing different data types is meaningless, the similarity of literal nodes is considered only when the two data types are equal.

Inferring the semantics of literal nodes is challenging. To calculate the similarity of pairs of IRI nodes, an effective literal node similarity metric is needed. We make use of string similarities for the lexical form components of the literal nodes that measures common words within the two lexical forms along with their auto-generated importance weights. While calculating the weight of word importance in literal nodes consisting of a set of words, we consider the following factors: the source subject node, the frequency of the word within the triple collection for each subject node, and the frequency of the word within the entire dataset.

$$LiteralSim(x, y) = \frac{\sum_{i \in (x \cap y)} |t_i| \times w_i}{\sum_{j \in (x \cup y)} |t_j| \times w_j} \quad (1)$$

where t is the term that appears in the neighbor literal nodes, such that, u, v are IRI nodes in V , x, y are literal nodes in V , and $l(u, x), l(v, y) \in E$, and $\sum_{j \in (x \cup y)} w_j = 1$. $|t_i|$ is the number of times the term appears and w_i is the importance weight of the term for the literal nodes (u, v) .

The importance weight of the term for the literal nodes is calculated based on the concept of the term frequency-inverse document frequency ($tf - idf$)

[27,37], which is a widely known technique in information retrieval. $tf - idf$ indicates that some terms may be important in some documents but not as important in other documents. Said differently, the importance of a word in a document increases by its frequency in the document but its importance decreases by its frequency in the corpus [34].

3.1 Computation of Pairwise Entity Similarities

To identify the type classes in the summary graph, a metric is required to calculate the similarities of entities. For entity similarity metric, we employ an efficient graph node pair similarity metric, which utilizes the graph localities and neighborhood similarity within RoleSim similarity [21] in conjunction with the Jaccard measure context [19].

Jaccard Similarity Measure

The Jaccard similarity coefficient also known as the Jaccard index is a well-known statistical measure. It is commonly used for comparing similarity and diversity of sample sets. Jaccard similarity is simply defined as the size of the intersection divided by the size of the union of the sample sets [19].

For given two sets $S1$ and $S2$ in a dataset, the Jaccard similarity, $J(S1, S2)$, between $S1$ and $S2$ is formulated as:

$$J(S1, S2) = \frac{|S1 \cap S2|}{|S1 \cup S2|} \quad (2)$$

When calculating the Jaccard similarity in RDF data, the subject nodes are considered to be the names or labels for the sets. Thus, the subject of the triples determine the sets. Similarly, the properties of the triples whose subject is the name or label of the set are the elements of each set. The objects of the triples whose subject is the name or label of the set become the neighbors of each subject set. The object nodes, or in other words the neighboring nodes, may themselves be names or labels of sets. For given two subject nodes u and v in an RDF graph, we calculate the Jaccard similarity by noting that $|u \cap v|$ is the number of predicates that the subject nodes u and v have in common while $|u \cup v|$ is the number of predicates in the union of the subject nodes u and v .

RoleSim Similarity Measure

The Jaccard similarity has a limitation when the Jaccard index applied to an RDF graph. Because the Jaccard index determines the set similarity based on the number of common set elements only, by treating the subject nodes as sets and the predicates of the subject nodes as the set elements. However, it does not consider the relations between set elements. Thus, it does not take into account the neighboring node similarities.

For this reason, we utilize the RoleSim similarity metric which is based on the maximal matching of neighborhood pairs and a simple iterative computational method. The intuition in RoleSim similarity measure is that two nodes or entities tend to have the same role when they interact with equivalent sets of neighbors.

Given a regular unlabeled graph $G = (V, E)$, RoleSim measures the similarity of each node pair in V based on their neighborhood similarities [21]:

$$\begin{aligned} RoleSim(u, v) = & (1 - \beta) \\ & \times \max_{M \in Mm(u, v)} \frac{\sum_{(x, y) \in M} RoleSim(x, y)}{N_u + N_v - |M|} \\ & + \beta \end{aligned} \quad (3)$$

$RoleSim(u, v)$ denotes the similarity of the nodes $u, v \in V$. The definition of RoleSim is recursive; i.e., $RoleSim(x, y)$ is calculated the same way as $RoleSim(u, v)$. $N(u)$ and $N(v)$ denote their respective sets of neighborhoods and N_u and N_v denote their respective degrees, i.e., $N_u = |N(u)|$ and $N_v = |N(v)|$.

M is defined as a set of ordered pairs (x, y) where $x \in N(u)$ and $y \in N(v)$ such that there does not exist $(x', y') \in M$, s.t. $x = x'$ or $y = y'$, and moreover, M is maximal in that no more ordered pairs may be added to M and keep the constraint above. $Mm(u, v)$ is the set of all such M 's. $Mm(u, v)$ is a set of sets.

M is a maximal subset of $N(u) \times N(v)$ such that no element of $N(u)$ appears more than once as a first coordinate and no element of $N(v)$ appears more than once as a second coordinate of an ordered pair in M . Thus, $|M| = \min(N_u, N_v)$. The maximal matching ensures that the total value of selected cells has the maximum possible value. The maximal matching value, $M(u, v)$, is calculated as

$$M(u, v) = \max_{M \in Mm(u, v)} \frac{\sum_{(x, y) \in M} RoleSim(x, y)}{Max(N_u, N_v)} \quad (4)$$

The parameter β is a decay factor, $0 < \beta < 1$. The parameter β is for decreasing the influence of neighbors with further distance which dampens the recursive effect.

Combined Pairwise Entity Similarity Measure

To utilize neighborhood similarity in RDF graphs, we improve the initial Jaccard similarity by augmenting it with the RoleSim similarity measure of the neighboring nodes. When computing neighborhood similarity, comparing all

neighbors to all neighbors is not an efficient method. Thus, we compare only the neighboring nodes which are related by the same predicate. For instance, given two nodes u and v , let's assume that s_1 and s_2 are neighbors of u , and t is a neighbor of v . We calculate similarity of the neighborhood pairs (s_1, t) and (s_2, t) only if there is a predicate which connects u to s_1 , u to s_2 and also connects v to t , and we use the maximum similarity between the neighborhood pairs (s_1, t) and (s_2, t) as implied in the maximal matching concept of RoleSim similarity measure. The impact of the similarity of the neighbor nodes are weighted by each common predicate.

We note, however, that the generic version of the RoleSim measure is introduced for the unlabeled graph. In this work the input data is in RDF model, which is a directed and labeled graph. Therefore, we utilize the $RoleSim(u, v)$ measure when there may be multiple neighbors reached from the node pairs u and v by a common predicate, where u and v are the nodes in the input graph.

In the lists below, for $1 \leq i \leq j$, l_i is a label for an edge, i.e., $l_i \in L$. When $1 \leq h \leq j$ and if i and h are not equal, then l_i and l_h are different labels, i.e., l_i and l_h are different properties. $[x_i]$ and $[y_i]$ are the sets of nodes which are related to u and v , respectively, by predicate l_i .

$$l_1(u, [x_1]), l_2(u, [x_2]), \dots, l_j(u, [x_j]) \in E$$

$$l_1(v, [y_1]), l_2(v, [y_2]), \dots, l_j(v, [y_j]) \in E.$$

Thus, we are assuming that there are j different predicates which are predicates in triples with subject u and are also predicates in triples with subject v .

Then, by using the Jaccard index in conjunction with the RoleSim measure, their similarity can be calculated as:

$$\begin{aligned} PairSim(u, v)^k &= (1 - \beta) \\ &\times \frac{1}{|u \cup v|} \\ &\times \left(\sum_{j \in (u \cap v)} \max_{M \in Mm^j(u, v)} \left(\frac{\sum_{(x, y) \in M} Sim(x, y)^{k-1}}{N_u^j + N_v^j - |M|} \right) \times w_j \right) \\ &+ \beta \end{aligned} \tag{5}$$

where k is the iteration number $1 \leq k < MaxIter$, $MaxIter$ is the maximum number of iterations, such that, if $k = 3$ then $PairSim(u, v)^k$ denotes to the similarity of the node pair (u, v) at the third iteration and $PairSim(u, v)^{k-1}$ denotes to the similarity of the node pair (u, v) by the end of the second iteration. Also, $N_{(u)}^j$ and $N_{(v)}^j$ denote their respective neighborhoods that are reached by j th common edge. $x \in N_{(u)}^j$ and $y \in N_{(v)}^j$, and N_u^j and N_v^j denote their respective degree connected by j th common edge. Said differently, $N_{(u)}^j$ is the cardinality of $[x_j]$, and $N_{(v)}^j$ is the cardinality of $[y_j]$.

w_j is the weight of the property connecting the graph nodes (u, v) and their respective neighbors (x, y) .

$$Sim(x, y)^{k-1} = \begin{cases} PairSim(x, y)^{k-1}, & \text{if } x, y \text{ are IRI nodes} \\ LiteralSim(x, y), & \text{if } x, y \text{ are Literal nodes} \\ 0, & \text{otherwise} \end{cases} \quad (6)$$

We define M to be a set of ordered pairs (x, y) where $x \in N_{(u)}^j$ and $y \in N_{(v)}^j$ such that there does not exist $(x', y') \in M$, s.t. $x = x'$ or $y = y'$, and furthermore, M is maximal in that no more ordered pairs may be added to M and keep the constraint above. $Mm^j(u, v)$ is the set of all such M 's. $Mm^j(u, v)$ is a set of sets.

By a ‘‘maximal nonrepeating matching’’, we mean that we form as many pairs as we can from the elements in $N_{(u)}^j$ and $N_{(v)}^j$ with the restriction that no element in either $N_{(u)}^j$ and $N_{(v)}^j$ may be used in more than one ordered pair.

The parameter β is a decay factor $0 < \beta < 1$, which helps reduce the influence of neighbors with further distance due to the recursive effect. $l_1(u, x)$ and $l_2(v, y)$ represent directed edge labels s.t. $l_1, l_2 \in L$, and $l_1 = l_2$, $x \in N_{(u)}$ and $y \in N_{(v)}$.

3.2 The Summary Graph Generator Algorithm

While calculating the neighborhood similarity, our proposed node similarity metric makes calls to the immediate neighbors' similarities. Since neighbors' similarities depend on their own neighbors' similarities, the immediate neighbors' similarities are not known ahead of time. A solution involving recursive calls is not an efficient option in this case as it may lead to inefficient resource utilization and excessive recursion. For instance, an object node n_1 of a subject node n_2 in an RDF triple may be a subject node n_1 of the object node n_2 in another RDF triple. Therefore, our algorithm runs in multiple iterations until the rate of change in calculated similarities drops under a given threshold. It is a similar approach to the PageRank algorithm [30]. The initial similarity of a node pair is set to 1 if they share a common predicate and 0, otherwise. Such that:

$$\begin{aligned} \forall(u, v \in V) : \\ (S(u, v) = 1) &\rightarrow (|u \cap v| > 0) \text{ and,} \\ (S(u, v) = 0) &\rightarrow (|u \cap v| = 0) \end{aligned}$$

Our approach is two folds. Firstly, the pairwise similarity algorithm calculates the similarity values for each pair which constructs a similarity matrix. Once the pairwise similarities converge, the type class generation algorithm begins and generates the type classes i.e, it assigns the node u and v to the

same type class if their dissimilarity value is less than an auto calculated ϵ threshold which is the class dissimilarity threshold.

As the algorithm generates common pairs if two candidate nodes that share at least one common predicate, the overall complexity for the algorithm is n^2 in the worst case. It occurs when all subject nodes in the RDF graph have a common predicate with every other subject node. When the noise predicates excluded, i.e. the predicates that is referenced by most if not all the subject nodes, the algorithm performs better than n^2 . Thus, the complexity of the algorithm depends on the characteristics of the dataset. On a dense graph, the complexity approaches to n^2 while it gets near to $n(\log n)k$ time in sparse graphs, where k is a constant number of iteration.

The basic steps of the algorithm include sorting the triples according to their predicate label, extraction of the subject node pairs for each of the predicates, running the similarity computation algorithm in iterations until convergence and generating the type classes based on the calculated similarity measures.

The type class generation algorithm creates distinct classes, such that subject node pairs that have similarity greater than a given threshold get put to the same type class. The input parameter β is a decay factor $0 < \beta < 1$. $l_1(u, x)$ and $l_2(v, y)$ represent directed edge labels s.t. $l_1, l_2 \in L$, and $l_1 = l_2$, $x \in N_{(u)}$ and $y \in N_{(v)}$.

3.3 Dynamic Assignment of Weights of IRI Node Descriptors

An IRI node is described through its predicates and the collection of literal neighboring nodes in the lexical form. For simplicity, we call the predicates and literal neighboring nodes as descriptors of the IRI nodes. As stated above, the similarity of a pair of IRI nodes depend upon their descriptor similarities and the similarities of their neighbors.

The weight of each descriptor may vary significantly as each descriptor may have different impact on an IRI node. Hence, identifying appropriate metrics for generating IRI descriptor weights is a vital task in computation of accurate similarity values.

Upon investigations on the factors that can impact the weight of a descriptor, we propose an approach in this study for generating the importance weights of the IRI node descriptors automatically. Based on the investigations, we think that the weight of a descriptor may differ for each IRI for which it is a descriptor and the weight increases proportionally by the number of times a descriptor appears in the reference IRI, but it is offset by the frequency of the descriptor in the entire RDF dataset. This tendency is similar to the concept of the term frequency-inverse document frequency ($tf-idf$). While computing the weight of properties dynamically, we apply the $tfidf$ to the properties and nodes in RDF graphs. $tfidf$ is calculated as follows:

$$tfidf(p, u, G) = tf(p, u) \times idf(p, G). \quad (7)$$

Algorithm 1: SimMeasure

```

input      : Graph  $G(V, L, E)$ 
output     : Similarity-Matrix  $S$ , Pairs  $H$ 
parameter: MaximumIteration  $MaxIter$ , Iteration-Convergence-Threshold  $Ict$ 
 $\forall pair(u, v) \in V (S(u, v) \leftarrow 0);$ 
 $H \leftarrow \emptyset;$ 
 $T \leftarrow Sort(E)$  by  $l, u, v$  s.t.  $l(u, v) \in E;$ 
for each distinct pair( $u, v$ ) from  $T$  do
    if  $\exists (l_1(u, x) \text{ and } l_2(v, y))$  s.t.  $l_1 = l_2$  then
         $S(u, v) \leftarrow 1$ 
         $P(u, v) \leftarrow (u, v, L^j, N^j(u), N^j(v))$  where  $u, v \in V$ ,  $L^j$  is the list of common
            labels between  $u$  and  $v$ , and  $L^j \in L$ 
         $H \leftarrow H \cup \{P(u, v)\}$ 
    end
end
 $S_{previous} \leftarrow \emptyset$ 
 $converged \leftarrow false$ 
 $count \leftarrow 0$ 
while  $converged = false$  and  $count < MaxIter$  do
    for each  $((u, v, L^j, N^j(u), N^j(v)) \in H)$  do
         $PairSim(u, v)^k = (1 - \beta) \times \frac{1}{|u \cup v|} \times (\sum_{j \in (u \cap v)} max_{M \in M^{m^j}(u, v)} (\frac{\sum_{(x, y) \in M} Sim(x, y)^{k-1}}{N_u^j + N_v^j - |M|}) \times w_j) + \beta$ 
         $S(u, v) \leftarrow PairSim(u, v)^k$ 
    end
     $converged = |S - S_{previous}| \leq Ict$ 
     $S_{previous} \leftarrow S$ 
     $count \leftarrow count + 1$ 
end
return  $S, H$ 

```

where the term frequency (tf) [27] represents the frequency of a proposition p with respect to a graph subject node u . More exactly, when $u \in V$ and $p \in L$, then

$$f(p, u) = |\{v \in V : p(u, v) \in E\}|. \quad (8)$$

Equivalently, $f(p, u)$ is the number of RDF triples with subject u and property p .

To define $tf(p, u)$, it is helpful to have a notation for the set of all properties with subject u . Thus, for $u \in V$, $L(u) = \{q \in L : \exists v \in V \text{ with } q(u, v) \in E\}$. Then

$$tf(p, u) = \frac{f(p, u)}{\sum_{q \in L(u)} f(q, u)}. \quad (9)$$

Algorithm 2: CreateClasses

```

input      : Similarity-Matrix  $S$ , Pairs  $H$ 
output     : Auto-Generated-Type-Classes-Map  $C$ 
parameter: Class-Dissimilarity-Threshold  $\epsilon$ 
for each  $((u, v, L^j, N^j(u), N^j(v)) \in H)$  do
  if  $C(u)$  exists then
     $c_i \leftarrow C(u)$ 
  else
     $c_i \leftarrow \{u\}$ 
  end
  end
  if  $1 - S(u, v) < \epsilon$  then
    if  $C(v)$  exists then
       $c_i \leftarrow c_i \cup C(v)$  else
         $c_i \leftarrow c_i \cup \{v\}$ 
      end
    end
     $c_i \leftarrow C(v)$ 
     $C(u) \leftarrow c_i$ 
     $C(v) \leftarrow c_i$ 
  end
end
return  $C$ 

```

The inverse document frequency (idf) [37] represents the frequency of a property usage across all graph nodes, and it is defined as

$$idf(p, G) = \ln \frac{|V|}{|\{u \in V : p \in L(u)\}|}. \quad (10)$$

The property importance weights are based on the degree of distinctiveness of a property describing an entity. With property distinctiveness, we mean the uniqueness of a property in describing the key characteristics of an entity type. For instance, if a property is specific to an entity type, it is a distinguishing character of the type from other types. When a property exists in all entity types, its quality of being distinctive is low. The noise labels tend to be common for a majority of entities if not for all entities. By increasing importance weights of properties with a higher degree of distinctiveness, we reduce the importance of noise labels automatically. As a result, the noise labels have significantly less impact on the overall similarity measures.

3.4 Class Predicate Stability

In this work, the summary graph is built automatically from an RDF dataset. However, automatically generated summary graphs can be error prone. It is essential to have an effective metric to measure the degree of confidence of a relation between classes in the summary graph. We define this metric as Class

Predicate Stability (CPS), which is a similar notion to the concept of stability that introduced by Paige and Tarjan [31].

For u and v being IRIs in the dataset, $G = (V, E, L)$, and $u \in c_1$ and $v \in c_2$ with both c_1 and c_2 being type classes in the summary graph, $G' = (V', E', L')$, a class relation between the class c_1 and the class c_2 is generated as a predicate and represented as $l(c_1, c_2)$ if there is at least one relation $l(u, v)$. Consequently, $l \in L'$ and $l(c_1, c_2) \in E'$.

The CPS metric is calculated as the number of the IRI nodes u in class c_1 having a triple of the form (u, p, v) with $u \in c_1$ and $v \in c_2$ divided by the total number of the IRI nodes in c_1 in the summary graph such that the triple (c_1, p, c_2) is in the summary graph G' and c_1 and c_2 are type class IRI nodes with p being a predicate between them. $CPS(c_1, p, c_2)$ is formulated as

$$CPS(c_1, p, c_2) = \frac{|(u, p, v) : u \in c_1, v \in c_2|}{|c_1|} \quad (11)$$

where $|c_1|$ is the number of IRI nodes in the class c_1 . Note that $|c_1| > 0$.

The CPS value for a triple (c_1, p, c_2) indicates the degree of partitioning coarseness of the type classes c_1 and c_2 with the predicate p in the summary graph. Hence, the mean of all the CPS values in the summary graph is an indicator of accuracy for the generated summary graph. $CPS(G')$ is formulated as

$$CPS(G') = \frac{\sum_{i=1}^{|E'|} CPS(c_1^i, p^i, c_2^i)}{|E'|} \quad (12)$$

where $G' = (V', E', L')$ is the summary graph and $p^i(c_1^i, c_2^i) \in E'$, and thus $|E'| > 0$.

For two classes c_1 and c_2 in the summary graph, when either all the IRI nodes from c_1 are connected with a predicate p to at least one IRI node in c_2 or none of the IRI nodes in c_1 are connected with the predicate p to an IRI node in c_2 , we call that the classes c_1 and c_2 have full CPS.

3.5 Automatic Calculation of the Class Dissimilarity Thresholds

Our approach automatically builds the summary graph from RDF data. A drawback in the automatic summary graph generation approach is the need for estimating the optimum parameters that help determine the type classes. As expected, higher class dissimilarity threshold generates more coarse classes, whereas the classes become more granular when the threshold is chosen smaller. The optimum values for the class dissimilarity threshold depend on the characteristics of the datasets. In real-world datasets, users may not have a good grasp on the underlying data to determine optimal class dissimilarity threshold values.

To determine how closely the entities fit the type class, an effective metric is needed to measure the degree of fit within each type class in the summary graph. For this purpose, we utilize the root-mean-square deviation (RMSD), which is a commonly used measure of the differences between the values in comparison [24].

The root-mean-square deviation (RMSD) in RDF summary graphs

The RMSD represents the amount of the deviations of IRI node property values from the class center and provides a single measure of predictive power. In RDF summary graph, we calculate the overall RMSD by aggregating the sum of RMSD values for each type class in the summary graph.

To calculate the RMSD of summary graph, we first determine the centroids for each type class and then compute the RMSD between the class centroids and all IRI nodes within the type class using Manhattan distance. In RMSD calculation, the IRI node properties represent the dimensions of the IRI nodes within the type class. $RMSD(G')$ of summary graph G' is formulated as follows

$$RMSD(G') = \sum_{c_i \in G'} \sqrt{\frac{\sum_{(i=1) \in L'}^n (x_i - \bar{x})}{n}} \quad (13)$$

where c_i , L' are, respectively, the list of classes and the property labels in the summary graph G' . x_i represents the IRI nodes in type class and \bar{x} denotes the centroid for members of a particular type class in the summary graph G' .

Higher RMSD values in a summary graph indicate that entities within type classes sparsely located. When the entities in type classes are very similar to each other, the center of the cluster will be dense. Thus, the sum of distances to the centroids and the cumulative RMSD value will be lower accordingly.

Discovery of Class Dissimilarity Threshold

To discover the type classes in summary graph automatically, we propose a measure, called Favorability, to calculate the class dissimilarity threshold automatically as follows.

$$Favorability(G') = \max \left\{ \frac{Stability(G') * TypificationRate(G')}{(RMSD(G') + 0.1)} \right\} \quad (14)$$

The idea behind the formula is that we think that the quality of summary graph type classes is associated and directly proportional to the summary graph stability, a measure of relationships between type classes, and the ratio of entities belonging to a type class, and inversely proportional to the RMSD, the degree of inner class deviation.

Algorithm 3: *FindOptimumEpsilon*

```

input      : Similarity-Matrix  $S$ , Pairs  $H$ , Minimum-Threshold  $min_\epsilon$ ,
              Maximum-Threshold  $max_\epsilon$ , Number-of-try  $n$ , Previous-Favorability
               $prev\_favor$ , Previous-Optimum-Threshold  $prev\_optimum_\epsilon$ 
output    : Optimum-Threshold  $optimum_\epsilon$ 
parameter: Epsilon-Convergence-Threshold  $Ect$ 

 $current_\epsilon \leftarrow min_\epsilon$ 
 $inc \leftarrow (max_\epsilon - min_\epsilon)/n$ 
 $optimum_{favor} \leftarrow prev\_favor$ 
 $optimum_\epsilon \leftarrow prev\_optimum_\epsilon$ 
while  $current_\epsilon \leq max_\epsilon$  do
     $(G', C) \leftarrow CreateClasses(S, H, current_\epsilon)$ 
     $Favorability(G') = \frac{Stability(G') * TypificationRate(G')}{(RMSD(G') + 0.1)}$ 
    if  $Favorability(G') > optimum_{favor}$  then
         $optimum_{favor} \leftarrow Favorability(G')$ 
         $optimum_\epsilon \leftarrow current_\epsilon$ 
    end
     $current_\epsilon \leftarrow current_\epsilon + inc$ 
end
if  $|optimum_{favor} - prev\_favor| > Ect$  then
     $optimum_\epsilon \leftarrow FindOptimumEpsilon(S, H, optimum_\epsilon - inc, optimum_\epsilon +$ 
         $inc, n/2, optimum_{favor}, optimum_\epsilon)$ 
end
return  $optimum_\epsilon$ 

```

In the formula, $Favorability(G')$ is the class dissimilarity threshold for the summary graph G' and $TypificationRate(G')$ represents the rate of entities that belong to a type class based on the class dissimilarity threshold. The $TypificationRate(G')$ is low when the class dissimilarity threshold is too high since the number of entities satisfying high similarity threshold for class membership will be small.

To obtain the high quality results while reducing the computation cost, we gradually change the threshold values in constant number of times and set the class threshold value that provides the maximum value of the proposed measure. The algorithm 3 demonstrates how the optimum class dissimilarity threshold is discovered utilizing the favorability measure. In the algorithm, $optimum_\epsilon$ refers to the optimum class dissimilarity threshold for the summary graph.

The proposed automatic threshold discovery measure is not assumed to be perfect. Finding optimum summary graph type classes is a formidable problem as the quality of summary graph is dependent on the type of datasets. Despite this, the proposed measure integrates different aspects of the graph summaries and provides intuitively accurate graph summaries based on our evaluations.

4 Evaluations

In the evaluations, we conducted preliminary experiments on three datasets: a subset of DBpedia [3]; a subset of SemanticDB [13], and a subset of Lehigh University Benchmark (LUBM) [17]. Our experimental datasets are in different domains and they represent different aspects of real world semantic data.

SemanticDB is a Semantic Web content repository for Clinical Research and Quality Reporting in cardiovascular surgery domain. The structured entity type information exist in SemanticDB which we utilized as the ground truth for the verification of the algorithm. Lehigh University Benchmark (LUBM) is a well-known benchmark for OWL knowledge base systems, which also has entity type information available. But unlike SemanticDB, LUBM data has hierarchical types. Lastly, DBpedia a central source in the Linked Open Data Cloud [6] and is a commonly used general purpose dataset. However, using the entity type information for the verification of the algorithm is more problematic in DBpedia, as the type information may not present, or an entity may have several types including the hierarchical types. Therefore, we manually verified the results of the algorithm. Table 1 demonstrates a sample of RDF triples from each dataset in the evaluations.

4.1 Assessing Algorithm Parameters

We tested several parameters of the algorithm, including the maximum iteration, beta factor, class dissimilarity threshold, iteration convergence threshold (*Ict*), and the size of dataset in type generation. The results of our evaluations are demonstrated in Table 2. For verification, we extracted the ground truth, entity types present in the datasets, against the entity type classes generated by the algorithm. For the assessment of our evaluations, we used the measure of precision. Precision is defined as the ratio of correct results over all results.

The similarity computation algorithm stops the iterations, once the rate of change in the similarity measures drops below the threshold or once it reaches the maximum number of iterations. In our evaluations, we observed that the similarity measures typically converge after a few iterations with the values of the maximum number of iterations and the iteration convergence threshold being as 10 and 0.001, respectively.

4.2 Performance of dynamic assignment of descriptor weights

We also evaluated the performance of dynamic assignment of descriptor weights. Table 3 shows a sample of dynamically assigned descriptor weights from each dataset. As expected, the algorithm assigned higher weights to the properties with a higher degree of distinctiveness describing the resource type. For instance in LUBM dataset, takesCourse property is more descriptive of the Student type than the name property, which is a common property for all

Table 1 A Sample of RDF Triples from Each Dataset

Dataset	Subject	Predicate	Object
SemanticDB	SurgeryProcedure:236	SurgeryProcedureClass	"cardiac valve"
SemanticDB	SurgeryProcedure:236	CardiacValveEtiology	"other"
SemanticDB	SurgeryProcedure:236	belongsToEvent	Event:184
SemanticDB	SurgeryProcedure:236	SurgeryProcedureDescription	"pulmonary valve repair"
SemanticDB	SurgeryProcedure:236	CardiacValveStatusologyData	"native"
SemanticDB	SurgeryProcedure:104	SurgeryProcedureClass	"cardiac valve"
SemanticDB	SurgeryProcedure:104	CardiacValveEtiology	"rheumatic"
SemanticDB	SurgeryProcedure:104	belongsToEvent	Event:81
SemanticDB	SurgeryProcedure:104	SurgeryProcedureDescription	"mitral valve repair"
SemanticDB	SurgeryProcedure:104	CardiacValveStatus	"native"
LUBM	Student49	telephone	"xxx-xxx-xxxx"
LUBM	Student49	memberOf	http://www.Department3.University0.edu
LUBM	Student49	takesCourse	Course32
LUBM	Student49	name	"UndergraduateStudent49"
LUBM	Student49	emailAddress	"Student49@Department3.University0.edu"
LUBM	Student49	type	UndergraduateStudent
LUBM	Student10	telephone	"xxx-xxx-xxxx"
LUBM	Student10	memberOf	http://www.Department3.University0.edu
LUBM	Student10	takesCourse	Course20
LUBM	Student10	name	"UndergraduateStudent10"
LUBM	Student10	emailAddress	"Student10@Department3.University0.edu"
LUBM	Student10	type	UndergraduateStudent
DBPedia	Allen_Ginsberg	wikiPageUsesTemplate	Template:Infobox_writer
DBPedia	Allen_Ginsberg	influenced	John_Lennon
DBPedia	Allen_Ginsberg	influences	Fyodor_Dostoyevsky
DBPedia	Allen_Ginsberg	deathPlace	"New York City, United States"@en
DBPedia	Allen_Ginsberg	deathDate	"1997-04-05"
DBPedia	Allen_Ginsberg	birthPlace	"Newark, New Jersey, United States"@en
DBPedia	Allen_Ginsberg	birthDate	"1926-06-03"
DBPedia	Allen_Ginsberg	deathPlace	"New York City, United States"@en
DBPedia	Albert_Camus	wikiPageUsesTemplate	Template:Infobox_philosopher
DBPedia	Albert_Camus	influenced	Orhan_Pamuk
DBPedia	Albert_Camus	influences	Friedrich_Nietzsche
DBPedia	Albert_Camus	deathPlace	"Villeblevin, Yonne, Burgundy, France"@en
DBPedia	Albert_Camus	deathDate	"1960-01-04"
DBPedia	Albert_Camus	birthPlace	"Drean, El Taref, Algeria"@en
DBPedia	Albert_Camus	birthDate	"1913-11-07"

type classes in the dataset. Thus, takesCourse was assigned a weight of 44.1% as compared to the weight of 7.5% for name.

4.3 Effectiveness of the Automatic Computation of Class Thresholds

In a set of evaluations, we further assessed the effectiveness of automatic calculation of the class threshold approach using a subset of the same set of datasets. As demonstrated in Table 4, the stability, RMSD and optimum class threshold may vary depending on the characteristics of the datasets. In LUBM

Table 2 Evaluations of Algorithm Parameters

Dataset	#Triples	Class_Threshold	#Iterations	Stability	Precision
SemanticDB	6,450	0.5	4	61.0%	87.3%
LUBM	6,484	0.3	3	67.8%	90.7%
DBPedia	10,000	0.6	3	82.4%	92.8%

Table 3 An Excerpt from Dynamically Assigned Weights of Descriptors

Dataset	Node_Pair	Descriptor_Type	Descriptor	Weight
LUBM	(Student49,Student10)	Property	memberOf	14.7%
LUBM	(Student49,Student10)	Property	takesCourse	44.1%
LUBM	(Student49,Student10)	Property	emailAddress	14.0%
LUBM	(Student49,Student10)	Property	type	5.7%
LUBM	(Student49,Student10)	Property	name	7.5%
LUBM	(Student49,Student10)	Property	telephone	14.0%
SemanticDB	(Procedure:236,Procedure:104)	Literal	"cardiac"	13.6%
SemanticDB	(Procedure:236,Procedure:104)	Literal	"native"	15.2%
SemanticDB	(Procedure:236,Procedure:104)	Literal	"other"	14.3%
SemanticDB	(Procedure:236,Procedure:104)	Literal	"pulmonary"	22.8%
SemanticDB	(Procedure:236,Procedure:104)	Literal	"repair"	17.2%
SemanticDB	(Procedure:236,Procedure:104)	Literal	"valve"	16.9%
DBPedia	(Allen_Ginsberg,Albert_Camus)	Property	wikiPageUsesTemplate	2.2%
DBPedia	(Allen_Ginsberg,Albert_Camus)	Property	influences	58.3%
DBPedia	(Allen_Ginsberg,Albert_Camus)	Property	deathDate	2.2%
DBPedia	(Allen_Ginsberg,Albert_Camus)	Property	birthDate	2.4%
DBPedia	(Allen_Ginsberg,Albert_Camus)	Property	birthPlace	2.1%
DBPedia	(Allen_Ginsberg,Albert_Camus)	Property	deathPlace	2.1%
DBPedia	(Allen_Ginsberg,Albert_Camus)	Property	influenced	30.7%

dataset, the RMSD result was higher compared to the other datasets. Among them, the highest optimum class dissimilarity threshold, 0.56, was achieved in DBPedia dataset. This means that the entities within the type classes of the summary graph generated by the dataset contained similar properties and were very similar to the entities that belonged to the same type class. In our evaluations, we observed that the epsilon convergence threshold around 0.9 performed well.

Table 4 Automatic Calculation of the Class Thresholds Results

Dataset	#Triples	Optimum_Class_Threshold	RMSD	Stability
SemanticDB	1000	0.32	0.8	74.0%
LUBM	1000	0.26	4.8	83.0%
DBPedia	1000	0.56	0.9	82.8%

4.4 Generated Summary Graph

Figure 2 illustrates a small sample set of entities in the RDF graph from SemanticDB and their corresponding type classes in the summary graph. As demonstrated in Figure 2, the classes C-E1 and C-E2 represent the entities that are patient event types. They are classified in two different classes because when compared with the original dataset we observed that the entities in C-E1 are more specifically patient surgery-related event types while the entities in C-E2 are patient-encounter related event types. Also, the classes E-SP1 and E-SP2 are surgical procedure types. More specifically, the entities in E-SP1 are coronary artery and vascular procedure-related procedures while the entities in E-SP2 are cardiac valve related-procedures. The classes C-VP and C-CAG represent the entities that are related to vascular procedures and coronary artery grafts, respectively. We implemented a basic algorithm to name the classes based on the class member IRIs. The classes C-E1, C-E2, C-SP1, C-SP2, C-VP and C-CAG are named as C-Event-1, C-Event-2, C-SurgicalProcedure-1, C-SurgicalProcedure-2, C-VascularProcedure and C-CoronaryArteryGraft, respectively.

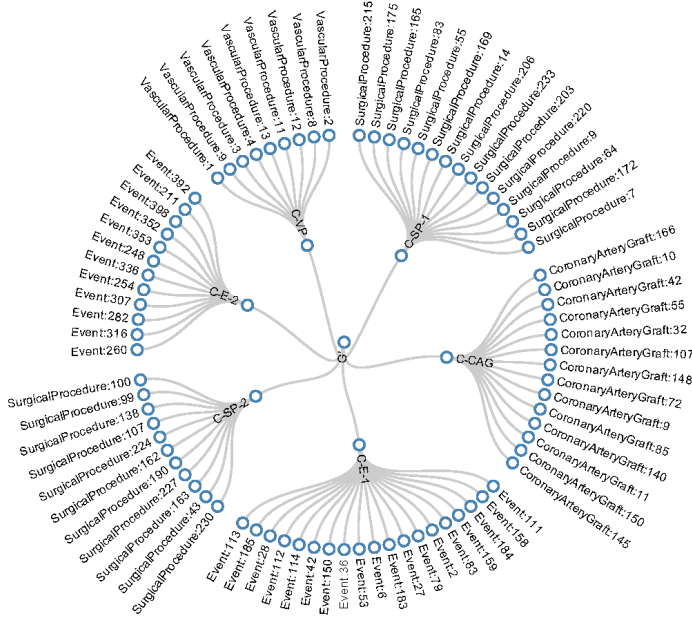


Fig. 2 A figure consisting of different types of entities and elements belonging to the type classes.

The summary graph is generated along with the classes and the class relations with a stability measure for each relation. Figure 3 shows an excerpt from the summary graph representing the class relations from SemanticDB

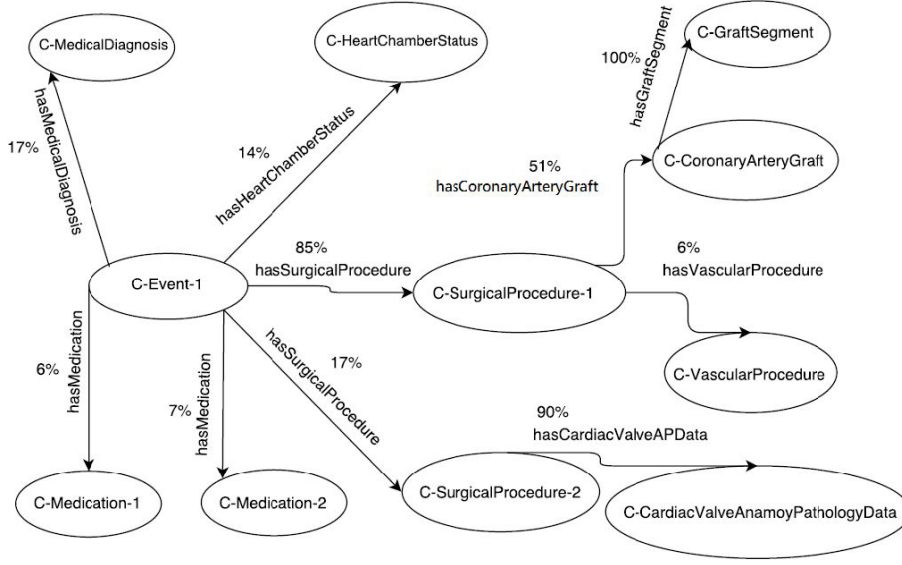


Fig. 3 An excerpt from the generated summary graph.

dataset. The percentage values beside the predicates are the stability (CPS) measure.

Overall, we observed that the class dissimilarity threshold ranging between 0.25 to 0.6 with the beta factor of 0.15 appeared to work well in our evaluations. The automatically calculated class dissimilarity threshold values during the evaluations were in close proximity of the threshold values for the datasets that were kept as the ground truth in the assessment.

5 Limitations of the method

The algorithm used for the similarity calculation runs in the n^2 in the worst case and in the $n(\log n)k$ time in average, where k is a constant number of iterations. For Web-scale usage, the scalability of the algorithm needs to be further improved as the size of the input RDF data can be very large. For instance, as of today, the Linking Open Data[6] project already contains more than 30 billions triples. In future work, we plan to address the performance issues for big datasets in the worst-case scenario and perform Web-scale evaluations.

Furthermore, the literal node similarity calculation currently does not perform well in cases where the literal nodes belong to different languages with disparate linguistic properties as we do not perform any linguistic analysis.

Also, in the current study, the classes in the summary graph are automatically named exploiting the frequent entity names and literal values that belong

to the related class. The naming method may not always generate the best names for human readers.

6 Related Work

The problem of Graph Summarization has been studied by various communities from different perspectives including Graph compression, graph partitioning, social network analysis, data visualization.

From the Graph Compression perspective, numerous approaches have explored the graph summarization problem with the aim of reducing the storage space of the large graph datasets [33,15,18,11]. Different from these approaches, we deal with labeled directed graphs as in the case of RDF. Also, a summary graph structure based on the original graph is generated in our method.

Several studies such as [28,10] have broadly investigated statistical methods to help understand the properties of large networks. These approaches provide useful information but they do not generate a summary graph from the graph data as it is the focus of our approach.

In the area of graph partitioning area, many methods have been introduced [29,42,39,43] to partition graph data into specific components. While these methods are helpful in discovering neighborhoods in large graph networks, they don't consider the similarities of the node properties. Tian et al. [39,43] proposed an aggregation-based graph summarization utilizing graph node attributes. However, the approach only deals with categorical node attributes and users need to group numerical attributes into categories manually, which is not feasible for large real-world datasets.

In the Semantic Web community, there has also been some related studies [8,23,40,5,16]. The studies in [8] and [16] are query driven graph summarization methods. They primarily focus on the problem of SPARQL query formulation over RDF data. The approaches using bisimulation [23,40] have a limitation to be applied in real-world datasets due to the exponential complexity of bisimulation.

Neighborhood-based similarity measures have been investigated by several studies including SimRank [20], SimRank++ [2], PageSim [25], MatchSim [26], PathSim [38], and Co-Citation [36]. Especially, SimRank is a widely-known measure, which utilizes the mean of the edge similarities between nodes. However, this may reduce the similarity score of similar graph nodes in a counter-intuitive manner when the nodes have multiple edges that differ in weights. On contrary, our method considers the maximal matching for calculating the similarity in a structural context.

Entity properties might have different impact on entity similarity scores. The weights of the entity properties can be determined using a similarity measure. There are some studies that try to calculate the property weights and apply them in similarity calculations such as [35,9]. But, they primarily focus on instance matching. In instance matching, the property weights yield

precedence to properties making the instances more unique. Contrary to instance matching, the properties that would help describe the entity types more distinctively are weighted higher in our approach. In [9], they determine the property weights using the distinct value based weight generation and assign higher weight to a property that references more distinct values. However, a training set of instances may not always be available.

7 Conclusion

In this paper, we have investigated the main aspects for graph summary problem in RDF graphs. We described our pairwise graph node similarity calculation with the addition of the property and string word importance weights, along with the Class Predicate Stability metric, which allows evaluation of the degree of confidence of each class predicate in the summary graph. Furthermore, we studied obtaining the optimum value of the class dissimilarity threshold automatically in RDF summary graphs. Based on our investigations, a measure to determine optimum class dissimilarity thresholds and an effective method to discover the type classes automatically were introduced. Using a set of real-world datasets, we assessed the effectiveness of our automatic summary graph generation approach. For future work, we plan to focus on the scalability of the proposed method in very large datasets.

Acknowledgements The authors would like to thank Prof. Austin Melton for his invaluable help and his guidance during the study, Dr. Ruoming Jin and Dr. Viktor Lee for sharing RoleSim similarity measure.

References

1. Adida, B., Birbeck, M., McCarron, S., Pemberton, S.: RDFa in XHTML: Syntax and processing. Recommendation, W3C (2008)
2. Antonellis, I., Molina, H.G., Chang, C.C.: Simrank++: Query rewriting through link analysis of the click graph. *Proceedings of the VLDB Endowment* **1**(1), 408–421 (2008)
3. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: *Dbpedia: A nucleus for a web of open data*. Springer (2007)
4. Aydar, M., Ayvaz, S., Melton, A.C.: Automatic weight generation and class predicate stability in rdf summary graphs. In: *Workshop on Intelligent Exploration of Semantic Data (IESD2015)*, co-located with ISWC2015, vol. 1472 (2015)
5. Ayvaz, S., Aydar, M., Melton, A.: Building summary graphs of rdf data in semantic web. In: *Computer Software and Applications Conference (COMPSAC)*, 2015 IEEE 39th Annual, vol. 2, pp. 686–691 (2015). DOI 10.1109/COMPSAC.2015.107
6. Bizer, C., Heath, T., Berners-Lee, T.: Linked data-the story so far. *International journal on semantic web and information systems* **5**(3), 1–22 (2009)
7. Brickley, D., Guha, R.V.: *RDF Schema 1.1. W3c Recommendation* (2014). URL <http://www.w3.org/TR/2014/REC-rdf-schema-20140225/>
8. Campinas, S., Perry, T.E., Ceccarelli, D., Delbru, R., Tummarello, G.: Introducing rdf graph summary with application to assisted sparql formulation. In: *2012 23rd International Workshop on Database and Expert Systems Applications*, pp. 261–266. IEEE (2012)

9. Castano, S., Ferrara, A., Montanelli, S., Lorusso, D.: Instance Matching for Ontology Population. In: SEBD, pp. 121–132 (2008)
10. Chakrabarti, D., Faloutsos, C.: Graph mining: Laws, generators, and algorithms. *ACM computing surveys (CSUR)* **38**(1), 2 (2006)
11. Chierichetti, F., Kumar, R., Lattanzi, S., Mitzenmacher, M., Panconesi, A., Raghavan, P.: On compressing social networks. In: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 219–228. ACM (2009)
12. Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1 Concepts and Abstract Syntax. W3c Recommendation (2014). URL <http://www.w3.org/TR/rdf11-concepts/section-IRIs>
13. D Pierce, C., Booth, D., Ogbuji, C., Deaton, C., Blackstone, E., Lenat, D.: Semanticdb: A semantic web infrastructure for clinical research and quality reporting. *Current Bioinformatics* **7**(3), 267–277 (2012)
14. Duan, S., Kementsietsidis, A., Srinivas, K., Udrea, O.: Apples and oranges: a comparison of rdf benchmarks and real rdf datasets. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, pp. 145–156. ACM (2011)
15. Fan, W., Li, J., Wang, X., Wu, Y.: Query preserving graph compression. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pp. 157–168. ACM (2012)
16. Goasdoué, F., Manolescu, I.: Query-oriented summarization of rdf graphs. *Proceedings of the VLDB Endowment* **8**(12) (2015)
17. Guo, Y., Pan, Z., Heflin, J.: Lubm: A benchmark for owl knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web* **3**(2), 158–182 (2005)
18. He, X., Kao, M.Y., Lu, H.I.: A fast general methodology for information-theoretically optimal encodings of graphs. *SIAM Journal on Computing* **30**(3), 838–846 (2000)
19. Jain, A.K., Dubes, R.C.: Algorithms for clustering data. Prentice-Hall, Inc. (1988)
20. Jeh, G., Widom, J.: SimRank: a measure of structural-context similarity. In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 538–543. ACM (2002)
21. Jin, R., Lee, V.E., Hong, H.: Axiomatic ranking of network role similarity. In: Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 922–930. ACM (2011)
22. Khare, R., elik, T.: Microformats: a pragmatic path to the semantic web. In: Proceedings of the 15th international conference on World Wide Web, pp. 865–866. ACM (2006)
23. Khatchadourian, S., Consens, M.P.: Explod: summary-based exploration of interlinking and rdf usage in the linked open data cloud. In: Extended Semantic Web Conference, vol. 272–287, pp. 272–287. Springer (2010)
24. Levinson, N.: The wiener (root mean square) error criterion in filter design and prediction. *Journal of Mathematics and Physics* **25**(1), 261–278 (1946)
25. Lin, Z., Lyu, M.R., King, I.: Pagesim: a novel link-based measure of web page aimilarity. In: Proceedings of the 15th international conference on World Wide Web, pp. 1019–1020. ACM (2006)
26. Lin, Z., Lyu, M.R., King, I.: Matchsim: a novel neighbor-based similarity measure with maximum neighborhood matching. In: Proceedings of the 18th ACM conference on Information and knowledge management, pp. 1613–1616. ACM (2009)
27. Luhn, H.P.: A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of research and development* **1**(4), 309–317 (1957)
28. Newman, M.E.: The structure and function of complex networks. *SIAM review* **45**(2), 167–256 (2003)
29. Newman, M.E., Girvan, M.: Finding and evaluating community structure in networks. *Physical review E* **69**(2), 026,113 (2004)
30. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: Bringing order to the web. Stanford InfoLab (1999)
31. Paige, R., Tarjan, R.E.: Three partition refinement algorithms. *SIAM Journal on Computing* **16**(6), 973–989 (1987)
32. Parundekar, R., Knoblock, C.A., Ambite, J.L.: Discovering concept coverings in ontologies of linked data sources. In: International Semantic Web Conference, pp. 427–443. Springer (2012)

33. Raghavan, S., Garcia-Molina, H.: Representing web graphs. In: Data Engineering, 2003. Proceedings. 19th International Conference on, pp. 405–416. IEEE (2003)
34. Rajaraman, A., Ullman, J.D.: Mining of massive datasets. Cambridge University Press (2011)
35. Seddiqui, M.H., Nath, R.P.D., Aono, M.: An Efficient Metric of Automatic Weight Generation for Properties in Instance Matching Technique. *International Journal of Web & Semantic Technology* **6**(1), 1 (2015)
36. Small, H.: Co-citation in the scientific literature: A new measure of the relationship between two documents. *Journal of the American Society for information Science* **24**(4), 265–269 (1973)
37. Sparck Jones, K.: A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation* **28**(1), 11–21 (1972)
38. Sun, Y., Han, J., Yan, X., Yu, P.S., Wu, T.: Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *VLDB11* (2011)
39. Tian, Y., Hankins, R.A., Patel, J.M.: Efficient aggregation for graph summarization. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pp. 567–580. ACM (2008)
40. Tran, T., Ladwig, G.: Structure index for rdf data (2010)
41. Tran, T., Wang, H., Rudolph, S., Cimiano, P.: Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In: Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on, pp. 405–416. IEEE (2009)
42. Xu, X., Yuruk, N., Feng, Z., Schweiger, T.A.: Scan: a structural clustering algorithm for networks. In: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 824–833. ACM (2007)
43. Zhang, N., Tian, Y., Patel, J.M.: Discovery-driven graph summarization. In: 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010), pp. 880–891. IEEE (2010)