Contents lists available at ScienceDirect

International Journal of Information Management Data Insights

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

Efficient semantic summary graphs for querying large knowledge graphs

Emetis Niazmand[a,b,*], Gezim Sejdiu[c], Damien Graux[d], Maria-Esther Vidal[a,b]

[a] *Leibniz Information Centre for Science and Technology University Library (TIB), Germany*
[b] *Leibniz University of Hannover, Welfengarten 1B, Hannover 30167, Germany*
[c] *Smart Data Analytics, University of Bonn, Bonn, Germany*
[d] *Inria, Université Côte d'Azur, CNRS, I3S Sophia Antipolis, France*

A R T I C L E   I N F O

Keywords:
Knowledge graph
Summarization graph
SPARQL evaluation
Embedding model
Distributed context

A B S T R A C T

Knowledge Graphs (KGs) integrate heterogeneous data, but one challenge is the development of efficient tools for allowing end users to extract useful insights from these sources of knowledge. In such a context, reducing the size of a Resource Description Framework (RDF) graph while preserving all information can speed up query engines by limiting data shuffle, especially in a distributed setting. This paper presents two algorithms for RDF graph summarization: Grouping Based Summarization (GBS) and Query Based Summarization (QBS). The latter is an optimized and lossless approach for the former method. We empirically study the effectiveness of the proposed lossless RDF graph summarization to retrieve complete data, by rewriting an RDF Query Language called SPARQL query with fewer triple patterns using a semantic similarity. We conduct our experimental study in instances of four datasets with different sizes. Compared with the state-of-the-art query engine Sparklify executed over the original RDF graphs as a baseline, QBS query execution time is reduced by up to 80% and the summarized RDF graph is decreased by up to 99%.

1. Introduction

During the past decades, the number of linked datasets – known as knowledge graphs (KGs)– has rapidly increased as evidenced in the current state of the Linked Open Data cloud[1]. These datasets are structured following the W3C's standard Resource Description Framework, RDF (Manola, Miller, McBride et al., 2004), and share knowledge on various domains, from a more general purpose KGs such as DBpedia (Lehmann et al., 2015) or WikiData (Vrandecic & Krötzsch, 2014) to specialized ones, e.g., SemanGit (Kubitza, Böckmann, & Graux, 2019). Real-world applications over these types of sources demand the development of optimized techniques to extract meaningful information. The Semantic Web community has actively contributed to RDF management and has proposed formalisms, e.g., SPARQL (Harris, Seaborne, & Prud'hommeaux, 2013) and SHACL (Spahiu, Maurino, & Palmonari, 2018), to express queries and integrity constraints over RDF graphs. Moreover, within the years, efficiency has also been addressed, and various methods have been proposed; they include methods to store RDF graphs, e.g., centralized (Faye, Curé, & Blin, 2012) or distributed (Kaoudi & Manolescu, 2015), as well as to query RDF graphs (Vidal et al., 2010). Indeed, the task of query processing can become incredibly complex whenever RDF graphs come along with large ontologies, and there may be portions of the ontology with no instances in a knowledge graph. Also, complex queries that include graph pattern expressions (e.g., multi-union queries) represent challenges for query engines in processing time (Pérez, Arenas, & Gutiérrez, 2009). Graph summarization is a technique to solve this issue by providing a compact representation of a graph where redundant data is reduced (Shin, Ghoting, Kim, & Raghavan, 2019). As a result, a summarized graph's size is decreased, and effective techniques can be devised to speed up query processing (Kondylakis, Kotzinos, & Manolescu, 2019). RDF summarization has been used in query answering and optimization. It has been applied to recognizing the most notable nodes, discovering schema from the data, and visualizing the RDF graph to quickly understand the data (Cebiric et al., 2019). We propose graph summarization methods by applying both word embedding and graph embedding models to find the most similar predicates by encoding them as vectors. The word embedding models resort to Natural Language Processing (NLP) techniques to represent words in a numeric vector space (Jurafsky & Martin, 2009). Word embedding models used to convert textual information and social media data such as tweet sentences to numeric weightage in vector format. They are studied in a specific domain to solve real issues, such

* Corresponding author.
E-mail addresses: Emetis.Niazmand@tib.eu (E. Niazmand), sejdiu@cs.uni-bonn.de (G. Sejdiu), damien.graux@inria.fr (D. Graux), Maria.Vidal@tib.eu, vidal@l3s.de (M.-E. Vidal).

[1] As of August 2021, the LOD-cloud gathers around 1512 datasets sharing 413,734,019,304 RDF Triples. https://lod-cloud.net/.

https://doi.org/10.1016/j.jjimei.2022.100082
Received 25 November 2021; Received in revised form 13 April 2022; Accepted 16 May 2022

as Neogi, Garg, Mishra, & Dwivedi (2021) and Mishra, Urolagin, Jothi, Neogi, & Nawaz (2021). There are more use cases which employ word embedding models for vector representation of textual words. Chauhan & Palivela (2021) propose a framework which improves the detection of fake news and real news. This framework makes use of neural networks and a tokenization method. The tokenization method has been proposed for feature extraction or vectorization, which assigns tokens to word embeddings. Word embedding models can be applied to RDF graphs as well, and RDF2Vec is an exemplary approach presented by Ristoski & Paulheim (2016).

We aim to provide an algorithm in which a summarized RDF graph groups RDF triples composed of similar predicates; the similarity metrics computed over the embeddings determine this relatedness. SPARQL queries are rewritten based on the summarized RDF graph. As a result, query execution time reduced, while answer completeness is maximized. Our goal is to achieve the following research objectives:

- Role of summarization in the RDF graph size reduction.
- Impact of summarization in query processing.

Two approaches are presented: Grouping Based Summarization (GBS) and Query Based Summarization (QBS). GBS decreases an RDF graph size and QBS considers criteria of graph summarization to rewrite SPARQL queries into queries with fewer triple patterns, but with equivalent answers. Our query rewriting techniques resort to semantic similarity metrics to identify related predicates in the triple patterns of a SPARQL query and replace them with a predicate that represents all of them. QBS has the following desirable characteristics: **a)** *Compactness*: graph summarization provides fewer nodes and edges compared with the original RDF graph by considering only a part of the RDF graph which is related to the given SPARQL query; **b)** *Lossless query processing*: returns the same answers by querying over summarized graph compared with the original one based on similarity metric by transforming a query to the simple one; and **c)** *Low-cost query processing*: speeds up query processing over the summarized RDF graph. The GBS and QBS performance is evaluated; the Sparklify component (Stadler, Sejdiu, Graux, & Lehmann, 2019) is used as a default query engine from the SANSA Stack (Lehmann et al., 2017). The Waterloo SPARQL Diversity Test Suite (WatDiv) benchmark generator (Aluç, Hartig, Özsu, & Daudjee, 2014) is utilized to generate two RDF graphs (WatDiv.10M and WatDiv.100M) and queries; also, the Entity Summarization Benchmark, ESBM (Liu, Cheng, Gunaratna, & Qu, 2020), and a dump of DBpedia[2] are included in the study. We report on twenty queries where the execution time is accelerated by up to 80%. The observed results are promising and provide evidence of our proposed approaches' compactness power and their impact on query processing.

In particular, the contributions of this work are as follows:

- Graph summaries are able to reduce RDF triples required in query processing.
- Query rewriting techniques guided by RDF graph summarization. These techniques ensure answer completeness.
- An empirical study over state-of-the-art benchmarks. Observed results indicate the positive effects of reducing redundant information in the portion of an RDF graph required to execute a SPARQL query.

The rest of the paper is organized as follows: In Section 2, we review the related efforts in the domain of RDF summarization. Section 3 presents an in-depth example to illustrate our challenges. Section 4 presents our proposed approaches. The methodology and the results of our empirical evaluation are reported in Section 5. We have mentioned our discussions in Section 6. Finally, in Section 7, we conclude and draw the next challenges to be addressed.

[2] https://wiki.dbpedia.org/.

2. Related work

Graph summarization techniques reduce the size of graph, speed up graph query evaluation, as well as facilitate graph visualization and analytics. In addition, it provides semantic searches with a reduction in computational complexity. We analyze existing approaches for RDF graph summarization and query processing over summarized RDF graphs.

2.1. Graph summarization in RDF

Graph databases use graph structures for representing entities as nodes and their relationships as edges of a graph Bourbakis (1998). The increment of data in graph databases makes the query processing complicated. Summarization technique is a way to overcome the complexity of search query in graph databases (LeFevre & Terzi, 2010). Graph summarization has been studied for semi-structured graph data models such as XML (Qun, Lim, & Ong, 2003) and RDF graphs. There are many existing works in this area to compact RDF graphs whose structure is computed from the original RDF graph, such that all the paths present in the original graph are also present in the summary graph (Bonifati, Dumbrava, & Kondylakis, 2020). These techniques can be classified into four categories, as the following (Cebiric et al., 2019):

2.1.1. Structural methods

This summarization method considers the structural RDF graphs. One summarization technique following this method is the adaptive structural summary for RDF graph (ASSG) presented by Zhang, Duan, Yuan, & Zhang (2014). It compresses a part of RDF graph which is considered by a collection of queries. This technique requires some user-selected queries for building the summary graph. By compressing only the part which consists of the users' queries, the number of edges and nodes is decreased. This technique considers only the structure of the RDF graph and not from a semantic point of view. The Query Based Summarization approach presented in this paper is based on a similar method, but by considering both structure and semantic. Practically, the nodes with the same labels and ranks are assigned in the same equivalence class. Each equivalence class in graph data has a set of nodes, the rank of the nodes, and the labels of the nodes. Therefore, the graph data is divided into some equivalence classes. As a consequence, the compressed graph has fewer nodes and edges compare with the original one. The approach by Gurajada, Seufert, Miliaraki, & Theobald (2014b) uses structural methods to summarize the RDF graphs and studies efficient query processing in the TriAD (Gurajada, Seufert, Miliaraki, & Theobald, 2014a) a distributed RDF data management engine, by relying on a summary of the RDF graph stored within the system. The other approach presented by Sydow, Pikula, & Schenkel (2013) shows the problem of selecting the most important part of an RDF graph based on a chosen entity by user. This approach lets the system generate a summarization of facts concerning the selective entity where is close to the Query Based Summarization approach presented in this paper.

2.1.2. Pattern mining methods

This method discovers patterns to build summary graphs. For example, Zneika, Lucchese, Vodislav, & Kotzinos (2016) presents an approach for summarizing RDF graphs using mining a set of approximate graph patterns and calculating the number of instances covered by each pattern. Then it transforms the patterns to an RDF schema that describes the contents of the knowledge graph. In that case, the evaluation of queries are done over the summarized graph instead of the original graph. Moreover, the computational methods presented by Karim, Vidal, & Auer (2020) identify frequent star patterns to generate compact representation of RDF graphs, with a minimized number of frequent star patterns.

2.1.3. Statistical methods

This summarization method follows a frequency-based perspective to summarize graphs. The work by Zhu, Ghasemi-Gol, Szekely, Galstyan, & Knoblock (2016) presents a technique called CoSum where a multi-type graph is as an input and the output is a super-graph. CoSum is assigned to statistical RDF summarization type; it generates summary graphs frequency-based and in quantitatively way. According to the authors, CoSum technique should be used for summarizing the graph by clustering the nodes which share the same type. The idea of grouping subjects with the same predicate and objects in Grouping Based Summarization approach comes from this technique. Then, each cluster refers to the Super-Node which consists of nodes with the same type. These Super-Nodes are linked to each other by weighted edges. Therefore, the main purpose of this approach is to automatically group elements that correspond to the same entity, which is called *Entity Resolution* (Benjelloun et al., 2009). In general, this technique tries to transform a k-type graph to another k-type summary graph, which consists of Super-Nodes and Super-Edges linked among each other. CoSum as a summarizing technique provides a solution to deal with the following challenges: **i)** An RDF graph is modeled as a multi-type graph and the collective entity resolution is formulated as a multi-type graph summarization problem. **ii)** A multi-type graph co-summarization-based method is proposed in order to identify entities and link connections between them at the same time. **iii)** A generic framework is provided to accept different domains-specific knowledge. In the summary graph, each Super-Node is a group of some vertices with the same type and each Super-Edge connects these clusters of nodes to each other.

2.1.4. Hybrid methods

This method combines two or all other categories to generate summary graphs. As an example, the summarization method presented by Zheng et al. (2016) is the hybrid RDF summarization method, because it considers both structure and patterns to construct a summary graph.

2.2. Semantic search and query processing

Motivated by the aim of simplifying queries consisting of the union of some queries, Zheng et al. (2016) developed a solution based on similarity search. This means instead of using multiple union of queries to get the complete answers, only by using a single or less union of queries can get the correct and the same results which are given by multi-union queries. Our work shares the same observation: reducing SPARQL complexity can be achieved thanks to summarization techniques. This technique includes both structural and semantic similarities. Since queries can be different in terms of structure, but they have similar semantic meanings, some operations such as semantic path substitution are introduced. Semantic path substitution operation is used to replace a path with an edge by mining the structure patterns. A dictionary of semantic instances is provided to mine semantic graph patterns by keeping instances which are semantically equivalent. Finally, rewriting a given query graph by semantic path substitution gives a set of semantically equivalent queries. Also, based on these operations, a similarity measure, called *Semantic Graph Edit Distance (sged)*, is defined by Zheng et al. (2016). *Sged* measures the cost of transforming one Sub-Graph to another one. Then Sub-Graphs extracted from the RDF graph will be chosen to provide the summary graph if they have minimum *sged-based* transformation cost.

3. Challenges and motivation

Efficient query processing over large RDF graphs is one of the main challenges in data management. We motivate this data management problem with two examples and illustrate – with a real-world use case – the impact of an RDF graph size on execution time. Fig. 1a depicts a portion of an RDF graph with entities related by properties. It comprises properties that are semantically similar (e.g., $country$, $nationality$, and $birthPlace$). Let us consider a graph summarization method by Zhu et al. (2016) that groups similar entities and properties in a graph. All the elements of a group (i.e., entities or properties) are summarized into one element (i.e., into an entity or a property) in the summarized graph. Some works have been done for grouping the elements with similar semantic meaning, e.g., Singh, Devi, Devi, & Mahanta (2022) propose a method to group the terms with similar semantic meaning by evaluating the similarity between words using GloVe (Pennington, Socher, & Manning, 2014).

The results of applying this method to the RDF graph in Fig. 1a are illustrated in Fig. 1b. Moreover, Fig. 1c presents the RDF serialization of the RDF triples in Fig. 1a and b.

A portion of DBpedia that consists of 2047 RDF triples is presented in Fig. 2a. Nodes and edges are represented as ovals and rectangles, respectively. The dataset has 1000 edges and 510 nodes. This figure also presents a SPARQL query comprising four triple patterns. The evaluation of this query retrieves five answers that correspond to the names of people in Germany, or with German nationality, or born or die in Germany. The Sparklify query engine[3] produces these answers in 19 seconds. Fig. 2b and c depict the execution time over summarized RDF graphs computed following the two graph summarization methods previously described. The results in Fig. 2b suggest that the execution of the SPARQL query over the summarized graph by naive approach, even producing all the results, can be costly. This approach grouped source nodes with similar edges and target nodes in 174 Sub-Graphs. Thus, query processing over the summarized RDF graph requires 43 secs. to produce the five answers. Alternatively, graph summarization can be done during query processing. An optimized query-based graph summarization method can identify the portion of the RDF graph required to answer the query, and then it summarizes only this portion of the original RDF graph. The results in Fig. 2c show the query execution over a summarized RDF graph by optimized approach with 16 edges and 17 nodes; it retrieves all the five answers in 5 seconds. The RDF graphs in Fig. 2 are generated by Cytoscape[4].

Fig. 3 a presents a compact representation of the portion of DBpedia in Fig. 2a required to answer the SPARQL query in Fig. 3b. This graph represents–with one property– the three related properties *country, nationality*, and *birthPlace*. The property *deathPlace* remains in the compact query since it is not semantically similar to other properties. This compact modeling reduces the RDF graph size and enables rewriting the query into a query with fewer triples in Fig. 3b. As a result, the rewritten query execution time is reduced to 5 seconds, while the complete five answers are produced. These examples illustrate the relevance of efficient summarization techniques in RDF data management. In Section 4, we address this problem, and describe summarization techniques able to reduce the size of RDF graphs and speed up execution time during query processing over summarized graphs.

4. Research problem and proposed approach

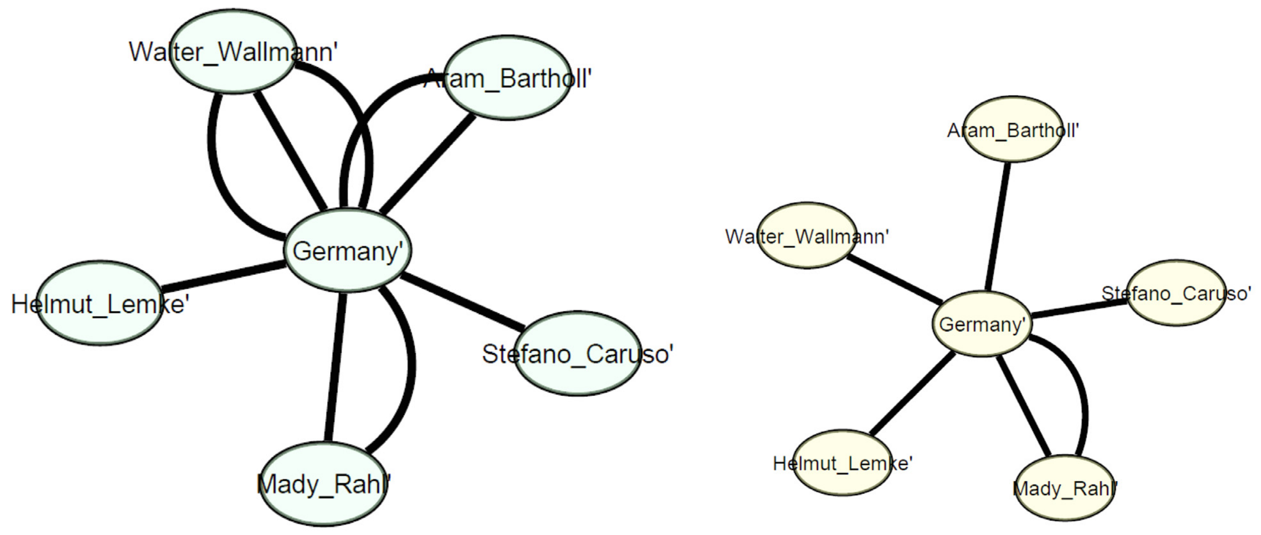
In this section, we discuss the problem of query processing over summarized RDF graphs. Also, we introduce important preliminary definitions and provide the solution by proposing two approaches, naive and optimized, for summarizing RDF graphs without losing necessary information.

4.1. Problem statement

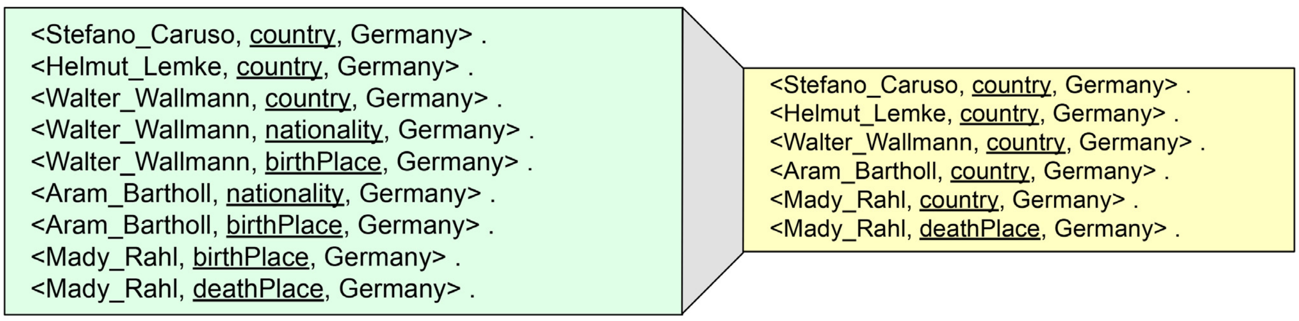
Summarization techniques minimize the size of RDF graphs, which helps optimize query processing. Meanwhile, preserving all needed information should be considered during summarization process. Summarizing graph based on a semantic similarity measure is a technique to solve query processing over large RDF graphs. This section presents techniques that exploit knowledge encoded in RDF graphs, similarity measures, and SPARQL queries; they generate summarized graphs against which queries can be processed. To illustrate the relevance of determining relatedness using similarity measures, consider the RDF graph in

[3] http://sansa-stack.net/sparklify/.

[4] http://www.cytoscape.org/.



(a) Original RDF Graph (b) Summarized RDF Graph


<Stefano_Caruso, country, Germany> .
<Helmut_Lemke, country, Germany> .
<Walter_Wallmann, country, Germany> .
<Walter_Wallmann, nationality, Germany> .
<Walter_Wallmann, birthPlace, Germany> .
<Aram_Bartholl, nationality, Germany> .
<Aram_Bartholl, birthPlace, Germany> .
<Mady_Rahl, birthPlace, Germany> .
<Mady_Rahl, deathPlace, Germany> .

<Stefano_Caruso, country, Germany> .
<Helmut_Lemke, country, Germany> .
<Walter_Wallmann, country, Germany> .
<Aram_Bartholl, country, Germany> .
<Mady_Rahl, country, Germany> .
<Mady_Rahl, deathPlace, Germany> .

(c) Triples summarization

Fig. 1. Motivating Example (Compactness). (a) An RDF graph representing entities with similar properties; (b) A lossless summarization of the RDF graph preserving main information; (c) Triples related to the RDF graph in the green box are summarized to a smaller portion in the yellow box. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Fig. 1b; it summarizes the edges of an RDF graph in Fig. 1a that are related or similar. A similarity measure is a function that given two entities associates a value in the range [0.1] that indicates a degree of relatedness of the input entities. Similarity measures rely on various properties of the input entities to estimate similarity.

4.2. Preliminaries

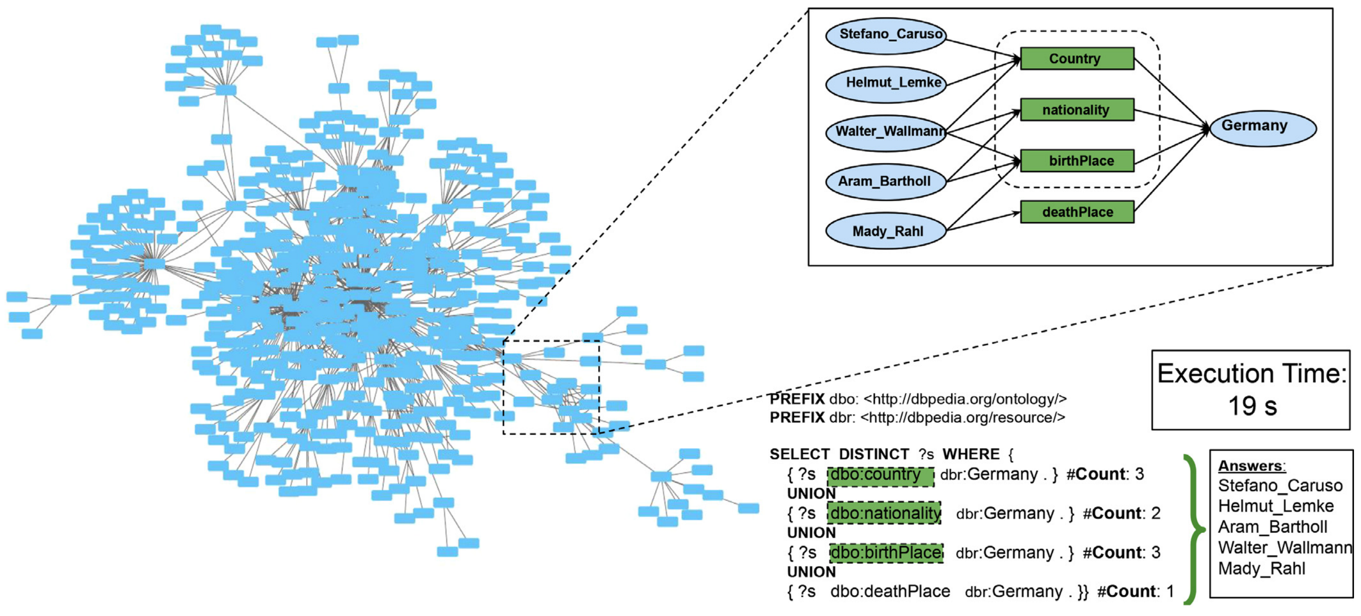
Given two classes C_1={$e_{11}, e_{12}, \ldots, e_{1n}$} and C_2={$e_{21}, e_{22}, \ldots, e_{2m}$} where e_{1i} and e_{2j} are entities in classes C_1 and C_2, respectively. The semantic similarity between these two classes is defined as $sim(C_1, C_2)$ = $Avg(sem\_sim(e_{1i}, e_{2j}))$ for the set of all pairs of (e_{1i}, e_{2j}) in C_1 X C_2 and e_{1i} !=e_{2j}. Therefore, two classes are similar to each other if and only if a set of entities in class C_1 are similar to a set of entities in class C_2 (Jatnika, Bijaksana, & Ardiyanti, 2019). Also, these entities should not be the same. The value of similarity is equal to the average of the semantic similarity value of all entity pairs.

One of the metrics to measure the similarity is cosine similarity; it calculates the similarity between two n-dimensional vectors by looking for a cosine value from the angle between two vectors. The entities in classes are converted to the vectors by a model to calculate the angle between them. The value of cosine similarity is between 0 and 1. If the value is closer to 1, it means entities are more similar to each other. And if the value is closer to 0, it means the similarity between entities is less. In the following, the formula shows the semantic similarity between sets of entities (e_{1i} and e_{2j}):

$$sem\_sim(e_{1i}, e_{2j}) = \cos(\theta) = \frac{e_{1i}.e_{2j}}{|e_{1i}||e_{2j}|} \tag{1}$$

There are many embedding models can be used to measure semantic similarity and relatedness by the cosine between the concepts' embedding vectors. Some of these methods focus on the terms called word embedding, and some focus on the relations known as graph embedding. Word2Vec presented by Mikolov, Chen, Corrado, & Dean (2013) as a word embedding model is used in this work to generate concept sentence embeddings based on the terms. Word2Vec model transforms words into low-dimensional word embeddings; it resorts to small neural networks to calculate these word embeddings based on contextual knowledge encoded in public background knowledge bases. In addition, to compute word embeddings, the Word2Vec model calculates the cosine of the angle between these low-dimensional vectors that represent these embeddings. Word2Vec model resorts to the cosine similarity as a measure to find similar words. In order to find similar words, a trained Word2Vec model based on gensim library[5] can be used. The main precondition of word embedding is that words with similar meaning should have a similar representation. There are many entities and relations that are semantically similar, but they are represented differently in the knowl-

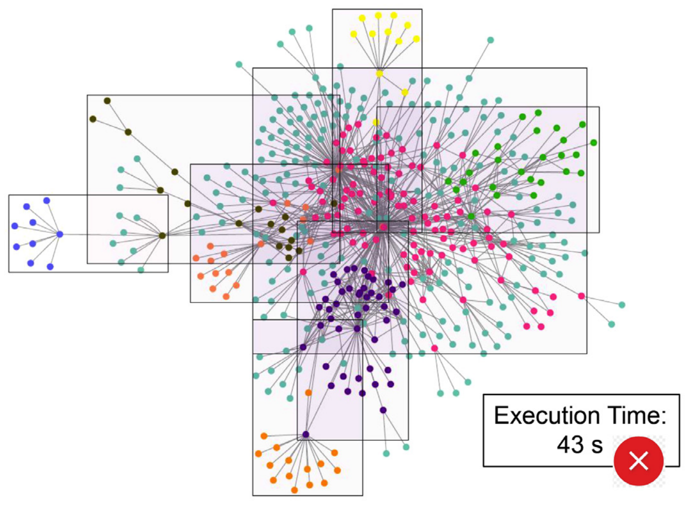
[5] https://radimrehurek.com/gensim/models/word2vec.html.



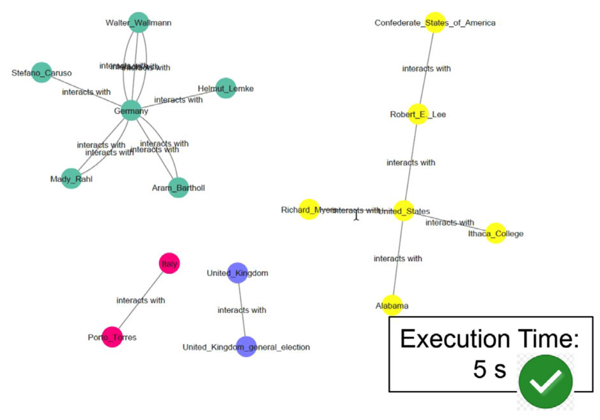
```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>

SELECT DISTINCT ?s WHERE {
  { ?s  dbo:country  dbr:Germany . } #Count: 3
  UNION
  { ?s  dbo:nationality  dbr:Germany . } #Count: 2
  UNION
  { ?s  dbo:birthPlace  dbr:Germany . } #Count: 3
  UNION
  { ?s  dbo:deathPlace  dbr:Germany . }} #Count: 1
```

(a) Original RDF Graph and SPARQL query



(b) A Naive Approach for Summarizing RDF Graphs



(c) An Optimized Approach for Summarizing RDF Graphs

Fig. 2. Motivating Example (Execution Time). (a) A SPARQL query retrieves five answers in 19 seconds from the original RDF graph; (b) Summarized RDF graph by naive approach retrieves the same answers in 43 seconds; (c) Optimized approach retrieves the same answers in 5 seconds.

edge graph. Thus, considering context by additional training data is an important task to generate contextualized word embedding.

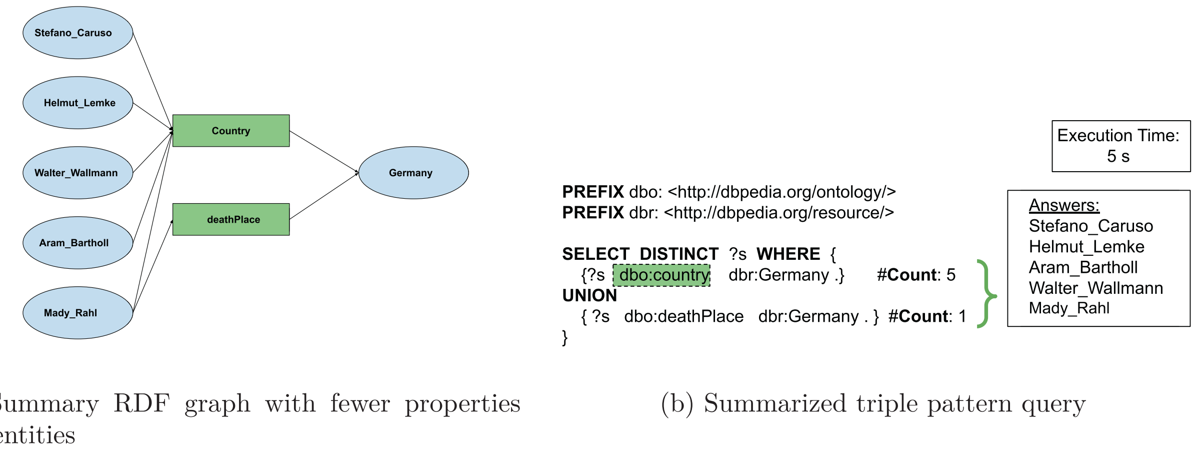
In the running example, Word2Vec is utilized to determine that the properties *country, nationality*, and *birthPlace* are related and similar. Similarity values can guide the summary of properties in RDF graphs and allow for the transformation of SPARQL queries. Fig. 3 illustrates a transformed SPARQL query. Albeit simpler, the transformed SPARQL query can retrieve the same results as the original query, but in less time. Meanwhile, there are a number of applications where data are represented in the form of graphs, which required graph embedding.

RDF2Vec presented by Ristoski, Rosati, Noia, Leone, & Paulheim (2019) as a graph embedding model is applied to learn the context of the relations. In the case of RDF knowledge graphs, entities and relations between entities are considered instead of word sequences. First, the graph data is converted into sequences of entities; it can be considered as sentences using two different approaches, i.e., graph walks and Weisfeiler-Lehman (WL) subtree RDF graph kernels. Using those sentences, RDF2Vec trains the same neural language models to represent each entity in the RDF graph as a vector of numerical values in a latent feature space.

Built on existing results on graph embeddings and summarization, we propose two approaches for summarizing RDF graphs. The first called Grouping Based Summarization (GBS) approach; it summarizes the RDF graph based on grouping subjects with the same predicates and objects. The second one, optimized for the first one, called Query Based Summarization (QBS) considers only the part of the RDF graph which is related to the SPARQL query. In the next, GBS and QBS are defined in detail.

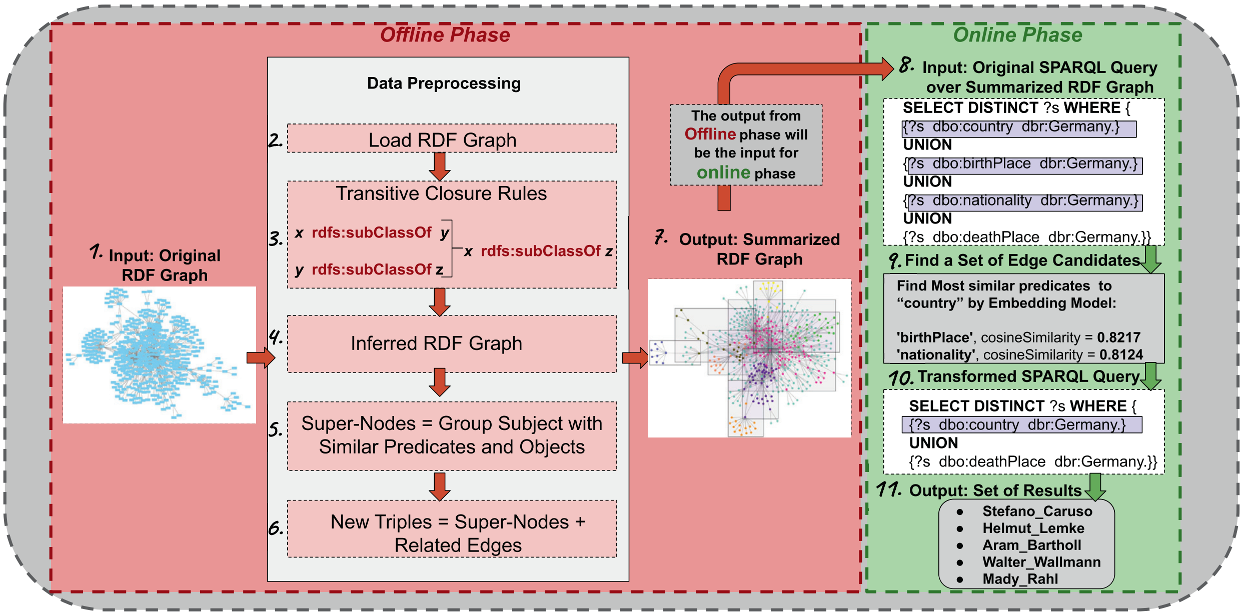
4.3. A naive approach for summarizing RDF graphs

A Grouping Based Summarization (GBS) approach able to reduce size of RDF graph represents our naive method. GBS works into phases: *Offline Phase* and *Online Phase*. In Fig. 4 both phases are shown. The



Stefano_Caruso
Helmut_Lemke
Walter_Wallmann
Aram_Bartholl
Mady_Rahl
Country
deathPlace
Germany
Execution Time: 5 s
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>
SELECT DISTINCT ?s WHERE {
{?s dbo:country dbr:Germany .} #Count: 5
UNION
{ ?s dbo:deathPlace dbr:Germany . } #Count: 1
}
Answers:
Stefano_Caruso
Helmut_Lemke
Aram_Bartholl
Walter_Wallmann
Mady_Rahl
(a) Summary RDF graph with fewer properties and entities
(b) Summarized triple pattern query

Fig. 3. Example of Summarized RDF. (a) An RDF graph by considering only one of the most similar predicates returns five answers in 5 seconds. Simplify returning the same answers without being aware of schema and all predicates in less time by reducing the number of triples in RDF dataset; (b) Retrieve the same answers in less time compared with original RDF graph by applying the transformed SPARQL query as a simple one over the summarized RDF graph.



Offline Phase
Online Phase
Data Preprocessing
1. Input: Original RDF Graph
2. Load RDF Graph
3. Transitive Closure Rules
x rdfs:subClassOf y
y rdfs:subClassOf z
x rdfs:subClassOf z
4. Inferred RDF Graph
5. Super-Nodes = Group Subject with Similar Predicates and Objects
6. New Triples = Super-Nodes + Related Edges
The output from Offline phase will be the input for online phase
7. Output: Summarized RDF Graph
8. Input: Original SPARQL Query over Summarized RDF Graph
SELECT DISTINCT ?s WHERE {
{?s dbo:country dbr:Germany.}
UNION
{?s dbo:birthPlace dbr:Germany.}
UNION
{?s dbo:nationality dbr:Germany.}
UNION
{?s dbo:deathPlace dbr:Germany.}}
9. Find a Set of Edge Candidates
Find Most similar predicates to "country" by Embedding Model:
'birthPlace', cosineSimilarity = 0.8217
'nationality', cosineSimilarity = 0.8124
10. Transformed SPARQL Query
SELECT DISTINCT ?s WHERE {
{?s dbo:country dbr:Germany.}
UNION
{?s dbo:deathPlace dbr:Germany.}}
11. Output: Set of Results
Stefano_Caruso
Helmut_Lemke
Aram_Bartholl
Walter_Wallmann
Mady_Rahl

Fig. 4. Proposed Summary Graph Architecture for the Grouping Based Summarization (GBS) approach in two phases (The Naïve Approach).

input of the offline phase is an RDF dataset that has been loaded as an RDF graph, and the output is a semantic summary graph. For the online phase, the input is a generated summary graph from the offline phase, a SPARQL query, and a semantic similarity metric and the output is a transformation of the query to the one with fewer triple patterns with final answers.

In the offline phase presented in Algorithm 1 , data should be preprocessed. All edges and vertices are read from the RDF graph. As discussed before, the aim is to provide a semantic summary graph without losing the needed information. After receiving a dataset as an input in *(step 1)*, the RDF graph is loaded *(step 2)*. Then, the RDF graph is expanded to find new relations which are already not available in the original RDF graph, but they have similar semantic meanings. This extension is performed by computing the transitive closure of the properties in the RDF graph.

Inference layer is used in order to extract new knowledge. For inferring the new facts from the current knowledge bases, inference rules are applied. Transitive Closure (TC) is one of the inference rules which is exerted in this work to infer more facts. It is deployed in the graph in order to find more facts which not exist in the original one.

For example, in the given original RDF graph, there is no relation between entities *Germany* and *Country*. As seen in Fig. 5, by applying TC inference rule, the new triple ⟨*Germany, type, Country*⟩ has been inferred out of existing triples ⟨*Germany, type, EuropeanCountry*⟩ and ⟨*EuropeanCountry, subClassOf, Country*⟩. Since the original RDF graph has expanded by applying inference rules with more triples, queries over *Country* and *Germany* can be equally answered.

The Transitive Closure (TC) inference rule is deployed in the graph in order to find more facts *(step 3)* and for expanding the RDF graph. In *(step 4)*, the inferred RDF graph is generated with the new properties. Subjects with similar predicates and objects need to be identified to find Super-Nodes (SN). In order to store the results from massive data, in-memory Spark Resilient Distributed Dataset (RDD) is used *(step 5)*.

The Resilient Distributed Dataset (RDD) is the core of Apache Spark.[6] As it comes from the name, RDD is a resilient, distributed, and immutable collection of data that are partitioned over a cluster of machines. In Spark RDD, a cluster of workers is connected to a driver

[6] https://spark.apache.org.

Algorithm 1: The Grouping Based Summarization (GBS) Algorithm, *Offline phase-Summarizing Graph*.

```
   Input  : RDF dataset has been loaded as RDF graph (G)
   Output: Semantic Summary Graph ($G'$)
1  G ← RDFGraphLoader.loadFromDisk(spark, input, parallelism) ;
2  inferredGraph ← TransitiveReasoner.apply(G) ;
3  SN ← RDD[(List[Subjects])] from inferredGraph where Subjects have similar Predicates and Objects ;
4  /*SN is a list of Super-Nodes*/newTriples ← Triple.create(SN, Predicate, Object) ;
5  Buffer ← new ArrayBuffer(triple.length) ;
6  foreach (SN, Predicate, Object) ∈ inferredGraph do
7      if newTriples not exists in Buffer then
8          Buffer += newTriples ;
       end
   end
9  $G'$ ← Buffer ;
10 return $G'$
```

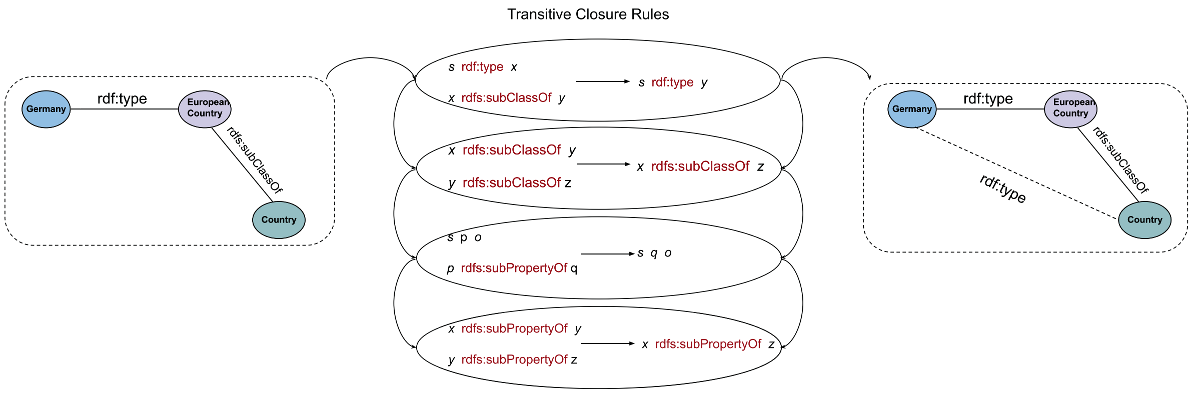


Fig. 5. Example of Transitive Closure (TC) rule in Inference Layer.

or master node. A master node will take care of work execution while worker nodes execute the jobs which are split and then distributed to them. In Sparklify, RDF graphs are stored and modeled based on Spark RDD through fast processing for efficient evaluation of SPARQL queries over distributed RDF datasets.

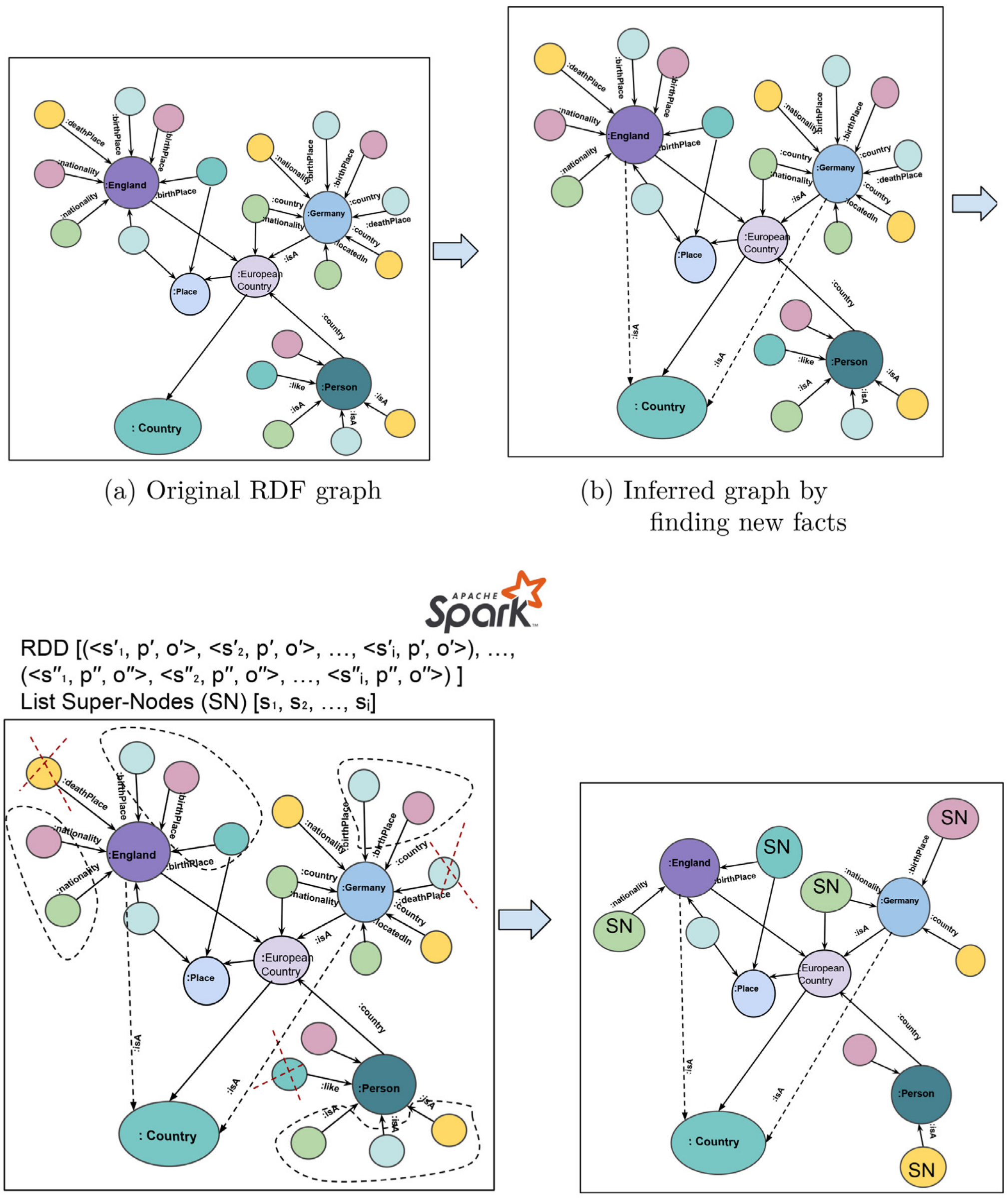
All subjects which have common predicates and objects are grouped. RDDs of (key,value) pairs are used like (*Predicate, Object, List*(*Subjects*)); this resembles the technique proposed by Consens, Fionda, Khatchadourian, & Pirrò (2015); it also finds the same nodes by clustering entities with the same type. Pair RDDs display operations such as $reduceByKey()$ and $groupByKey()$ for combining and grouping values with the same key. Each of these RDDs of pairs can be considered as a Sub-Graph. Also, subjects in the group list are considered as Super-Nodes (SN). Fig. 6 shows with a simple example how a summary RDF graph has been generated by GBS algorithm in offline phase. For example, all people who were born in Germany can be grouped and consider as a Super-Node (SN). New triples are created out of these Super-Nodes with their related edges and added to the buffer*(step 6)*. So, a summarized RDF graph is generated from the original one. In general, the total number of edges and vertices of this summary graph is less than the original RDF graph. Therefore, the summarized graph is created *(step 7)*.

After generating a summary RDF graph, the aim is to have a complete set of answers corresponding to a reduced number of triple patterns in the query. In the online phase, a multi triple pattern query is processed in *(step 8)* to find a set of edge candidates based on the embedding model such as Word2Vec *(step 9)*. As explained in Section 4, cosine similarity as a measure is used to find similar predicates by their distance from each other. Also, a trained model based on gensim library has been used. In our model, semantically similar predicates tend to lie close to each other. For example, the cosine similarity value between a given predicate $country$ and predicates $nationality$, $birthPlace$, and $deathPlace$ is 0.8217, 0.8124, and 0.3672, respectively. In *(step 9)*, the edges which have a higher similarity value than a given threshold (> 0.5) are selected. Therefore, predicate $deathPlace$ cannot be considered as a similar predicate to $country$. The similar edges are considered as strong relations between vertices and are called Super-Edges. As seen in Algorithm 2, the Super-Edges discovered are used in transforming the query to the simple one to find complete results in *(step 10)*. Word embedding techniques consider similarity between edges based on their distance. Thus, in summarized RDF graphs where the size of graph is smaller and predicates are closer to each other, there is a possibility that a founded predicate is similar to the others, not only in terms of distance but also from the semantic point of view. A simple example in Fig. 7 demonstrates how the algorithm of GBS approach works in online phase.

In *(step 11)*, final answers are generated by getting help from a query engine, e.g., Sparklify, over the summary RDF graph. After retrieving answers, it is observed that some required information is lost. In Section 5, the results of evaluation show that querying over summarized RDF graph in GBS approach for large RDF graphs returns less number of answers compared with querying over original RDF graph. In order to avoid the problem of losing information during query processing, we propose the Query Based Summarization (QBS) approach.

4.4. An optimized approach for summarizing RDF graphs

The optimized approach in order to reduce size of RDF graphs is Query Based Summarization (QBS) approach. QBS guides the summa-


:deathPlace
:birthPlace
:nationality
:England
:Germany
:country
:locatedIn
:isA
:European Country
:Place
:Person
:like
: Country
RDD [(<s′1, p′, o′>, <s′2, p′, o′>, …, <s′i, p′, o′>), …,
(<s″1, p″, o″>, <s″2, p″, o″>, …, <s″i, p″, o″>)]
List Super-Nodes (SN) [s1, s2, …, si]
SN

(a) Original RDF graph

(b) Inferred graph by finding new facts

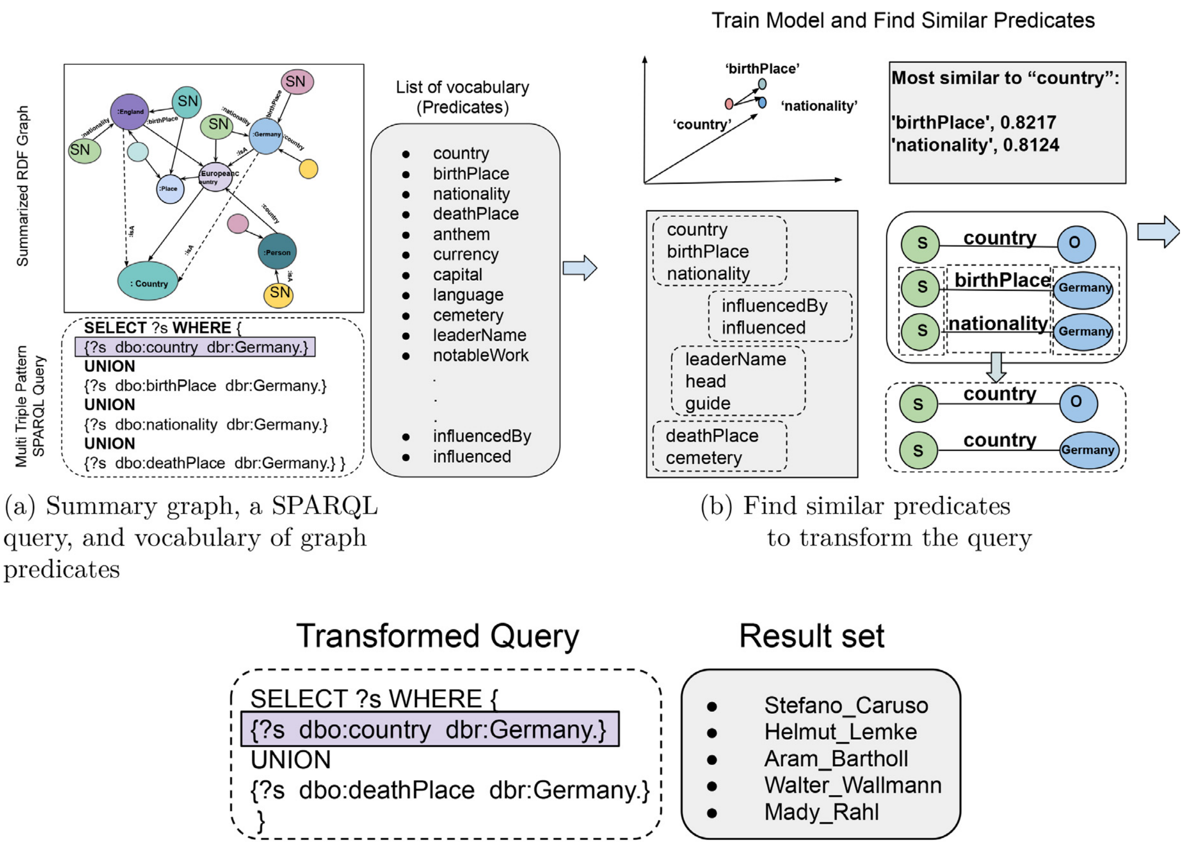
(c) Group subjects with the same predicate and object using RDD Spark

(d) Summarized RDF graph

Fig. 6. Example of Grouping Based Summarization (GBS) approach (offline phase). (a) A part of original RDF graph; (b) Graph expanded using the Transitive Closure rule to find new facts which do not exist in the original RDF graph, but they are semantically true; (c) Subjects with the same predicate and object are grouped to find Super-Nodes (SN) to create new triples using RDD Spark, also the nodes that do not share the same predicate and object will be removed; (d) Generate a summary RDF graph with fewer nodes and edges.

rization process based on an input query. Thus, instead of considering a whole RDF graph to summarize, only the part of the original RDF graph which is related to the user query will be considered. This idea is inspired by Zhang et al. (2014). In this way, not only the execution time is reduced, but also all the necessary information is preserved. The input of this algorithm is an RDF dataset that has been loaded as an RDF graph, a SPARQL query, and a semantic similarity metric. The output is a transformation of the query to the simple one with fewer triple patterns and a semantic summary graph with final answers; it is presented in Algorithm 3. After receiving an RDF graph as input and load it *(step 1–2)*, the query of the user should be collected to extract predicates and objects related to the query as Super-Predicates and Super-Objects *(step 3)*. In this approach, contrary to GBS, the inferred graph is not generated due to the time-consuming and considering small part of graph related to the query. Therefore, the RDF graph is not expanded like in GBS approach.

After extracting the Super-Predicates and Super-Objects in *(step 3)*, a Sub-Graph (g) consisting of triples with predicates equal to the Super-Predicates or with objects equal to the Super-Objects is generated *(step 4)*. The architecture of Query Based Summarization (QBS) is illustrated in Fig. 8. In the next step, word and graph embedding models are applied to this Sub-Graph to find edge candidate sets. Unlike the GBS approach, the embedding models consider the Sub-Graph instead of the whole graph. By this method, only relevant predicates will be found as similar *(step 5)*. Indeed, embedding models help that query is transformed into a simple SPARQL query. It also helps that Super-Subjects are found by extracting triples with the predicates equal to the edge candidate sets *(step 6)*. By having Super-Subjects, Super-Predicates, and Super-Objects



(a) Summary graph, a SPARQL query, and vocabulary of graph predicates

(b) Find similar predicates to transform the query

(c) A transformed query and answers

Fig. 7. Example of Grouping Based Summarization (GBS) approach (online phase). (a) The summary RDF graph generated from offline phase with a multi triple pattern SPARQL query and a list of vocabulary before training the model; (b) Apply both word embedding model and graph embedding model to find most similar predicates to transform multi triple pattern queries to simple ones by considering the most similar predicate as a Super-Edge; (c) Transformed query with results.

Algorithm 2: The Grouping Based Summarization (GBS) Algorithm, *Online Phase-Query Rewriting.*

```
Input  : SPARQL Query (Q); Summary Graph ($G'$) from Algorithm~1; List of Vocabulary (V); Semantic Similarity Metric
Output: Transformed SPARQL Query ($Q'$); List of result
1  initialize training model ;
2  V ← list_of_vocabulary ;
3  foreach vocabulary ∈ V do
4      model ← trained model ;
   end
5  Q.Predicates ← set of predicates extracted from Q ;
6  forall the p ∈ Q.Predicates do
7      similar_set_of_P ← predicates q in $G'$ with cosine-similarity(q,P) >0.5 ;
8      Q.replaceBy(P, representativeOf(synonym_set_of_P)) ;
   end
9  $Q'$ ← Q ;
10 result ← $G'$.sparql($Q'$) ;
11 return result
```

from the previous steps, new triples are created in *(step 7)* and added to our Sub-Graph (g) to generate the final graph as a summary graph *(step 8)*. The Super-Edges discovered earlier are used in transforming the query to the simple one *(step 9)*.

Since QBS relies on the query of users, the generated summary graph contains all information related to the query. Therefore, querying the transformed query over the summary graph returns all possible answers. This is proved by Theorem 4.1. By applying the simple query over this summary graph, the answers are retrieved *(step 10)*.

Theorem 4.1. *If C_1 and C_2 are classes in the RDF graph G, and C_1 is the domain of p_1 and C_2 is the domain of p_2. Also, p_1 is similar to p_2 according to a given* ***semantic similarity metric****. Let G'' be the compact representation of G by QBS approach, where p is the property used to represent p_1 and p_2 in G''. The following properties hold:*

1. *The cardinality of G'' and cardinality of G are the same.*

$$cardinality(G'') = cardinality(G) \tag{2}$$

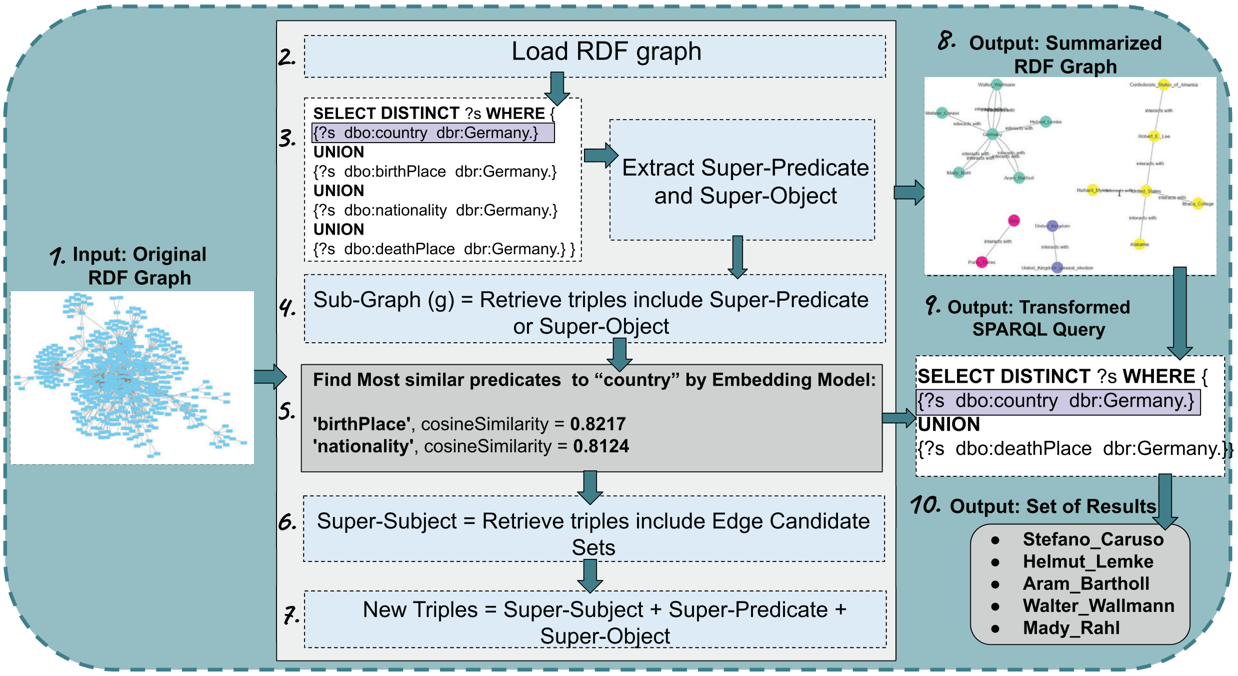
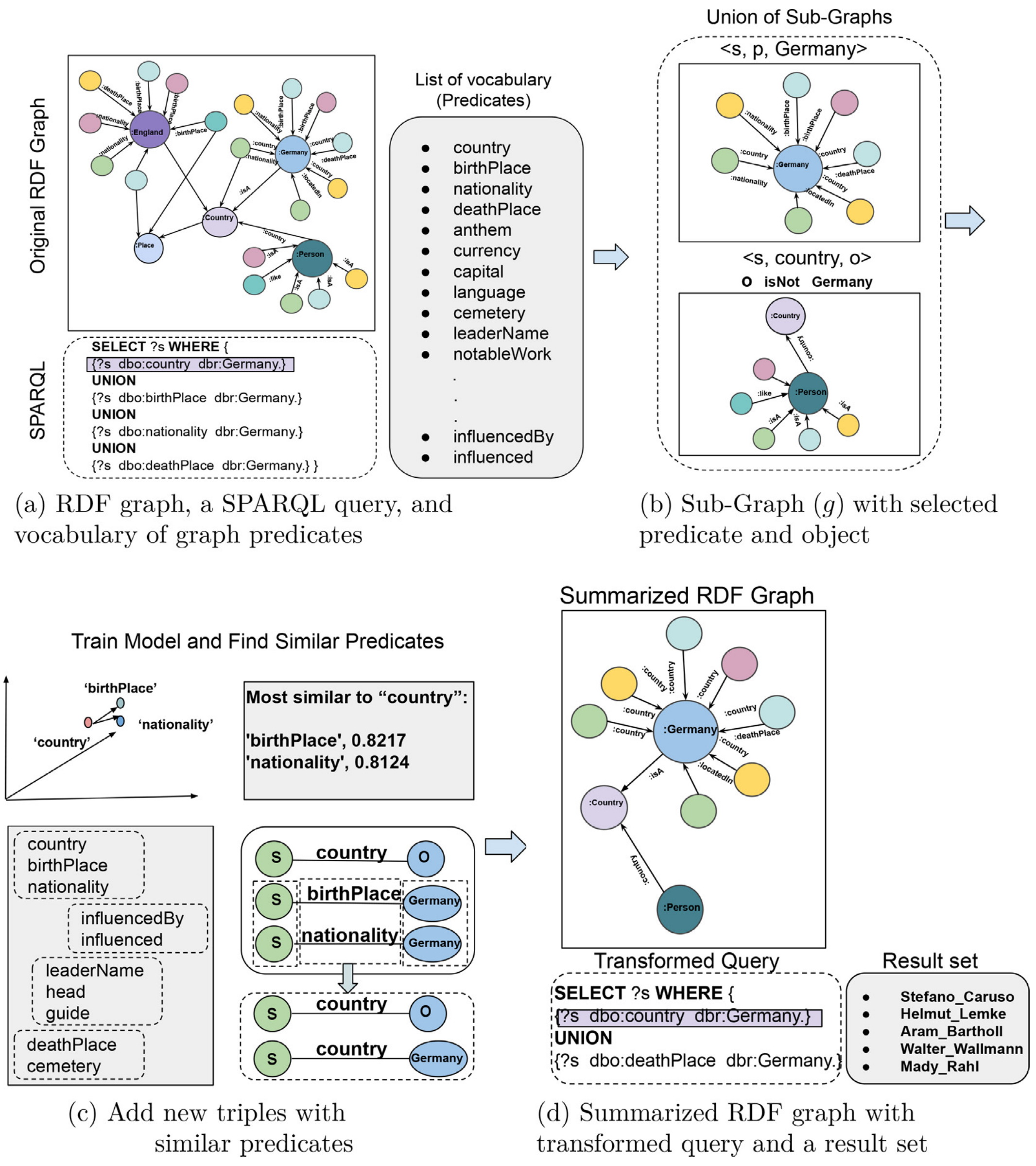

1. Input: Original RDF Graph
2. Load RDF graph
3. SELECT DISTINCT ?s WHERE { {?s dbo:country dbr:Germany.} UNION {?s dbo:birthPlace dbr:Germany.} UNION {?s dbo:nationality dbr:Germany.} UNION {?s dbo:deathPlace dbr:Germany.} }
Extract Super-Predicate and Super-Object
4. Sub-Graph (g) = Retrieve triples include Super-Predicate or Super-Object
5. Find Most similar predicates to "country" by Embedding Model:
'birthPlace', cosineSimilarity = 0.8217
'nationality', cosineSimilarity = 0.8124
6. Super-Subject = Retrieve triples include Edge Candidate Sets
7. New Triples = Super-Subject + Super-Predicate + Super-Object
8. Output: Summarized RDF Graph
9. Output: Transformed SPARQL Query
SELECT DISTINCT ?s WHERE { {?s dbo:country dbr:Germany.} UNION {?s dbo:deathPlace dbr:Germany.}}
10. Output: Set of Results
Stefano_Caruso
Helmut_Lemke
Aram_Bartholl
Walter_Wallmann
Mady_Rahl

Fig. 8. Summary Graph Architecture for the Query Based Summarization (QBS) approach (The Optimized Approach).

Algorithm 3: The Query Based Summarization (QBS) Algorithm.

```
Input  : RDF dataset loaded as RDF graph (G); SPARQL Query (Q); List of Vocabulary (V); Semantic Similarity Metric
Output: Transformed SPARQL Query ($Q''$); Summary Graph ($G''$); List of result
1  G ← RDFGraphLoader.loadFromDisk(spark, input, parallelism) ;
2  Super-Predicates ← Q.getSetOfPredicates ;
3  Super-Objects ← Q.getSetOfObjects ;
4  g ← triples includes Super-Predicates or Super-Objects ;
   /*g is a Sub-Graph*5/initialize training model ;
6  V ← list_of_vocabulary ;
7  foreach vocabulary ∈ V do
8      model ← trained model ;
   end
9  Q.Predicates ← set of predicates extracted from Q ;
10 forall the p ∈ Q.Predicates do
11     similar_set_of_P ← predicates q in g with cosine-similarity(q,P) >0.5 ;
12     Q.replaceBy(P, representativeOf(synonym_set_of_P)) ;
   end
13 $Q''$ ← Q ;
14 tsp ← G.getSetOfPredicates.getURI.contains(similar_set_of_P) ;
   /*tsp is a set of triples contain similar predicates*15/Super-Subjects ← tsp.getSetOfSubjects ;
16 newTriples ← Triple.create(Super-Subjects, Super-Predicates, Super-Objects) ;
17 Buffer ← new ArrayBuffer(triple.length) ;
18 foreach (Super-Subject, Super-Predicate, Super-Object) ∈ G do
19     if newTriples not exists in Buffer then
20         Buffer += newTriples ;
       end
   end
21 $G''$ ← Buffer.union(g) ;
22 result ← $G''$.sparql($Q''$) ;
23 return result
```



```
SELECT ?s WHERE {
{?s dbo:country dbr:Germany.}
UNION
{?s dbo:birthPlace dbr:Germany.}
UNION
{?s dbo:nationality dbr:Germany.}
UNION
{?s dbo:deathPlace dbr:Germany.} }
```

```
Most similar to "country":
'birthPlace', 0.8217
'nationality', 0.8124
```

```
SELECT ?s WHERE {
{?s dbo:country dbr:Germany.}
UNION
{?s dbo:deathPlace dbr:Germany.}
```

(a) RDF graph, a SPARQL query, and vocabulary of graph predicates

(b) Sub-Graph (g) with selected predicate and object

(c) Add new triples with similar predicates

(d) Summarized RDF graph with transformed query and a result set

Fig. 9. Example of Query Based Summarization (QBS). (a) The original RDF graph portion and a SPARQL query with predicate vocabulary before training the model; (b) Predicate and object of one triple pattern are considered as Super-Edge and Super-Object, respectively to extract the Sub-Graph (g); (c) The embedding models encode predicates as vectors to find the most similar predicates to the Super-Edge trained to create new triples; (d) Summary RDF graph with a transformed query and answers.

2. *For SPARQL query Q, where p_1 or p_2 are used in the triple patterns of Q. Q can be a conjunctive query or include the UNION or OPTIONAL operator. If Q'' is the transformation of Q where p_1 and p_2 are replaced by p, the evaluation Q' in G' and the evaluation of Q in G are the same.*

$$[[Q]]G = [[Q'']]G'' \tag{3}$$

Proof. Consider the Query Based Summarization approach, which is dependent on the query of the user to summarize the RDF graph. All the properties and their relations related to the query appear in G''; it is the compact representation of G. Thus, G'' consists of all relations and entities related to the query Q''. Therefore, the properties p_1 and p_2 in G which are similar based on the semantic similarity metric, and they belong to different classes are considered as predicate p in G''. Hence, the cardinality of G'' is equal to the cardinality of G. Again, the query Q with properties of p_1 and p_2 is rewritten to query Q'' with the property of p, and duplicated triple patterns are eliminated. So, query Q over G is the same as query Q'' over G''. □

Fig. 9 illustrates the QBS algorithm with a simple example. A given query in Fig. 9a can also be considered as a multi triple pattern query without union operation. The number of answers by querying over the original RDF graph is equal to the number of answers by querying over the summarized graph. This has been proved in Theorem 4.1.

5. Empirical evaluation

The effectiveness of the approaches described in Section 4 is analyzed based on the size of the RDF graph, the number of retrieved data, scalability, and query processing. We aim to answer the following research questions: **RQ1)** What is the impact of predicate relatedness by analyzing similarity measures in different size of datasets using diverse embedding models on finding complete answers? **RQ2)** Are the proposed RDF summary graph techniques able to reduce the size of the original RDF graph by considering only required triples for querying? **RQ3)** How is the effectiveness of the proposed summarized techniques on answer completeness and cardinality compared with the original RDF graph? **RQ4)** How is the impact of the complexity of query on query processing and execution time in the proposed RDF summary graph compared with the original RDF graph?

5.1. Data preparation and methods

GBS and QBS have been evaluated in terms of compactness, completeness, and execution time. The summarization ratio is calculated by comparing the size of original RDF graphs with the size of summary graphs generated by two summarization approaches. The number of retrieved answers over original graphs is compared with the number of answers by querying over summarized graphs. Finally, the speed of query processing is computed by total time of returning answers by original and summary graphs using Sparklify. One of the most important aspects of our evaluation is to collect and process the dataset to ensure that it is being worked by our approach. Therefore, the first step is preparing datasets to meet our conditions as an input. One condition is the dataset does not contain any literals. Since literal values and properties coming with literal values cannot be embedded by knowledge graph embeddings techniques. Then, RDF dataset should be loaded as an RDF graph. Later, the complex query is evaluated against this graph, and the answers are returned by Sparklify. The number of results and the running time are measured. For the proposed approaches, after loading the RDF dataset as an original RDF graph, the summarization technique is applied to generate a summary RDF graph. The number of answers retrieved by querying transformed queries over the summarized graph and query processing time are compared with the results produced over the RDF graph.

5.2. Experimental setup

We conduct an experimental study to assess the accuracy of our approach compared with the baseline. Our experimental configuration involves the datasets and queries used for our evaluation, as well as metrics and implementation.

5.2.1. Datasets and queries

Four datasets with different sizes are applied to realize the effectiveness of the mentioned techniques. One is a small part of DBpedia dataset consists of *2,047* triples wherein a bunch of companies, persons, and places with some of their information are stored. The second one is an Entity Summarization BenchMark (ESBM) with *6,584* triples which are sample entities from two datasets, DBpedia and LinkedMDB (Hassanzadeh & Consens, 2009) a popular movie database. The other selected datasets are Waterloo SPARQL Diversity Test Suite (WatDiv) with *10,916,457* triples (WatDiv.10M) and with *108,997,714* triples (WatDiv.100M) as medium and large datasets, respectively. Our evaluation is composed of 15 queries selected from QALD-3[7] for DBpedia dataset, from ESBM Benchmark v1.2[8] for ESBM, and from Query Generator (v0.6)[9] for WatDiv.

5.2.2. Metrics

Three main metrics in the evaluation of summary graphs are considered to answer the above research questions. **a)** *Compactness*: the size of summary RDF graph should be typically smaller than the size of original RDF graphs. The Summarization Ratio (SR) is a metric to show the value of compactness; it is calculated by the number of triples in the summary graph divide into the number of triples in the original RDF graph. **b)** *Cardinality*: the number of answers returned by a query over the original RDF graph should be the same as over the summarized graph. **c)** *Execution Time*: the running time for query processing over the summarized RDF graph should be less than over the original RDF graph.

5.2.3. Implementation

The approaches are implemented in Scala 2.11.12 and Spark 2.2.0 over query engine Sparklify from the SANSA Stack framework. Sparklify executed over the original RDF graph is used as the baseline of our work. The proposed algorithms are compared with the original RDF graph based on Sparklify, **i) Sparklify + GBS**; and **ii) Sparklify + QBS**. We evaluate our experiments on three servers with 256 cores and executor memory of 100 GB.

5.3. Impact of predicate relatedness by analyzing similarity measures

The distribution of similar predicates in each RDF graph using diverse embedding models is different. Referring to the research question **RQ1**, the predicate relatedness in each dataset has an impact on finding complete answers. For rewriting queries, the embedding techniques such as Word2Vec and RDF2Vec find similar predicates to the given predicate in the query to transform it. If the most similar and semantically relevant predicates are found, the transformation of a query would be more efficient in terms of finding the complete answers. Since finding the most similar predicates can affect verifying the cardinality property; it becomes an important issue on rewriting queries. During experiments, it is discovered that not only different datasets with various sizes have a different distribution of similarity values, but also the different distribution can be seen in diverse embedding models. It means not only the number of triples in RDF datasets can change the result of predicate relatedness, but also the techniques used for embedding have an impact. Figs. 10 and 11 show the distribution of cosine similarity to find the most similar predicates between the original RDF graph and summarized RDF graphs provided by Grouping Based Summarization(GBS) approach and Query Based Summarization (QBS) approach using word embedding model and graph embedding model, respectively.

The results suggest that the way to find similar predicates depends on the size of the RDF graph and the technique for embedding. As seen, the probability of cosine similarity close to 1 in QBS approach is higher than the probability in the original graph and summary graph by GBS approach. The reason is, summary graphs in QBS have much fewer and more similar triples compared with the original ones. In QBS approach, only a part of the RDF graph related to the query has been considered, not the whole graph. Hence, by considering the small part of RDF graph includes triples with the same objects, similar and relevant predicates are close to each other and the distance between them is less. Therefore, by decreasing the number of triples in the dataset and keeping relevant triples to the query, the probability of finding the most similar predicates to a given one in its neighborhood is higher. In GBS approach, the probability of having a cosine similarity close to 0.5 using word embedding and close to 0.7 using graph embedding is higher. The reason is, after summarizing RDF graphs based on an explanation in Section 4.3, many relevant triples are lost. So, the embedding model finds the properties which are not completely relevant and similar. If less relevant similar predicates are found for rewriting the query, then query over the summarized RDF graph will return incomplete answers. Hence, in GBS approach, applying a query over the summarized graph does not return the complete answers. Moreover, these observations are more obvious in Fig. 11 where RDF2Vec model used as graph embedding model to find similar properties. RDF2Vec employs different walking strategies such as Random Walk, NGram Walk, HALK Walk, Walklet Walk, and Anonymous Walk (Vandewiele et al., 2020). In this work, Random Walk is applied to extract the walks over the knowledge graphs. After finding the proper threshold, only predicates with cosine similarity

7 http://qald.aksw.org/index.php?x=task1&q=3.

8 https://w3id.org/esbm/.

9 https://dsg.uwaterloo.ca/watdiv/#download.

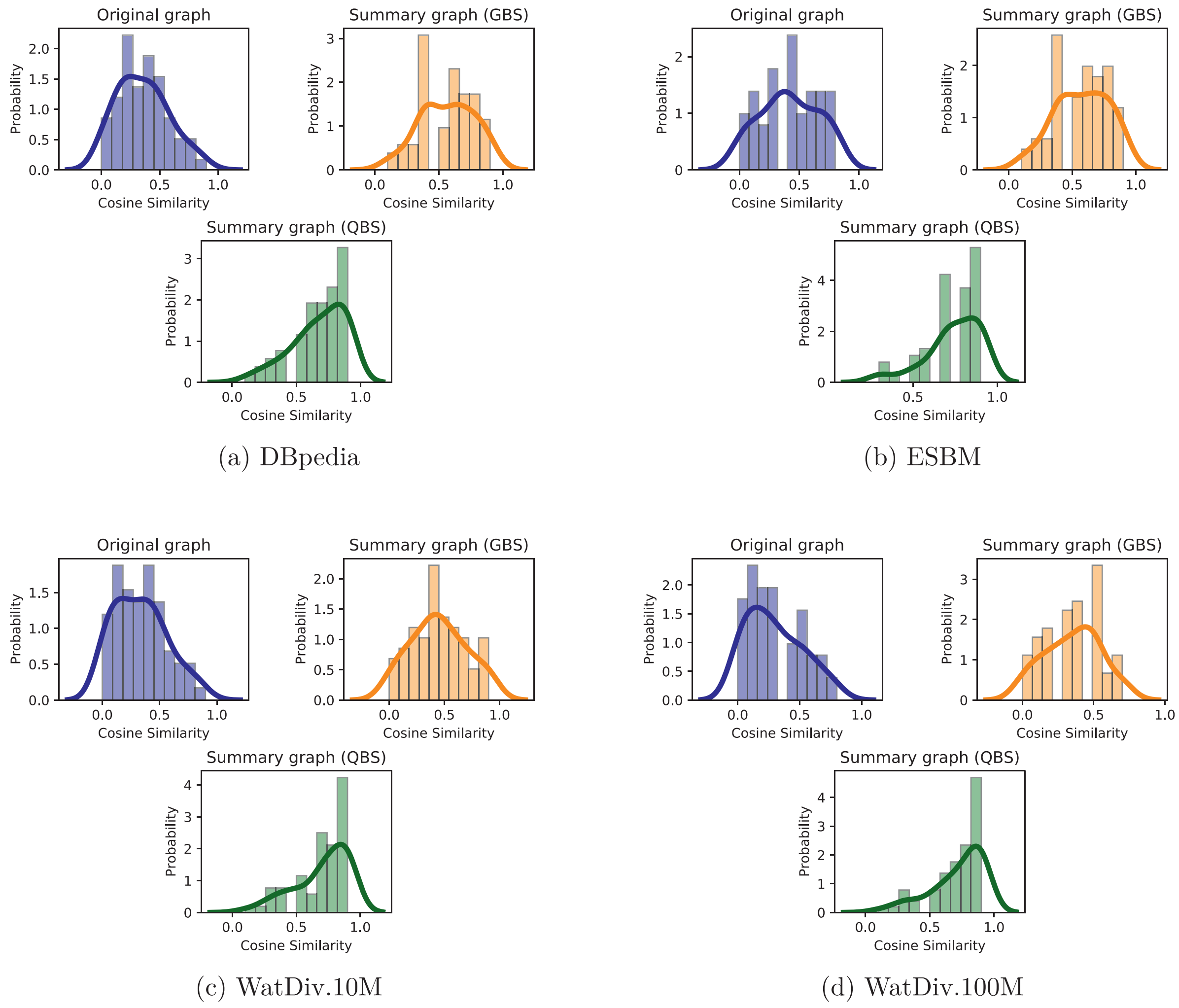


Fig. 10. Compare the distribution of cosine similarity for predicate relatedness in the original and the summarized RDF graph based on GBS and QBS approaches using word embedding model.

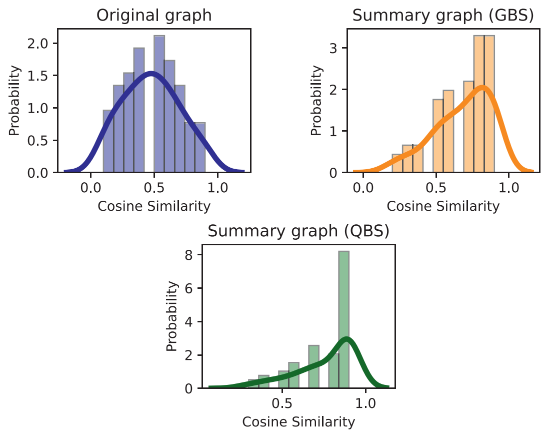
Table 1
RDF Graphs Summarization Ratio (SR) and Summarization Time (ST).

Dataset	#Triples of original RDF graph	GBS Approach SR	GBS Approach ST	QBS Approach SR	QBS Approach ST
DBpedia	2,047	32.2 %	36.8 s	99%	2.8 s
ESBM	6,584	33%	34.2 s	98.8%	2.2 s
WatDiv.10M	10,916,457	41.8%	100 s	96.6%	14.2 s
WatDiv.100M	108,997,714	57%	787.6 s	97.4%	53.6 s

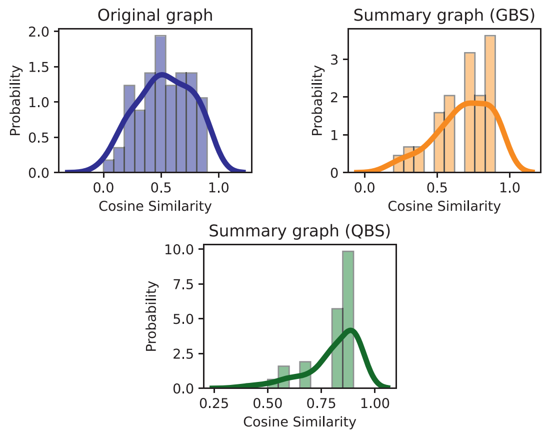
greater than the threshold are considered as synonyms to rewrite the queries. The predicate relatedness by analyzing similarity measures in RDF graphs plays an important role in transforming the query to retrieve complete answers. In QBS approach, by finding similar predicates using different embedding models, the transformation of the query has been done efficiently and leads to retrieving complete results.

5.4. Effectiveness of proposed summarized graph

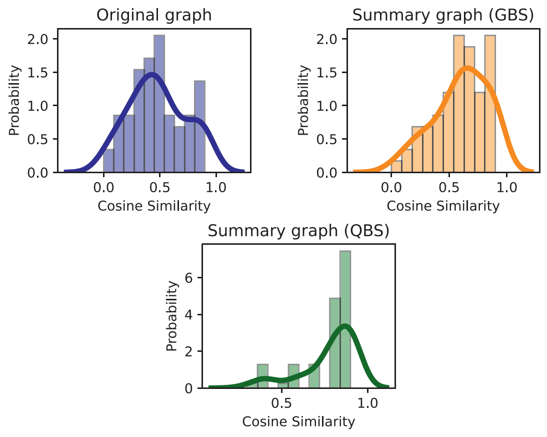
To evaluate the effectiveness of proposed approaches and answer research questions **RQ2** and **RQ3**, the results of experiments are reported based on reducing the size of the RDF graph and returning the complete answers. As seen in Table 1, the Summarization Ratio (SR) for QBS approach is higher than GBS approach. It means the number of triples in the summarized graph provided by QBS approach is less than the number of triples in the summarized graph provided by GBS approach. In QBS approach by considering a query of the user and the part of the original RDF graph which is related to this query, the Summarization Ratio (SR) has significantly increased compared with GBS approach. The compactness property has been verified in both approaches, but it has a better result in QBS approach. For example, the QBS approach makes the dump of DBpedia dataset *99%* smaller than the original dataset, while by GBS approach it gets only *32.2%* smaller than the original one. Therefore, the compactness based on QBS approach is more than GBS approach. Also, Table 1 provides the Summarization Time (ST) which is the time required to generate the summary graph in both approaches based on the given datasets. Here, it should be mentioned that summarizing the RDF graph in QBS approach is not only related to the size of dataset but also is based on the query of the user. Therefore, the size and time provided here for generating the summarization graph is the average of the size and time among different queries. Moreover, Table 2 shows the result of applying the SPARQL queries with multi triple patterns over the original RDF graph and compares them with the result of applying the transformation of these queries over summarized graphs provided by both approaches. Based on the results provided here the cardinality in GBS approach has been verified only for small datasets and not for medium and large datasets, while based on Theorem 4.1 in QBS approach all information related to the query will be found in the



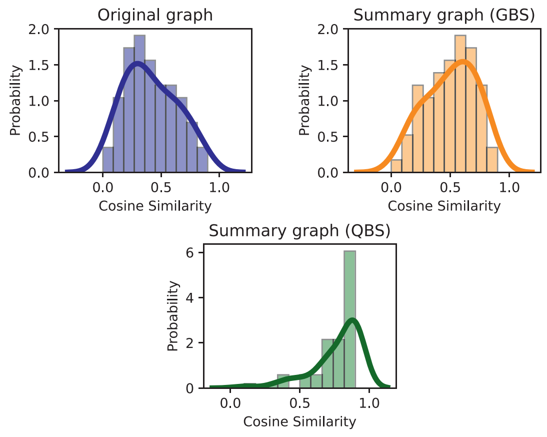
(a) DBpedia



(b) ESBM



(c) WatDiv.10M



(d) WatDiv.100M

Fig. 11. Compare the distribution of cosine similarity for predicate relatedness in the original and the summarized RDF graph based on GBS and QBS approaches using graph embedding model.

Table 2
Compare the number of answers by querying multi triple patterns query over the original graph and transformed query as simple query over the summarized graph provided by Grouping Based Summarization (GBS) approach and Query Based Summarization (QBS) approach using the query engine Sparklify.

		# of answers		
	⟶	Sparklify	Sparklify+GBS	Sparklify+QBS
DBpedia	*Q*1	5	5	5
	*Q*2	5	5	5
	*Q*3	9	9	9
	*Q*4	6	6	6
	*Q*5	8	8	8
ESBM	*Q*6	3	3	3
	*Q*7	2	2	2
	*Q*8	5	5	5
	*Q*9	8	8	8
	*Q*10	6	6	6
WatDiv.10M	*Q*11	86	54	86
	*Q*12	56	30	56
	*Q*13	99	64	99
	*Q*14	3,834	2,043	3,834
	*Q*15	59	26	59
WatDiv.100M	*Q*16	59	31	59
	*Q*17	554	206	554
	*Q*18	983	489	983
	*Q*19	38,895	21,089	38,895
	*Q*20	146	85	146

summarized graph. Therefore, in QBS approach, the cardinality for all size of datasets has been verified. In order to return the answers for SPARQL queries, Sparklify, the scalable component which is the default query engine in the SANSA Stack, has been used during all the experiments. First, the query is applied over the original RDF graph. Later, the transformation of this query which is a simple query with fewer triple patterns is applied to the summary graph. As seen in Table 2, the number of answers retrieved in GBS is less than the number of answers from the original RDF graph which means during summarizing some information is lost while in QBS the number of answers retrieved from the summarized graph is equal to the number of answers from the original RDF graph. Thus, in QBS, the cardinality property has been verified.

5.5. *Efficiency of transforming queries*

To answer the research question **RQ4**, we evaluate the execution time of query processing in the original RDF graph and the proposed summarized graphs provided by both approaches, GBS and QBS. The query processing has been done by the query engine, Sparklify. We report on the average query processing time after three times running the algorithms.

Table 3 shows the comparison of executing time over baseline and combing it with proposed approaches. Also, to verify that the query engine Sparklify combined with QBS approach to query over the summarized RDF graph achieve a greater reduction in execution time than

Table 3
Compare the execution time in the original and the summarized RDF graph provided by Grouping Based Summarization (GBS) and Query Based Summarization (QBS) approaches. Q is the query over the original RDF graph and Q' is the query over the summarized RDF graph. S represents as baseline Sparklify executed over the original RDF graph.

		Execution Time (seconds)				
	$\longrightarrow$	S(Total)	S+GBS(SUM)	S+GBS(QA)	S+QBS(SUM)	S+QBS(QA)
DBpedia	$Q1/Q'1$	19	37	6	2	3
	$Q2/Q'2$	18	39	6	3	5
	$Q3/Q'3$	29	45	9	3	6
	$Q4/Q'4$	18	30	5	3	5
	$Q5/Q'5$	17	33	6	3	4
ESBM	$Q6/Q'6$	32	45	9	3	5
	$Q7/Q'7$	19	36	7	2	4
	$Q8/Q'8$	20	31	5	2	6
	$Q9/Q'9$	20	30	6	2	5
	$Q10/Q'10$	19	29	5	2	4
WatDiv.10M	$Q11/Q'11$	85	102	56	14	32
	$Q12/Q'12$	79	92	47	11	28
	$Q13/Q'13$	80	89	78	15	26
	$Q14/Q'14$	82	123	158	13	24
	$Q15/Q'15$	98	94	38	18	28
WatDiv.100M	$Q16/Q'16$	321	802	274	88	109
	$Q17/Q'17$	147	541	349	42	56
	$Q18/Q'18$	192	627	482	41	86
	$Q19/Q'19$	401	1,190	850	63	168
	$Q20/Q'20$	182	778	321	34	105

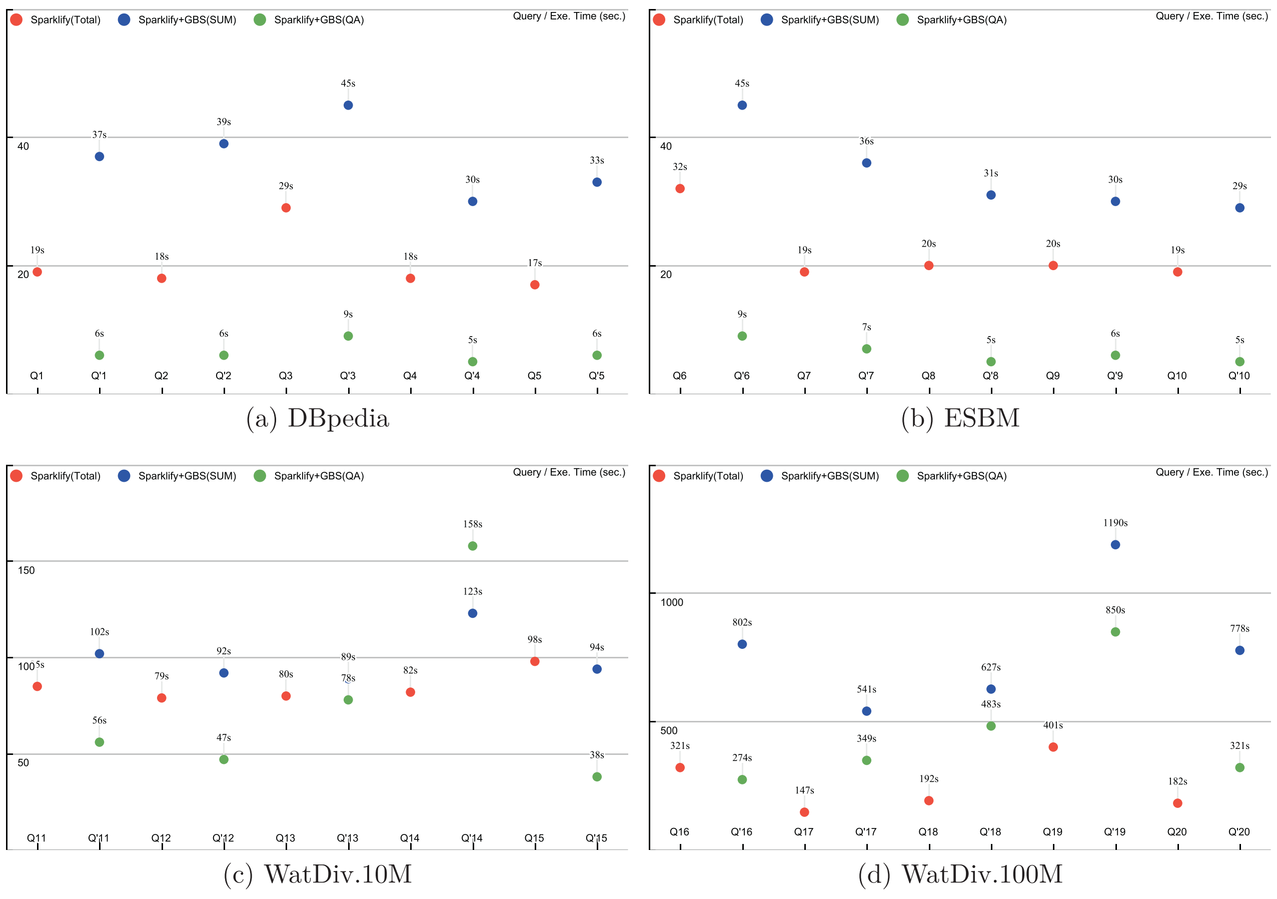


Fig. 12. Comparing execution time in original RDF graph and summary RDF graph provided by Grouping Based Summary Graph (GBS) approach over four different datasets with different sizes. Q and Q' are executed over the original and summarized RDF graphs, respectively. Red dots show the execution time (in seconds) to answer a query over the original RDF graph. Blue and green dots show the time for summarizing RDF graph (SUM) and the time for query answering (QA) over summary graph, respectively. As seen, by increasing the size of datasets, execution time in the summarized graph is much higher than the original graph. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

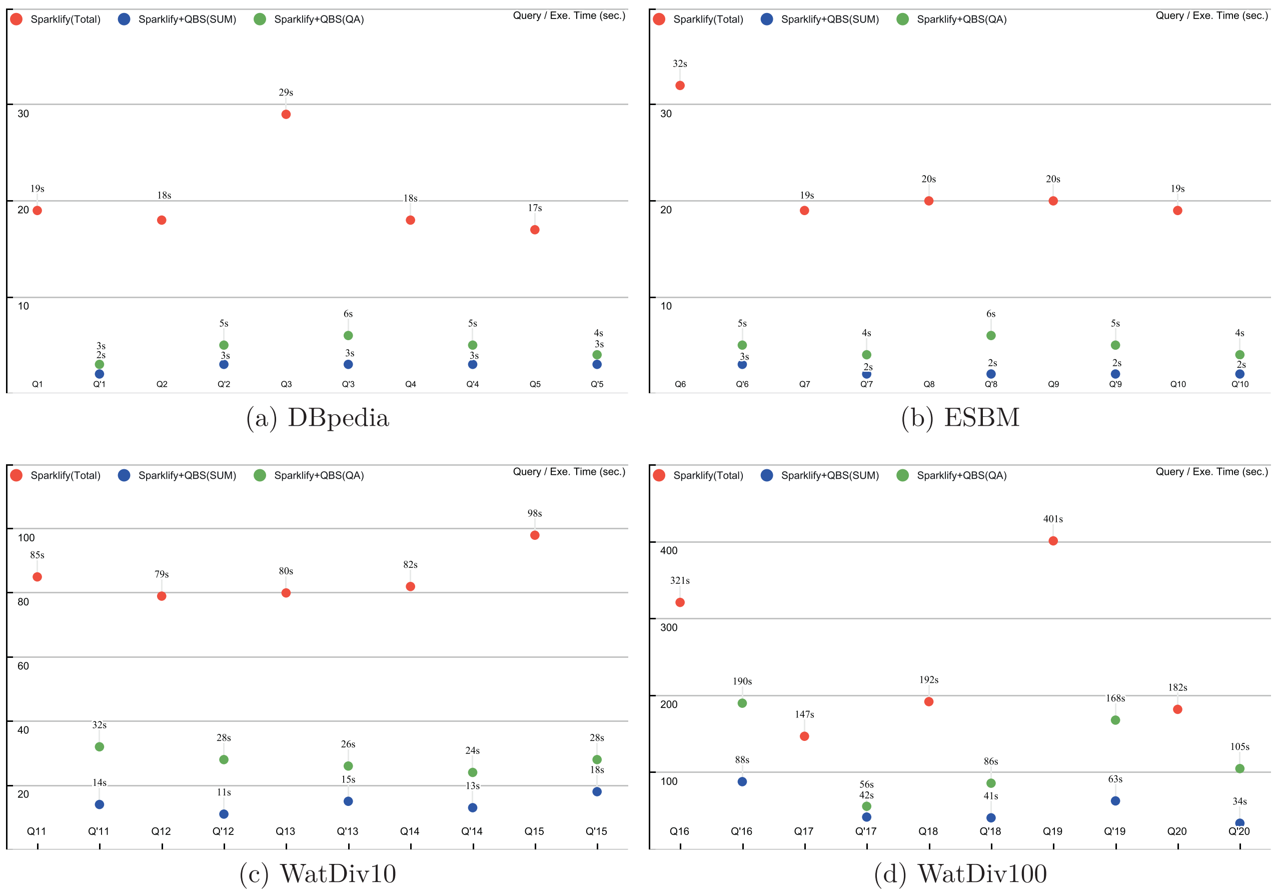


Fig. 13. Execution time in original and summary RDF graphs using Query Based Summary Graph (QBS) on four datasets. Q and Q' are queries over original and summarized RDF graphs, respectively. As seen, by increasing the size of datasets, execution time (in secs.) in the summarized graph is much less than in the original RDF graph. Thus, the execution time decreases when the dataset size on QBS increases.

using the Sparklify over the original RDF graph, a Wilcoxon signed rank test is run with the result of p-value $< 6.103e-05$. As a result, the observed outcomes indicate a reduction in execution time of query processing over summarized RDF graphs generated by QBS. The execution time of Sparklify against the original RDF graph is the baseline. Moreover, both query processing time distributions differ significantly.

Fig. 12 illustrates the execution time of 20 SPARQL queries over the original RDF graph compared with the summarized graph generated by Grouping Based Summary Graph (GBS) approach. The execution time is the summation of time for summarizing the RDF graph (SUM) and time for query answering (QA). As seen, execution time in summarized graph provided by GBS specially in larger datasets (Fig. 12c and d) is much higher than the execution time in original RDF graph. Fig. 13 shows the same experiments but over summarized RDF graph generated based on Query Based Summary Graph (QBS) approach in four different size of datasets. As seen in Fig. 13, the execution time in summarized graph provided by QBS specially in larger datasets (Fig. 13c and d) is much less than the execution time in original RDF graph. The approach of QBS speeds up the execution time by up to 80% by summarizing the RDF graph. Therefore, the execution time of query processing in the summarized graph of QBS approach is much less than in the summarized graph of GBS approach. Also, by comparing the summarizing time (SUM) and querying answering (QA) in both approaches it is observed that the time for summarizing in QBS approach is less than the time for querying, while in GBS approach the summarizing time (SUM) is higher than query answering (QA) time. In general, all the observations show the optimization of the grouped based approach, GBS, to the query based approach, QBS, in terms of compactness, completeness, and execution time.

6. Discussion

The presented experimental results confirm that the proposed RDF graph summarization technique is able to reduce the size of RDF graph by preserving necessary information. Moreover, the significant decrease in execution time is observed during query processing.

6.1. Contribution to literature

The results of our summarization approaches are compared with the SPARQL query engine, Sparklify, as a baseline over the original RDF graph. The number of retrieved answers and the execution time over the summarized graph compared with the original RDF graph are shown in Tables 2 and 3, respectively. The results describe that the query processing over proposed summarized RDF graph is superior to querying over original RDF graph. Table 4 provides an overview of existing summarization methods mentioned in Section 2, and compared with our approach. As seen, summarization methods in a large variety of concepts are different. However, QBS is able to not only reduce the size of the graph, but also speed up execution time. Thus, we define data management methods that can be used to enrich the portfolio of frameworks for managing and querying larger RDF graphs. Given the rapid increase of large RDF datasets, these methods will play a relevant role in scalability, providing thus the basis for the development of real-world applications.

6.2. Practical implication

The proposed approach is aimed to ensure the compactness, completeness, and improve execution time. It focuses on reducing the size

Table 4
Comparison of some graph summarization techniques.

Research	Summary type	Input	Technique	Output Graph	Purpose
ASSG by Zhang et al. (2014)	Structural RDF	Instance	Compression	Compressed Graph	Query Answering
Gurajada et al. (2014b)	Structural RDF	Instance	Partitioning	RDF Graph	Query Answering
Sydow et al. (2013)	Structural RDF	Instance	Selecting Sub-Graph	RDF Graph	Visualization
Zneika et al. (2016)	Pattern Mining	Instance	Approximate Graph Patterns	RDF Graph	Query Answering
Karim et al. (2020)	Pattern Mining	RDF Graph	Factorization	Factorized RDF Graph	Query Processing
CoSum by Zhu et al. (2016)	Statistical RDF	k-type	Grouping	Super	Entity Resolution
SPARQL Similarity Search by Zheng et al. (2016)	Hybrid RDF	Instance and Schema	Structural and Pattern Mining	Multi-layer Graph	Query Optimization
GBS (Proposed Naïve Approach)	Hybrid RDF	RDF Graph	Group based Summarization	Multi Sub-Graphs	Query Processing
QBS (Proposed Smart Approach)	Hybrid RDF	RDF Graph	Query based Summarization	Selective RDF Triples	Query Processing

of RDF graph by providing the lossless and low-cost query processing. Although our approach performed better than the baseline, there are still some issues to discuss. For example, the assumption in our work is the original knowledge graphs consist of synonyms properties. In case the knowledge graph has few or no synonyms for properties, the summarization cannot be done properly. In the naive summarizing approach (GBS), the size of the RDF graph has vital impact on rate of summarization and time of execution. On the other hand, in the optimized QBS approach, the summarization ratio and execution time has direct relationship to the complexity of queries. Transformation of these complex queries to the simple ones based on similarity measures needs to be done efficiently. Defining the proper candidate sets helps queries be transformed in a way to retrieve the complete answers in less execution time. The techniques and methods to select and prune candidate sets still is an issue to discuss. However, our optimized summarization approach can be applied to any RDF knowledge graphs with synonymous properties. Each embedding technique has its own advantages and disadvantages on a variety of knowledge graphs. Except embedding techniques used in this paper, other techniques can be studied and compared. Moreover, other state-of-the-art summarization techniques with more evaluation criteria such as recall and precision to assess the accuracy of our approach can be added to the empirical evaluation. These limitations of our work needs to be addressed in future work.

7. Conclusions and future work

In this paper, we tackled the challenge of RDF graph summarization to optimize query processing. The proposed techniques contribute to the portfolio of tools to efficiently manage knowledge graphs, which is of significant relevance given the role of knowledge graphs in knowledge representation. We presented the Grouping Based Summarization (GBS) and the Query Based Summarization (QBS) approaches. QBS optimizes GBS and ensures query completeness. Technically, we implemented our solutions on top of the state-of-the-art SANSA Stack, allowing our methods to run on large-scale RDF graphs using Apache Spark as a process engine. From the evaluation results, QBS provides a good Summarizing Ratio (SR) from *96%* to *99%* in terms of the number of triples that are needed for evaluation. From the query time point of view, there is a notable reduction in query execution, up to *80%* in the proposed Query Based Summarization (QBS) approach compared with the original RDF graph queried by Sparklify. This less complexity in QBS has led to a significant improvement compared with the original RDF graph. With due attention to the investigations and results obtained from the experiments in this work, the importance of summarizing large-scale RDF graphs using the SANSA framework becomes more distinct.

In the future, we will focus on extending the proposed summarization RDF graph with different types of data from structured to unstructured to retrieve complete results with a reduction in execution time. Also, we intend to evaluate the proposed method in other knowledge bases such as WikiData.

Acknowledgments

This work has been partially supported by the EU H2020 projects CLARIFY [grant number 875160] and PLATOON [grant number 872592]; and the EraMed project P4-LUCAT [grant number 53000015].

References

Aluç, G., Hartig, O., Özsu, M. T., … Daudjee, K. (2014). Diversified stress testing of RDF data management systems. In P. Mika, T. Tudorache, A. Bernstein, C. Welty, C. A. Knoblock, D. Vrandecic, … C. A. Goble (Eds.), *The semantic web - ISWC 2014 - 13th international semantic web conference, riva del garda, italy, october 19–23, 2014. proceedings, part I*. In *Lecture Notes in Computer Science: vol. 8796* (pp. 197–212). Springer. 10.1007/978-3-319-11964-9_13.

Benjelloun, O., Garcia-Molina, H., Menestrina, D., Su, Q., Whang, S. E., & Widom, J. (2009). Swoosh: A generic approach to entity resolution. *The VLDB Journal: Very Large Data Bases: A Publication of the VLDB Endowment, 18*(1), 255–276. 10.1007/s00778-008-0098-x.

Bonifati, A., Dumbrava, S., & Kondylakis, H. (2020). Graph summarization. *CoRR, abs/2004.14794*. https://arxiv.org/abs/2004.14794.

Bourbakis, N. (1998). Artificial intelligence and automation. *Advanced Series on Statistical Science and Applied Probability*. World Scientific. https://books.google.de/books?id=mV3wxKLHlnwC.

Cebiric, S., Goasdoué, F., Kondylakis, H., Kotzinos, D., Manolescu, I., Troullinou, G., & Zneika, M. (2019). Summarizing semantic graphs: A survey. *The VLDB Journal: Very Large Data Bases: A Publication of the VLDB Endowment, 28*(3), 295–327. 10.1007/s00778-018-0528-3.

Chauhan, T., & Palivela, H. (2021). Optimization and improvement of fake news detection using deep learning approaches for societal benefit. *International Journal of Information Management Data Insights, 1*(2), 100051. 10.1016/j.jjimei.2021.100051.

Consens, M. P., Fionda, V., Khatchadourian, S., & Pirrò, G. (2015). S+epps: Construct and explore bisimulation summaries, plus optimize navigational queries; all on existing SPARQL systems. *Proceedings of the VLDB Endowment, 8*(12), 2028–2031. 10.14778/2824032.2824128.

Faye, D., Curé, O., & Blin, G. (2012). A survey of rdf storage approaches. *Revue Africaine de la Recherche en Informatique et Mathématiques Appliquées, 15*, pp.25. 10.46298/arima.1956.

Gurajada, S., Seufert, S., Miliaraki, I., & Theobald, M. (2014a). Triad: a distributed shared-nothing RDF engine based on asynchronous message passing. In C. E. Dyreson, F. Li, & M. T. Özsu (Eds.), *International conference on management of data, SIGMOD 2014, snowbird, ut, usa, june 22–27, 2014* (pp. 289–300). ACM. 10.1145/2588555.2610511.

Gurajada, S., Seufert, S., Miliaraki, I., & Theobald, M. (2014b). Using graph summarization for join-ahead pruning in a distributed RDF engine. In *Proceedings of the sixth workshop on semantic web information management, SWIM 2014, snowbird, ut, usa, june 22–27, 2014* (pp. 41:1–41:4). ACM. 10.1145/2630602.2630610.

Harris, S., Seaborne, A., & Prud'hommeaux, E. (2013). Sparql 1.1 query language. *W3C Recommendation, 21*(10), 778. https://www.w3.org/TR/sparql11-query/.

Hassanzadeh, O., & Consens, M. P. (2009). Linked movie data base. In C. Bizer, T. Heath, T. Berners-Lee, & K. Idehen (Eds.), *Proceedings of the WWW2009 workshop on linked data on the web, LDOW 2009, madrid, spain, april 20, 2009*. In *CEUR Workshop Proceedings: vol. 538*. CEUR-WS.org. http://ceur-ws.org/Vol-538/ldow2009_paper12.pdf.

Jatnika, D., Bijaksana, M., & Ardiyanti, A. (2019). Word2vec model analysis for semantic similarities in english words. *Procedia Computer Science, 157*, 160–167. 10.1016/j.procs.2019.08.153.

Jurafsky, D., & Martin, J. H. (2009). Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition, 2nd edition. *Prentice Hall series in artificial intelligence*. Prentice Hall, Pearson Education International. https://www.worldcat.org/oclc/315913020.

Kaoudi, Z., & Manolescu, I. (2015). RDF In the clouds: A survey. *The VLDB Journal: Very Large Data Bases: A Publication of the VLDB Endowment, 24*(1), 67–91. 10.1007/s00778-014-0364-z.

Karim, F., Vidal, M., & Auer, S. (2020). Compacting frequent star patterns in RDF graphs. *Journal of Intelligent Information Systems, 55*(3), 561–585. 10.1007/s10844-020-00595-9.

Kondylakis, H., Kotzinos, D., & Manolescu, I. (2019). RDF graph summarization: principles, techniques and applications. In M. Herschel, H. Galhardas, B. Reinwald, I. Fundulaki, C. Binnig, & Z. Kaoudi (Eds.), *Advances in database technology - 22nd international conference on extending database technology, EDBT 2019, lisbon, portugal, march 26–29, 2019* (pp. 433–436). OpenProceedings.org. 10.5441/002/edbt.2019.38.

Kubitza, D. O., Böckmann, M., … Graux, D. (2019). Semangit: A linked dataset from git. In C. Ghidini, O. Hartig, M. Maleshkova, V. Svátek, I. F. Cruz, A. Hogan, … F. Gandon (Eds.), *The semantic web - ISWC 2019 - 18th international semantic web conference, Auckland, New Zealand, October 26–30, 2019, proceedings, part II*. In *Lecture Notes in Computer Science: vol. 11779* (pp. 215–228). Springer. 10.1007/978-3-030-30796-7_14.

LeFevre, K., & Terzi, E. (2010). Grass: Graph structure summarization. In *Proceedings of the SIAM international conference on data mining, SDM 2010, April 29, - May 1, 2010, Columbus, ohio, USA* (pp. 454–465). SIAM. 10.1137/1.9781611972801.40.

Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., … Bizer, C. (2015). Dbpedia - A large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web, 6*(2), 167–195. 10.3233/SW-140134.

Lehmann, J., Sejdiu, G., Bühmann, L., Westphal, P., Stadler, C., Ermilov, I., … Jabeen, H. (2017). Distributed semantic analytics using the SANSA stack. In C. d'Amato, M. Fernández, V. A. M. Tamma, F. Lécué, P. Cudré-Mauroux, J. F. Sequeda, … J. Heflin (Eds.), *The semantic web - ISWC 2017 - 16th international semantic web conference, vienna, austria, october 21–25, 2017, proceedings, part II*. In *Lecture Notes in Computer Science: vol. 10588* (pp. 147–155). Springer. 10.1007/978-3-319-68204-4_15.

Liu, Q., Cheng, G., Gunaratna, K., … Qu, Y. (2020). In A. Harth, S. Kirrane, A. N. Ngomo, H. Paulheim, A. Rula, A. L. Gentile, … M. Cochez (Eds.), *ESBM: an entity summarization benchmark: (vol.12123* (pp. 548–564)). Springer.

Manola, F., Miller, E., McBride, B., et al., (2004). RDF Primer. *W3C recommendation, 10*(1–107), 6. https://www.w3.org/TR/rdf11-primer/.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*. http://arxiv.org/abs/1301.3781.

Mishra, R. K., Urolagin, S., Jothi, J. A. A., Neogi, A. S., & Nawaz, N. (2021). Deep learning-based sentiment analysis and topic modeling on tourism during covid-19 pandemic. *Frontiers of Computer Science, 3*, 775368. 10.3389/fcomp.2021.775368.

Neogi, A. S., Garg, K. A., Mishra, R. K., & Dwivedi, Y. K. (2021). Sentiment analysis and classification of indian farmers' protest using twitter data. *International Journal of Information Management Data Insights, 1*(2), 100019. 10.1016/j.jjimei.2021.100019.

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. In A. Moschitti, B. Pang, & W. Daelemans (Eds.), *Proceedings of the 2014 conference on empirical methods in natural language processing, EMNLP 2014, October 25–29, 2014, Doha, qatar, A meeting of sigdat, a special interest group of the ACL* (pp. 1532–1543). ACL. 10.3115/v1/d14-1162.

Pérez, J., Arenas, M., & Gutiérrez, C. (2009). Semantics and complexity of SPARQL. *ACM Transactions on Database Systems, 34*(3), 16:1–16:45. 10.1145/1567274.1567278.

Qun, C., Lim, A., & Ong, K. W. (2003). D(k)-index: An adaptive structural summary for graph-structured data. In A. Y. Halevy, Z. G. Ives, & A. Doan (Eds.), *Proceedings of the 2003 ACM SIGMOD international conference on management of data, San Diego, California, USA, June 9–12, 2003* (pp. 134–144). ACM. 10.1145/872757.872776.

Ristoski, P., … Paulheim, H. (2016). Rdf2vec: RDF graph embeddings for data mining. In P. Groth, E. Simperl, A. J. G. Gray, M. Sabou, M. Krötzsch, F. Lécué, … Y. Gil (Eds.), *The semantic web - ISWC 2016 - 15th international semantic web conference, Kobe, Japan, October 17–21, 2016, proceedings, part I*. In *Lecture Notes in Computer Science: vol. 9981* (pp. 498–514). 10.1007/978-3-319-46523-4_30.

Ristoski, P., Rosati, J., Noia, T. D., Leone, R. D., & Paulheim, H. (2019). Rdf2vec: RDF graph embeddings and their applications. *Semantic Web, 10*(4), 721–752. 10.3233/SW-180317.

Shin, K., Ghoting, A., Kim, M., & Raghavan, H. (2019). Sweg: Lossless and lossy summarization of web-scale graphs. In L. Liu, R. W. White, A. Mantrach, F. Silvestri, J. J. McAuley, R. Baeza-Yates, & L. Zia (Eds.), *The world wide web conference, WWW 2019, san francisco, ca, usa, may 13–17, 2019* (pp. 1679–1690). ACM. 10.1145/3308558.3313402.

Singh, K., Devi, S., Devi, H., & Mahanta, A. (2022). A novel approach for dimension reduction using word embedding: An enhanced text classification approach. *International Journal of Information Management Data Insights, 2*(1), 100061. 10.1016/j.jjimei.2022.100061.

Spahiu, B., Maurino, A., & Palmonari, M. (2018). Towards improving the quality of knowledge graphs with data-driven ontology patterns and SHACL. In M. G. Skjæveland, Y. Hu, K. Hammar, V. Svátek, & A. Lawrynowicz (Eds.), *Proceedings of the 9th workshop on ontology design and patterns (WOP 2018) co-located with 17th international semantic web conference (ISWC 2018), monterey, usa, october 9th, 2018*. In *CEUR Workshop Proceedings: vol. 2195* (pp. 52–66). CEUR-WS.org. http://ceur-ws.org/Vol-2195/research_paper_2.pdf.

Stadler, C., Sejdiu, G., Graux, D., … Lehmann, J. (2019). Sparklify: A scalable software component for efficient evaluation of SPARQL queries over distributed RDF datasets. In C. Ghidini, O. Hartig, M. Maleshkova, V. Svátek, I. F. Cruz, A. Hogan, … F. Gandon (Eds.), *The semantic web - ISWC 2019 - 18th international semantic web conference, Auckland, New Zealand, October 26–30, 2019, proceedings, part II*. In *Lecture Notes in Computer Science: vol. 11779* (pp. 293–308). Springer. 10.1007/978-3-030-30796-7_19.

Sydow, M., Pikula, M., & Schenkel, R. (2013). The notion of diversity in graphical entity summarisation on semantic knowledge graphs. *Journal of Intelligent Information Systems, 41*(2), 109–149. 10.1007/s10844-013-0239-6.

Vandewiele, G., Steenwinckel, B., Bonte, P., Weyns, M., Paulheim, H., Ristoski, P., … Ongenae, F. (2020). Walk extraction strategies for node embeddings with rdf2vec in knowledge graphs. *CoRR, abs/2009.04404*. https://arxiv.org/abs/2009.04404.

Vidal, M., Ruckhaus, E., Lampo, T., Martínez, A., Sierra, J., & Polleres, A. (2010). Efficiently joining group patterns in SPARQL queries. In L. Aroyo, G. Antoniou, E. Hyvönen, A. ten Teije, H. Stuckenschmidt, L. Cabral, & T. Tudorache (Eds.), *The semantic web: Research and applications, 7th extended semantic web conference, ESWC 2010, heraklion, crete, greece, may 30, - june 3, 2010, proceedings, part I*. In *Lecture Notes in Computer Science: vol. 6088* (pp. 228–242). Springer. 10.1007/978-3-642-13486-9_16.

Vrandecic, D., & Krötzsch, M. (2014). Wikidata: A free collaborative knowledgebase. *Communications of the ACM, 57*(10), 78–85. 10.1145/2629489.

Zhang, H., Duan, Y., Yuan, X., & Zhang, Y. (2014). ASSG: Adaptive structural summary for RDF graph data. *CEUR Workshop Proceedings, 1272*, 233–236. http://ceur-ws.org/Vol-1272/paper_25.pdf.

Zheng, W., Zou, L., Peng, W., Yan, X., Song, S., & Zhao, D. (2016). Semantic SPARQL similarity search over RDF knowledge graphs. *Proceeding of the VLDB Endowment, 9*(11), 840–851. 10.14778/2983200.2983201.

Zhu, L., Ghasemi-Gol, M., Szekely, P. A., Galstyan, A., … Knoblock, C. A. (2016). Unsupervised entity resolution on multi-type graphs. In P. Groth, E. Simperl, A. J. G. Gray, M. Sabou, M. Krötzsch, F. Lécué, … Y. Gil (Eds.), *The semantic web - ISWC 2016 - 15th international semantic web conference, Kobe, Japan, October 17–21, 2016, proceedings, part I*. In *Lecture Notes in Computer Science: vol. 9981* (pp. 649–667). 10.1007/978-3-319-46523-4_39.

Zneika, M., Lucchese, C., Vodislav, D., & Kotzinos, D. (2016). Summarizing linked data RDF graphs using approximate graph pattern mining. In E. Pitoura, S. Maabout, G. Koutrika, A. Marian, L. Tanca, I. Manolescu, & K. Stefanidis (Eds.), *Proceedings of the 19th international conference on extending database technology, EDBT 2016, bordeaux, france, march 15–16, 2016, Bordeaux, France, March 15–16, 2016* (pp. 684–685). OpenProceedings.org. 10.5441/002/edbt.2016.86.