Dynamic Discovery of Type Classes and Relations in Semantic Web Data

Serkan Ayvaz1 • Mehmet Aydar2

Received: 22 January 2018 / Revised: 30 November 2018 / Accepted: 13 February 2019 / Published online: 20 February 2019
© Springer-Verlag GmbH Germany, part of Springer Nature 2019

Abstract
With the rapidly growing resource description framework (RDF) data on the Semantic Web, processing large semantic graph data has 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 based on instance similarities. To scale our approach, we utilized locality-sensitive hashing technique for identifying instance pairs which are candidates to be in the same type class. 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 have been significant developments in publishing data with expressed semantics in the Web. The Linking Open Data [9] and similar community projects have recommended the publication of large amount of globally useful datasets in machine-readable forms. Moreover, the utilization of Resource Description Framework (RDF), along with other forms of semantic data in forms of RDFa [1] and microformats [31] 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. An RDF triple consisting object type property 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 do not 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 understanding the conceptual summary of the raw data, and for Semantic Search algorithms in terms of query time since the entire input data do not need to be completely processed at the query time. We call this structure as Summary Graph [7,8]. Semantic search is a common graph processing task, which often requires an index structure for effective and faster graph processing. For instance, the semantic search approach proposed by [54] uses a sum-
mary 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 exist 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 (rdfs:type) in the dataset and organize the type classes and relations accordingly, if the dataset is published using a standard vocabulary [20]. (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. Rdfs: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 [43]. Thus, the methods, which rely on rdfs: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.

2 Defining Graph Summarization Problem

RDF data are a labeled, directed multi-graph that consists of a collection of statements. RDF statements are expressed unambiguously in the form of subject-predicate-object expressions that describe a resource, a relationship between two resources, or the type of a resource (type triple) [18].

Each RDF node corresponding to an RDF entity that describes a resource is represented with an IRI. In RDF data, the literal values including strings, numbers and dates, are represented by literal nodes. A property of the RDF subject node is also used interchangeably with a predicate in an RDF triple. A predicate in an RDF triple can be one of two types: a DatatypeProperty or an ObjectProperty. DatatypeProperty is a predicate between an IRI as the subject of the triple and a literal as the object of the triple. In ObjectProperty, both the subject and object of the triple are IRIs. In RDF graphs, each object node is also called a neighbor of that subject node in the triple.
The subject in an RDF triple can be either an IRI or a blank node. The predicate must be an IRI. The object in the RDF triple is either an IRI, a literal or a blank node. The nodes in the RDF graph consist of the subjects and objects of triples in RDF data. As an example, Figure 1 represents two sample university entities Kent_State and Case_Western and their properties.

Formally, RDF graph \( G = (V, L, E) \) is a directed labeled graph, where the set of vertices \( V \) represents entities (resources), the set of directed edges \( E \) of the form \( l(u, v) \), with \( u \in V, v \in V \). \( l \in L \) denote predicates (properties) between entities, and the labels \( L \) are labels or predicate names. An edge \( l(u, v) \) represents the RDF triple \( (u, l, v) \).

A summary graph is the directed graph, in which each node is a subset of the original data 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 \( v \).” as stated in [8]. 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 \( V \). Respectively, \( L' \) and \( E' \) in the graph \( G' \) are the sets of labels and edges. Consequently, the elements in the edges in \( E \) and equivalence classes in \( V' \) define the elements of \( E' \) and \( L' \subseteq L \).

For vertices \( u, v \in V \), there are \([u], [v] \in V' \). Also, if there is an edge \( l([u], [v]) \in E' \), then there must exist \( t \in [v] \subseteq V \) and \( s \in [u] \subseteq V \) such that \( l(t, s) \in E \).

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 [7,8]. This current study extends our summary graph generation approach [8] and enhances it by pre-determining RDF subject node pairs which are candidates to be in the same type classes using locality-sensitive hashing, 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 impacts 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 is 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 [10]. 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.

We make use of string similarities for the lexical form components of the literal nodes that measure 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.

\[
\text{LiteralSim}(x, y) = \frac{\sum_{i \in (x \cap y)} |t_i| \times w_i}{\sum_{j \in (x \cup y)} |t_j| \times w_j}
\]

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 \((tf - idf)\) [36,50], 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 [47].

### 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 [29] in conjunction with the Jaccard measure context [27].

**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 [27].

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

\[
J(S1, S2) = \frac{|S1 \cap S2|}{|S1 \cup S2|}
\]
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 determines 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 is 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 [29]:

\[
\text{RoleSim}(u, v) = (1 - \beta) \times \max_{M \in Mm(u,v)} \frac{\sum_{(x,y) \in M} \text{RoleSim}(x, y)}{N_u + N_v - |M|} + \beta \tag{3}
\]

\( \text{RoleSim}(u, v) \) denotes the similarity of the nodes \( u, v \in V \). The definition of RoleSim is recursive, i.e., \( \text{RoleSim}(x, y) \) is calculated the same way as \( \text{RoleSim}(u, v) \). \( N(u) \) 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. Figure 2 illustrates a running example of determining \( M(u,v) \), the maximal matching value. \( M(u,v) \) is calculated as

\[
M(u,v) = \max_{M \in Mm(u,v)} \frac{\sum_{(x,y) \in M} \text{RoleSim}(x, y)}{\max(N_u, N_v)} \tag{4}
\]

The parameter \( \beta \) is a decay factor, \( 0 < \beta < 1 \). The parameter \( \beta \) 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 us 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 is weighted by each common predicate.

The reason for using RoleSim in neighborhood similarity calculation of our approach is its maximal weight matching mechanism, which is powerful in discovering graph nodes having similar roles, namely the type classes, as it enables RoleSim to assign higher similarity scores to nodes belonging to the same type classes. As shown in Fig. 3, RoleSim outweighs the state-of-the-art graph similarity measures in the average ranking of all pairs of nodes within the same roles [29].

We note, however, that the generic version of the RoleSim measure is introduced for the unlabeled graph. In this work, the input data are in RDF model, which is a directed and
labeled graph. Therefore, we utilize the RoleSim 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 ≤ i ≤ j, li is a label for an edge, i.e., li ∈ L. When 1 ≤ h ≤ j and if i and h are not equal, then lh and lh are different labels, i.e., li and lh are different properties. [xi] and [yi] are the sets of nodes which are related to u and v, respectively, by predicate li.

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

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

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:

\[
PairSim(u, v)^k = (1 - \beta) \times \frac{1}{|u \cup v|} \times \left( \sum_{l \in (u,v)} \max_{M \in M^j(u,v)} \frac{\sum_{(x,y) \in M} Sim(x,y)^{k-1}}{N_d^u + N^l - |M|} \times w_j \right) + \beta
\]

where k is the iteration number 1 ≤ 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^l(u) and N^l(v) denote their respective neighborhoods that are reached by jth common edge. x ∈ N^l(u) and y ∈ N^l(v), and N^l_u and N^l_v denote their respective degree connected by jth common edge. Said differently, N^l(u) is the cardinality of [x_j], and N^l(v) 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).

We define M to be a set of ordered pairs (x, y) where x ∈ N^l_u and y ∈ N^l_v such that there does not exist (x', y') ∈ 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. \(M^j(u, v)\) is the set of all such M’s. \(M^j(u, v)\) 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 \(N^l_u\) and \(N^l_v\) with the restriction that no element in either \(N^l_u\) and \(N^l_v\) may be used in more than one ordered pair.

The parameter \(\beta\) 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 \(l_1 = l_2, x \in N^l_u\) and \(y \in N^l_v\).

As a simple illustrative running example which is used to demonstrate technical notion of our similarity metric PairSim, in Eq. 5, consider the similarity of the sample nodes in Fig. 1. At the initial state, the similarity of the two nodes Kent State and Case_Western is 1 as illustrated below. Running through the iterations, the similarity values get updated based on corresponding neighbor similarities and eventually converge to final values.

\[
PairSim(u, v)^k = (1 - \beta) \times \frac{1}{|u \cup v|} \times \left( \sum_{l \in (u,v)} \max_{M \in M^j(u,v)} \frac{\sum_{(x,y) \in M} Sim(x,y)^{k-1}}{N_d^u + N^l - |M|} \times w_j \right) + \beta
\]

\[
PairSim(u, v)^1 = (1 - 0.15) \times \frac{1}{10} \times \left( \frac{1}{1 + 1 - 1} + \frac{1}{1 + 1 - 1} + \frac{1}{1 + 1 - 1} + \frac{1}{1 + 2 - 1} + \frac{1}{1 + 1 - 1} + \frac{1}{1 + 1 - 1} \right) + 0.15
\]

\[
PairSim(u, v)^1 = 0.85 \times \frac{1}{10} \times (6 + 0.5 + 0.5) + 0.15 = 0.79
\]
3.2 The Summary Graph Generator Algorithm

3.2.1 Iteration Technique for Neighborhood Similarity

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 memory allocation and substantial resource overhead. 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 [40]. The initial similarity of a node pair is set to 1 if they share a common predicate and 0, otherwise. Such that:

$$\forall (u, v \in V) :$$

$$(S(u, v) = 1) \rightarrow (|u \cap v| > 0) \text{ and,}$$
$$(S(u, v) = 0) \rightarrow (|u \cap v| = 0)$$

3.2.2 Pairing Subject Nodes for Similarity Calculations

Since our technique is based on entity pair similarities, it is important to determine how to pair the subject nodes. A regular Brute Force pairwise comparison of the subject nodes requires $O(n^2)$ complexity as all possible pairs must be considered when generating pairs, which is the Cartesian product of all the subject nodes. $O(n^2)$ complexity is not feasible in terms of scalability for large-scale data as order of magnitude comparisons would take place. For instance, Brute Force method would require an order of $10^{12}$ comparisons for an RDF graph which has millions of subject nodes.

The first approach that we used for candidate node pairing was to pair subject nodes, which share at least one common predicate. As Algorithm 1 generates pairs if two candidate nodes that share at least one common predicate, the overall complexity for the entire algorithm is $O(n^2)$ 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 are referenced by most if not all the subject nodes, the algorithm performs better than $O(n^2)$. Thus, the complexity of the algorithm depends on the characteristics of the dataset. On a dense graph, the complexity approaches to $O(n^2)$ while it gets near to $n(log n)k$ time in sparse graphs, where $k$ is a constant number of iteration. In our evaluations, while this approach worked well for small datasets, it did not scale for large datasets.

To improve scalability of the algorithm for large datasets, we utilized a MinHash [11] approach for pairing subject nodes for similarity calculations. MinHash is a Locality-Sensitive Hashing(LSH) technique for quickly estimating similarity of two sets. MinHash is based on the intuition that given a set of locality-sensitive hash functions, the probability of two sets having the same hash value is equal to the Jaccard similarity of the sets. For an RDF graph when applied to MinHash, the subject nodes are the sets and the predicates of the subject node along with the string literals referenced by the subject node are the set elements [6]. In our evaluations, we observed that this approach scales well for large datasets, i.e., for RDF graphs consisting millions of triples.

3.2.3 The Algorithm

The algorithm is twofold. Firstly, the pairwise similarity algorithm calculates the similarity values for each pair which constructs a similarity matrix as demonstrated in Algorithm 1. Once the pairwise similarities converge, the type class generation algorithm as shown in Algorithm 2 begins generating the type classes. More precisely, it assigns nodes $u$ and $v$ to the same type class if their dissimilarity value is less than an auto-calculated $\epsilon$ threshold which is the class dissimilarity threshold. We note that the dissimilarity metric is based on equivalence relation, which holds reflexivity, symmetry, and transitivity properties. For instance, if two elements, $u$ and $v$ are related by equivalence relation, then $\text{dissimilarity}(u, u) = 0$ by reflexivity, $\text{dissimilarity}(u, v) = \text{dissimilarity}(v, u)$ by symmetry, and $\text{dissimilarity}(u, v) = \text{dissimilarity}(v, w)$ then $\text{dissimilarity}(v, w) = \text{dissimilarity}(u, w)$. In other words, if $u$ and $v$ have low dissimilarity and $v$ and $w$ have low dissimilarity, then $u$ and $w$ should also have low dissimilarity.

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 $\beta$ 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 depends upon their descriptor similarities and the similarities of their neighbors.

The weight of each descriptor may vary significantly as each descriptor may have different impacts 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 tf-idf to the properties and nodes in RDF graphs. tf-idf is calculated as follows:

\[
tf-idf(p, u, G) = tf(p, u) \times idf(p, G).
\]

where the term frequency (tf) [36] 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

\[
f(p, u) = |\{v \in V : p(u, v) \in E\}|.
\]

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,

\[
tf(p, u) = \frac{f(p, u)}{\sum_{q \in L(u)} f(q, u)}.
\]

The inverse document frequency (idf) [50] represents the frequency of a property usage across all graph nodes, and it is defined as

\[
idf(p, G) = \ln \frac{|V|}{|\{u \in V : p \in L(u)\}|}.
\]

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 was introduced by Paige and Tarjan [41].

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. 

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

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

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

where $G' = (V', E', L')$ is the summary graph and $p'(c_1', c_2') \in E'$, 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 [33].

#### 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

$$RMSD(G') = \sum_{c_i \in G'} \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}} \quad (13)$$

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 are 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**

As there could potentially be more than one summary graph generated by an input data graph, the quality of the obtained summary graphs ought to be assessed. To discover the type classes in summary graph optimally, we propose an intuitive measure, called Favorability, to calculate the class dissimilarity threshold automatically as follows.

\[
Favorability(G') = \max \left\{ \frac{Stability(G') \times TypificationRate(G')}{(RMSD(G') + 0.1)} \right\}
\]

(14)

**Algorithm 3: FindOptimumEpsilon**

| input | : Similarity-Matrix S, Pairs H, Minimum-Threshold \(min_c\), Maximum-Threshold \(max_c\), Number-of-try \(n\), Previous-Favorability \(prev\_favor\), Previous-Optimum-Threshold \(prev\_optimum_c\) |
|-------|-------------------------------------------------------------|
| output| : Optimum-Threshold \(optimum_c\) |
| parameter | : Epsilon-Convergence-Threshold \(Ect\) |
| current\(_c\) | ← \(min_c\) |
| inc | ← \((max_c - min_c)/n\) |
| optimum\(_{favor}\) | ← \(prev\_favor\) |
| optimum\(_{optimum}\) | ← \(prev\_optimum_c\) |

**while** \(current_c \leq max_c\) **do**

\[
Favorability(G') = \frac{Stability(G') \times TypificationRate(G')}{(RMSD(G') + 0.1)}
\]

**if** \(Favorability(G') > optimum\(_{favor}\)\) **then**

\[
optimum\(_{favor}\) \leftarrow Favorability(G')
\]

**end**

\(current_c \leftarrow current_c + inc\)

**end**

**if** \(\left| (optimum\(_{favor}\) - prev\_favor) \right| > Ect\) **then**

\[
optimum_c \leftarrow \text{FindOptimumEpsilon}(S, H, optimum_c - inc, optimum_c + inc, n/2, optimum\(_{favor}\), optimum\(_{optimum}\))
\]

**end**

**return** optimum\(_c\)

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 types 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 [5], a subset of SemanticDB [19], and a subset of Lehigh University Benchmark (LUBM) [24]. SemanticDB is a Semantic Web content repository for Clinical Research and Quality Reporting in cardiovascular surgery domain. Lehigh University Benchmark (LUBM) is a well-known benchmark for OWL knowledge base systems. Lastly, DBPedia a central source in the Linked Open Data Cloud [9] and is a commonly used general purpose dataset. Our experimental datasets are in different domains, and they represent different aspects of real-world semantic data. Table 1 demonstrates a sample of RDF triples from each dataset in the evaluations.

The source code of our system and datasets used in the evaluations are available for use for research purposes.¹

As LUBM is repeatable synthetic benchmark data, it can be produced in varying sizes. SemanticDB is a real-world dataset and was developed in modular form. In the evaluations involving LUBM and SemanticDB datasets, we chose smaller forms of the whole RDF graphs for manually observation of the results and verification purposes. For testing the scalability of our approach, we used partial RDF graph data from DBPedia in varying sizes. When selecting partial RDF graph, we extracted triples in various sizes from DBPedia by sequentially reading triples in full dumb file.

¹ http://bit.ly/rdfsummarizer.
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:104     | 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”                       |
| 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                                    | “New York City, United States”@en           |
| DBPedia   | Allen_Ginsberg           | birthDate                                     | “1926-06-03”                                 |
| 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”                                 |

4.1 Assessing Algorithm Parameters

We tested several parameters of the algorithm, including the maximum iteration, beta factor, class dissimilarity threshold, iteration convergence threshold (Ic_{i}), and the size of dataset in type generation. The results of our evaluations are demonstrated in Table 2.

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.
Table 2 Evaluations of algorithm parameters

| Dataset    | #Triples | Class_Threshold | #Iterations | Stability (%) | Precision (%) |
|------------|----------|-----------------|-------------|---------------|---------------|
| SemanticDB | 6450     | 0.5             | 4           | 61.0          | 87.3          |
| LUBM       | 6484     | 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        | influenced       | 30.7       |

4.2 Comparative Analysis Against the Ground Truth

The structured entity type information exists in SemanticDB which we utilized as the ground truth for the verification of the algorithm. Lehigh University Benchmark (LUBM) dataset also has entity type information available. As for the Dbpedia, we extracted the structured entity type information from the Dbpedia ontology for our data sample.

For verification, we compared entity types and type members present as in the ground truth, against the entity type classes and class members 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 precision results of our evaluations are also demonstrated in Table 2.

4.3 Performance of Dynamic Assignment of Descriptor Weights

In the evaluations, the performance of dynamic weight assignments was also assessed. An excerpt of dynamically assigned weights from each dataset is demonstrated in Table 3. The properties with a higher degree of distinctiveness describing the resource type are assigned to higher weights by the algorithm as anticipated. For example, takesCourse property in LUBM dataset is more descriptive of the Student type than the name property as it is a commonly used property for all type classes in the dataset. In comparison with the weight of 7.5% for name, takesCourse was thus assigned a much higher weight of 44.1%.

4.4 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. The class threshold values in Table 4 were determined based on optimal Favorability measures. As demonstrated in Table 4, the stability, RMSD and optimum class threshold may vary depending on the characteristics of the datasets. In LUBM 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.

| Dataset     | Optimum class threshold | RMSD | Stability (%) |
|-------------|-------------------------|------|--------------|
| SemanticDB  | 0.32                    | 0.8  | 74.0         |
| LUBM        | 0.26                    | 4.8  | 83.0         |
| DBPedia     | 0.56                    | 0.9  | 82.8         |

### 4.5 Generated Summary Graph

A small sample set of RDF entities from SemanticDB and the corresponding type classes from the generated summary graph are shown in Fig. 4. The classes C-E1 and C-E2 represent the entities that correspond to patient event types as illustrated in Fig. 4. The entities in C-E1 and C-E2 are grouped in two different type classes. The classification appears to be correct as we observed when exploring the original dataset that the entities in C-E1 are particularly patient surgery-related event types, whereas the entities in C-E2 are more specifically patient-encounter-related event types. Furthermore, the surgical procedure types are classified into two different classes C-SP1 and C-SP2. When observed in detail, the entities in C-SP1 appear to be coronary artery and vascular procedure-related procedures. On the other hand, the entities in C-SP2 represent cardiac-valve-related procedures. The entities in the classes of C-CAG...
and C-VP represent, respectively, coronary artery grafts and vascular procedures. Naming the classes in dynamically generated summary graph intuitively is beneficial for human observers. Thus, we developed a basic algorithm that gives names to the classes based on the IRIs of the class members. For instance, the algorithm generated the class names of C-E1, C-E2, C-SP1, C-SP2, C-VP and C-CAG in the summary graph for the entities belonging to C-Event-1, C-Event-2, C-SurgicalProcedure-1, C-SurgicalProcedure-2, C-VascularProcedure and C-CoronaryArteryGraft, respectively.

The generated summary graph is composed of the type classes and the class relations along with a stability measure of each relation. A small sample from the summary graph representing the class relations based on SemanticDB dataset is depicted in Fig. 5. The stability (CPS) measure values are demonstrated by the percentage values beside the predicates.

In our evaluations, the algorithm parameters consisting of a class dissimilarity value between 0.25 and 0.6 along with the beta factor of 0.15 typically performed well. We also observed that the automatically computed class dissimilarity threshold values were the same as or similar to the ground truth threshold values of the datasets in the evaluations.

### 4.6 Summary Graph Pruning

For evaluating potential performance improvements in the produced summary graph, we assessed the amount of triples pruned from processing in original data graph. Table 5 presents the evaluation results for summary graph triple pruning.

| Dataset   | DataGraphSize | SummaryGraphSize | PrunedTriples | PruneRatio (%) |
|-----------|---------------|------------------|---------------|----------------|
| SemanticDB| 6450          | 28               | 6422          | 99.56          |
| LUBM      | 6484          | 29               | 6455          | 99.56          |
| DBpedia   | 10,000        | 2950             | 7050          | 70.47          |
| DBpedia   | 100,000       | 3401             | 96,599        | 96.60          |
| DBpedia   | 250,000       | 10,237           | 239,763       | 95.91          |
| DBpedia   | 500,000       | 22,220           | 477,780       | 95.56          |
| DBpedia   | 1,000,000     | 38,468           | 961,532       | 96.15          |
| DBpedia   | 5,000,000     | 164,830          | 4,835,170     | 96.70          |
Table 6: Comparison of summary graph size reduction rates

| Methods            | LUBM (%) | SemanticWebDogFood (%) |
|--------------------|----------|-------------------------|
| Bisimulation \(k = 1\) | 99.9     | 96.0                    |
| Bisimulation \(k = 2\) | 99.8     | 60.0                    |
| Bisimulation \(k = 3\) | 99.7     | 35.0                    |
| Bisimulation \(k = 4\) | 99.5     | 26.0                    |
| Bisimulation \(k = 5\) | 99.5     | 25.0                    |
| Bisimulation \(k = 6\) | 99.5     | 24.0                    |
| Our Algorithm      | 99.6     | 92.1                    |

demonstrates the pruning ratio and the number of triples pruned from original data graph.

The summary graph is very small compared to the original data graph (0.4–4.5%). We observed that the pruning rates for SemanticDB and LUBM datasets are approximately 99%. These datasets are more structured in nature. On the other hand, DBpedia is a more sparse dataset. It contains many properties, and majority of the entities in DBpedia only use some of the properties. Despite this, out of 5,000,000 triples in DBpedia data graph, the summary graph only consisted 164,830 triples with pruning rate of 96.7. Higher pruning rates were observed in larger input graphs.

The summary graph pruning results also indicate that the potential performance improvements gained by pruning triples from repeated processing, e.g., semantic search, query processing, can exceed by far the preprocessing overhead incurred by the summary graph generation as more than 96 percent pruning rates were observed in average.

Furthermore, we conducted a comparative evaluation of our approach against the state-of-the-art bisimulation-based graph summarization [2] to assess the summary graph pruning rates of the algorithms. The evaluation results are presented in Table 6. Similar to our algorithm, the bisimulation algorithm results in approximately 99.5% reduction of summary graph size in synthetic benchmark dataset LUBM for various \(k\) where \(k\) indicates the partitions for \(k\)-bisimulation. However, bisimulation approach achieves only around 24–60% reduction in summary graph, whereas our approach results in 92.1% reduction in Semantic Web Dog Food [37], which is a real-world dataset about conference presenters at Semantic Web conferences. This example demonstrates the limitations of bisimulation-based graph summarization as it only considers the structure of the entities in RDF graphs. In a dataset where the entities are not represented in the same structure, many bisimulation blocks end up containing a few elements [2]. On the other hand, our approach takes into account the semantic similarity of entities in graph summarization and is not restricted to structural similarity. Therefore, even when the entities are not in the same structural representation, our approach can infer the semantic similarities between entities when compressing the original graph and consequently performs better in summary graph size reduction.

### 4.7 Scalability of summary graph generation approach

To evaluate the scalability of our method, we assessed the performance of the algorithm on DBpedia dataset with varying sizes. We note that, for pairing the subject nodes, we used the MinHash method described in Sect. 3.2.2 as it scales well for big datasets. The results indicated that the performance of the algorithm scales well with the size of the input graph data as shown in Table 7. The time for generating summary graph appears to be increasing faster than the input graph size as demonstrated in Fig. 6. This is due to the fact that the number of entity pairs to be considered in the similarity calculations increases more rapidly compared to the input graph size.

#### 5 Potential Limitations

For Web-scale usage, the scalability of the algorithm still 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 [9] project already contains more than 30 billion 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.

Although some approaches addressing different problems in related fields, e.g., community detection, graph partitioning, entity clustering, are not directly focusing on RDF graph summarization, they may still be applicable to the problem. In this study, we have not conducted extensive evaluations against such approaches. However, we plan to explore the applicability of those approaches to graph summarization problem and perform assessments in comparison with our proposed approach in the future.

Using summary graph structure over RDF graphs is known to improve the query performance up to orders of magnitude [17,54]. However, we have not integrated the constructed summary graphs into a semantic search system similar to [54] as it was considered out of the scope of this paper. In future work, we plan to develop a semantic search system that integrates automatically obtained summary graphs based on the proposed algorithm.

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 [15,21,25,46]. 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 [14,38] 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.

Although the graph partitioning is known to be an NP-complete problem and remains to be an active research field, many studies have investigated effective heuristics to obtain reasonably well partitions [39,52,57,58]. While these methods are helpful in discovering neighborhoods in large graph networks, they do not consider the similarities of the node properties. Karypis and Kumar offered a partitioning approach based on multilevel scheme involving coarsening, partitioning and refinement in constructing a partition for the original graph [30]. Tian et al. [52,58] proposed an aggregation-based graph summarization utilizing graph node attributes. Yet, 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 a similar aspect, some studies have explored partitioning methods to speed up query performances over schema-flexible relational data [16,26]. Herrmann et al. proposed an online algorithm for horizontal partitioning of irregularly structured data [26] that separate entities into fix-sized partitions based on the schema properties of the entities with the goal of increasing the query efficiency by early pruning of partitions containing irrelevant entities. But their approach is based on an assumption that each partition already created is homogeneous and irregularities do not exist in partitions.

In the Semantic Web community, there has also been some related studies [8,12,23,32,53]. The studies in [12] and [23] are query-driven graph summarization methods. They primarily focus on the problem of SPARQL query formulation over RDF data. The approaches using bisimulation [32,53] have some structural limitations due to constraints of bisimulation algorithm.

In the literature, there are some studies that suggested using index structures for graph data to speed up graph queries. Pham et al. proposed a technique to detect a relational schema from RDF data [44]. Different from our approach, they focus on improving the efficiency of structured query language SPARQL or SQL-based applications on RDF dataset using a RDBMS and incorporating RDF schema for RDBMS with goal of bridging between the Semantic Web and RDBMS. Similarly, Picalausa et al. attempted to improve query performances on RDF datasets in native RDF management systems by employing design structural indexing for practical SPARQL fragments [45]. In the area of physical RDF data storage systems, some systems have been investigating auxiliary structures for efficient data storage and query answering purposes over the past decade such as gStore [59], Matrix Bit [4], Parameterizable Index Graph [53]. In particular, gStore proposed an index structure called VS-tree for answering aggregate SPARQL queries quickly, as an improvement on subgraph matching, which is an NP-complete problem. However, their approach requires strict structural matching.
In [22], Gaertler presented graph clustering as a problem of finding natural decompositions of graphs and described some indices for measuring the quality of clustering. Palma et al. offered a semantic-based graph partitioning approach for discovery of Drug-Target interactions in heterogeneous bipartite graphs. Their approach uses semantic similarities in determining edge constraints that allow generating minimal partitioning of weighted bipartite graph with highest density [42]. This problem is analogous to Vertex Coloring Graph using Edge Constraints, which is known to be NP-complete problem. Although their improved implementation achieves polynomial time complexity of $O(|WE|^3)$, the approach is still unsuitable for large graph datasets. Similarly, Traverso-Ribón et al. provided an approach for discovering missing relations in knowledge graphs using bipartite graph partitioning [56]. Unlike [42], they consider the neighborhood similarity of entities along with datatype property values when calculating similarity values used in graph partitioning. Similar to [56], we also consider homophily prediction principle and neighborhood similarity in approach when discovering related entities.

Neighborhood-based similarity measures have been investigated by several studies including SimRank [28], SimRank++ [3], PageSim [34], MatchSim [35], PathSim [51], Co-Citation [49], and GADES [55]. For calculating similarity values between entities, GADES relies on semantic knowledge such hierarchies, neighbors, and node degrees embedded in knowledge graphs.\(^2\) Notably, SimRank is a widely known measure, which utilizes the mean of the edge similarities between nodes [3]. However, this may reduce the similarity score of similar graph nodes in a counterintuitive 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 impacts 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 [13,48]. 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 [13], they determine the property weights using the distinct value-based weight generation and assign higher weight to a property that references more distinct values. That being said, a training set consisting of instances may not always be readily available in real-world datasets.

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. Moreover, we scaled our summary graph generation algorithm by pairing subject nodes based on MinHash. 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 (2008) RDFa in XHTML: syntax and processing. Recommendation W3C
2. Alzogbi A, Lausen G (2013) Similar structures inside rdf-graphs. LDOV 996
3. Antonellis I, Molina HG, Chang CC (2008) Simrank++: query rewriting through link analysis of the click graph. Proc VLDB Endow 1(1):408–421
4. Atrre M, Chaoji V, Zaki MJ, Hendler JA (2010) Matrix bit loaded: a scalable lightweight join query processor for rdf data. In: Proceedings of the 19th international conference on World wide web, ACM, pp 41–50
5. Auer S, Bizer C, Kobilarov G, Lehmann J, Cyganiak R, Ives Z (2007) Dbpedia: a nucleus for a web of open data. Springer, Berlin
6. Aydar M, Ayvaz S (2018) An improved method of locality-sensitive hashing for scalable instance matching. Knowl Inf Syst pp 1–20
7. Aydar M, Ayvaz S, Melton AC (2015) 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
8. Ayvaz S, Aydar M, Melton A (2015) Building summary graphs of rdf data in semantic web. In: 2015 IEEE 39th annual computer software and applications conference (COMPSAC), vol 2, pp 686–691. https://doi.org/10.1109/COMPSAC.2015.107
9. Bizer C, Heath T, Berners-Lee T (2009) Linked data-the story so far. Int J Seman Web Inf Syst 5(3):1–22
10. Brickley D, Guha RV (2014) RDF schema 1.1. W3c Recommendation. http://www.w3.org/TR/2014/REC-rdf-schema-20140225/
11. Broder AZ (1997) On the resemblance and containment of documents. In: Proceedings of the compression and complexity of sequences 1997, IEEE, pp 21–29
12. Campinas S, Perry TE, Ceccarelli D, Delbru R, Tummarello G (2012) Introducing rdf graph summary with application to

\(^2\) https://github.com/RDF-Molecules/sim_service.
assisted sparql formulation. In: 2012 23rd international workshop on database and expert systems applications, IEEE, pp 261–266
13. Castano S, Ferrara A, Montanelli S, Lorusso D (2008) Instance matching for ontology population. In: SEBD, pp 121–132
14. Chakrabarti D, Faloutsos C (2006) Graph mining: laws, generators, and algorithms. ACM Comput Surv (CSUR) 38(1):2
15. Chierichetti F, Kumar R, Lattanzi S, Mitzenmacher M, Panconesi A, Raghavan P (2009) On compressing social networks. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining, ACM, pp 219–228
16. Chu E, Beckmann J, Naughton J (2007) The case for a wide-table approach to manage sparse relational data sets. In: Proceedings of the 2007 ACM SIGMOD international conference on management of data, ACM, pp 821–832
17. Consens MP, Fionda V, Khatchadourian S, Pirro G (2015) S+ epps: construct and explore bisimulation summaries, plus optimize navigational queries, all on existing sparql systems. Proc VLDB Endow 8(12):2028–2031
18. Cyganiak R, Wood D, Lanthaler M (2014) RDF 1.1 concepts and abstract syntax. W3c Recommendation. http://www.w3.org/TR/rdf11-concepts/section-IRIs
19. Pierce D, Booth C, Ogbuiji D, Deaton CC, Blackstone E, Lenat D (2012) Semanticdb: a semantic web infrastructure for clinical research and quality reporting. Curr Bioinform 7(3):267–277
20. Duan S, Kementsietsidis A, Srinivas K, Udrea O (2011) 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, ACM, pp 145–156
21. Fan W, Li J, Wang X, Wu Y (2012) Query preserving graph compression. In: Proceedings of the 2012 ACM SIGMOD international conference on management of data, ACM, pp 157–168
22. Gaertler M (2005) Clustering. In: Brandes U, Erlebach T (eds) Network analysis. Lecture Notes in computer science, chap. 8, Springer, Berlin, pp 178–215
23. Goasdoué F, Manolescu I (2015) Query-oriented summarization of rdf graphs. Proc VLDB Endow 8(12): https://doi.org/10.14778/2824032.2824124
24. Guo Y, Pan Z, Heffin J (2005) Lumb: a benchmark for owl knowledge base systems. Web Semant Sci Serv Agents World Wide Web 3(2):158–182
25. He X, Kao MY, Lu H (2000) A fast general methodology for information-theoretically optimal encodings of graphs. SIAM J Comput 30(3):838–846
26. Herrmann K, Voigt H, Lehner W (2014) Cinderella—adaptive online partitioning of irregularly structured data. In: 2014 IEEE 30th international conference on data engineering workshops (ICDEW), IEEE, pp 284–291
27. Jain AK, Dubes RC (1988) Algorithms for clustering data. Prentice-Hall, Inc., Upper Saddle River
28. Jeh G, Widom J (2002) SimRank: a measure of structural-context similarity. In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, pp 538–543
29. Jin R, Lee VE, Hong H (2011) Axiomatic ranking of network role similarity. In: Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, pp 922–930
30. Karypis G, Kumar V (1998) A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J Sci Comput 20(1):359–392
31. Khare R, Çelik T (2006) Microformats: a pragmatic path to the semantic web. In: Proceedings of the 15th international conference on world wide web, ACM, pp 865–866
32. Khatchadourian S, Consens MP (2010) Explod: summary-based exploration of interlinking and rdf usage in the linked open data cloud. In: Extended semantic web conference, vol 272–287, Springer, Berlin, pp 272–287
33. Levinson N (1946) The wiener (root mean square) error criterion in filter design and prediction. J Math Phys 25(1):261–278
34. Lim Z, Lyu MR, King I (2006) PageSim: a novel link-based measure of web page similarity. In: Proceedings of the 15th international conference on world wide web, ACM, pp 1019–1020
35. Lin Z., Lyu MR, King I (2009) Matchsim: a novel neighborhood-based similarity measure with maximum neighborhood matching. In: Proceedings of the 18th ACM conference on information and knowledge management, ACM, pp 1613–1616
36. Luhn HP (1957) A statistical approach to mechanized encoding and searching of literary information. IBM J Res Dev 1(4):309–317
37. Möller K, Heath T, Handschuh S, Domingue J (2007) Recipes for semantic web dog—food— the ESWC and ISWC metadata projects. In: The semantic web, Springer, Berlin, pp 802–815
38. Newman ME (2003) The structure and function of complex networks. SIAM Rev 45(2):167–256
39. Newman ME, Girvan M (2004) Finding and evaluating community structure in networks. Phys Rev E 69(2):026,113
40. Page L, Brin S, Motwani R, Winograd T (1999) The PageRank citation ranking: bringing order to the web. Stanford InfoLab
41. Paige R, Tarjan RE (1987) Three partition refinement algorithms. SIAM J Comput 16(6):973–989
42. Palma G, Vidal ME, Raschid L (2014) Top-k exploration prediction using semantic similarity and edge partitioning. In: International semantic web conference, Springer, Berlin, pp 131–146
43. Parundekar R, Knoblock CA, Ambite JL (2012) Discovering concept coverings in ontologies of linked data sources. In: International semantic web conference, Springer, Berlin, pp 427–443
44. Pham MD, Passing L, Erling O, Boncz P (2015) Deriving an emergent relational schema from rdf data. In: Proceedings of the 24th international conference on world wide web, international world wide web conferences steering committee, pp 864–874
45. Picalausa F, Luo Y, Fletcher GH, Hidders J, Vansummeren S (2012) A structural approach to indexing triples. In: Extended semantic web conference, Springer, Berlin, pp 406–421
46. Raghavan S, Garcia-Molina H (2003) Representing web graphs. In: Proceedings of the 19th international conference on data engineering, 2003, IEEE, pp 405–416
47. Rajaraman A, Ullman JD (2011) Mining of massive datasets. Cambridge University Press, Cambridge
48. Seddiqui MH, Nath RPD, Aono M (2015) An efficient metric of automatic weight generation for properties in instance matching technique. Int J Web Semant Technol 6(1):1
49. Small H (1973) Co-citation in the scientific literature: a new measure of the relationship between two documents. J Am Soc Inf Sci 24(4):265–269
50. Sparck Jones K (1972) A statistical interpretation of term specificity and its application in retrieval. J Doc 28(1):11–21
51. Sun Y, Han J, Yan X, Yu PS, Wu T (2011) Pathsim: meta path-based top-k similarity search in heterogeneous information networks. VLDB–11
52. Tian Y, Hankins RA, Patel JM (2008) Efficient aggregation for graph summarization. In: Proceedings of the 2008 ACM SIGMOD international conference on management of data, ACM, pp 567–580
53. Tran T, Ladwig G (2010) Structure index for rdf data. In: Workshop on semantic data management, IEEE, pp 261–266
54. Tran T, Wang H, Rudolph S, Cimiano P (2009) Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In: ICDE’09. IEEE 25th international conference on data engineering, 2009, IEEE, pp 101–104
55. Traverso I, Vidal ME, Kämpgen B, Sure-Vetter Y (2016) Gades: a graph-based semantic similarity measure. In: Proceedings of the
12th international conference on semantic systems, ACM, pp 101–104
56. Traverso-Ribón I, Palma G, Flores A, Vidal ME (2016) Considering semantics on the discovery of relations in knowledge graphs. In: European knowledge acquisition workshop, Springer, Berlin, pp 666–680
57. Xu X, Yuruk N, Feng Z, Schweiger TA (2007) Scan: a structural clustering algorithm for networks. In: Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining, ACM, pp 824–833
58. Zhang N, Tian Y, Patel JM (2010) Discovery-driven graph summarization. In: 2010 IEEE 26th international conference on data engineering (ICDE 2010), IEEE, pp 880–891
59. Zou L, Mo J, Chen L, Özsu MT, Zhao D (2011) gstore: answering sparql queries via subgraph matching. Proc VLDB Endow 4(8):482–493

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.