Towards Empty Answers in SPARQL: Approximating Querying with RDF Embedding

Meng Wang[1(✉)], Ruijie Wang[2], Jun Liu[3], Yihe Chen[4], Lei Zhang[5], and Guilin Qi[6]

[1] MOEKLINNS Lab, Xi'an Jiaotong University, Xi'an, China
wangmengsd@stu.xjtu.edu.cn
[2] School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an, China
xjdwrj@stu.xjtu.edu.cn
[3] Guang Dong Xi'an Jiaotong University Academy, Shunde, China
liukeen@xjtu.edu.cn
[4] University of Toronto, Toronto, Canada
[5] FIZ Karlsruhe – Leibniz Institute for Information Infrastructure, Karlsruhe, Germany
[6] School of Computer Science and Engineering, Southeast University, Nanjing, China

Abstract. The LOD cloud offers a plethora of RDF data sources where users discover items of interest by issuing SPARQL queries. A common query problem for users is to face with empty answers: given a SPARQL query that returns nothing, how to refine the query to obtain a non-empty set? In this paper, we propose an RDF graph embedding based framework to solve the SPARQL empty-answer problem in terms of a continuous vector space. We first project the RDF graph into a continuous vector space by an entity context preserving translational embedding model which is specially designed for SPARQL queries. Then, given a SPARQL query that returns an empty set, we partition it into several parts and compute approximate answers by leveraging RDF embeddings and the translation mechanism. We also generate alternative queries for returned answers, which helps users recognize their expectations and refine the original query finally. To validate the effectiveness and efficiency of our framework, we conduct extensive experiments on the real-world RDF dataset. The results show that our framework can significantly improve the quality of approximate answers and speed up the generation of alternative queries.

Keywords: SPARQL · Empty-answer · RDF · Graph embedding

1 Introduction

With the rapid development of Semantic Web technologies, various knowledge bases are published on the Linked Open Data (LOD) cloud using Resource

D. Vrandečić et al. (Eds.): ISWC 2018, LNCS 11136, pp. 513–529, 2018.
https://doi.org/10.1007/978-3-030-00671-6_30

Description Framework (RDF). To enable users to retrieve the desired data, many RDF datasets provide SPARQL endpoints that allow users to issue basic graph pattern (BGP) queries [11]. However, issuing SPARQL queries requires users to be precisely aware of the structure and schema of the RDF dataset, and this is challenging. Therefore, it is a common scenario for users where an inappropriate query returns an empty set, the so-called empty-answer problem.

Most existing work solves this problem through query relaxation approaches [6–9,12–14] which focus on relaxing RDF terms specified in the original query so that the new relaxed query returns sufficient answers. They find top-k optimal relaxed queries in the exponential search space then evaluate the matching process between the relaxed queries and the RDF graph, which is really time-consuming. **Is it possible to directly retrieve approximate answers for a failing query without changing any parts of the original query? The answer is yes.** In this paper, we stand on recent advances in RDF embedding techniques and address the SPARQL empty-answer problem from the angle of continuous vector space.

Motivating Example: A user wants to find drama films which were released in the United States and directed by Tim Burton. After issuing a SPARQL query over DBpedia [16], the user obtains an empty answer, as shown on the left of Fig. 1. In this example, *Tim Burton*, *director*, *country*, *United States*, *type*, and *drama films* are specified RDF terms in the SPARQL BGP [11]. Since each specified term must be matched, this is too restrictive for the graph pattern matching. To deal with such failing queries, users usually have no idea which parts of the query should be responsible for the missing possible answers.

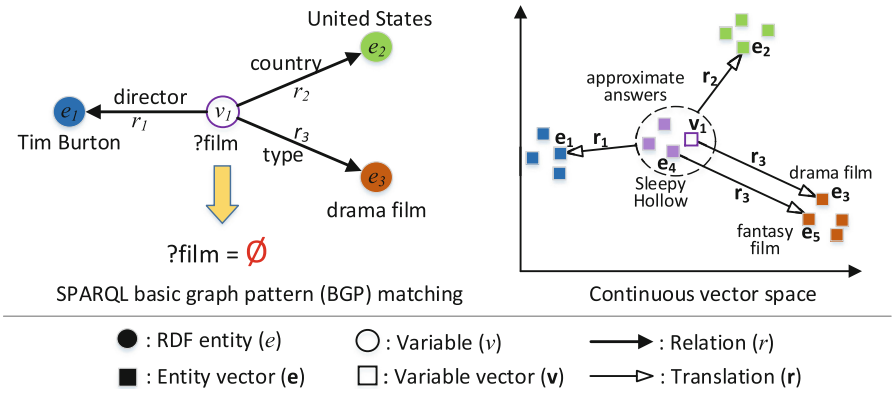

Fig. 1. Failing SPARQL BGP and RDF graph embeddings.

The right of Fig. 1 illustrates the ideal vector representation (i.e., the embedding) of the RDF graph to be queried in a continuous vector space, where entities are represented by vectors (boldface letters), and semantically similar entities are close to each other. For example, $\mathbf{e_3}$ and $\mathbf{e_5}$ are close since *drama film*(e_3) and

fantasy film(e_5) are similar. The relation between two entities is represented as a translation operation from the head entity to the tail entity, e.g., $\mathbf{e_4} + \mathbf{r_3} \approx \mathbf{e_5}$ when a triple $\langle$*Sleepy Hollow*(e_4), *type*(r_3), *fantasy film*(e_5)$\rangle$ holds. With the translation mechanism, although the expected answer v_1 does not exist in the RDF graph, we can still compute its vector representation $\mathbf{v_1}$ based on specified terms in the SPARQL, e.g., $\mathbf{v_1} \approx \mathbf{e_1} - \mathbf{r_1}$ according to $\langle v_1, r_1, e_1 \rangle$. Then, we can obtain $\mathbf{e_4}$ which is close to $\mathbf{v_1}$ in the space. *Sleepy Hollow*(e_4) is likely to meet the query intention of the original SPARQL query in the RDF graph.

Challenges: Leveraging RDF embeddings is a promising pathway to directly find approximate answers for a failing SPARQL query, but it is also troubling. Our method confronts the following challenges:

- *Limitations of existing embedding models*: We need to project the RDF graph into a continuous vector space, where semantically similar entities are close to each other and the relations among entities are represented by translations. However, none of the existing embedding models (e.g., RDF2vec [20], TransE [3], et al.) meet the requirements.
- *Variety of BGPs*: A BGP may contain multiple different variables and usually contains several triple patterns sharing the same variables or entities. Therefore, how to exploit the interplay between triple patterns and compute approximate answers for each variable is challenging and non-trivial.
- *Comprehensibility of approximate answers*: Obtaining approximate answers without any explanations is inadequate for satisfying users because they may ask why an approximate answer is returned.

Solutions: Given these challenges, we propose a novel framework to address the SPARQL empty-answer problem. The procedure includes the following:

- Firstly, the RDF graph is projected into a continuous vector space by an entity context preserving translational embedding model which is specially designed for SPARQL BGPs.
- Then, given a SPARQL query that returns an empty answer, the SPARQL BGP is partitioned into several parts based on different variables. By leveraging the RDF embeddings and the translation mechanism, approximate answers are further computed based on the vector representations of variables and specified query terms.
- Finally, approximate answers are returned to users. Each returned answer will be attached with an alternative query, which helps users recognize their expectations and refine the original query.

Contributions: Our framework makes the following contributions:

- To the best of our knowledge, we are the first to solve the SPARQL empty-answer problem from a continuous vector space perspective, which improves the quality of the returned answers and speeds up the generation of alternative queries.

- We propose a novel RDF graph embedding model which utilizes the translation mechanism to capture the relations between entities while considering the entity context to make the representations of semantically similar entities close to each other in the vector space.
- We propose efficient algorithms to compute approximate answers attached with alternative queries as the explanations for users.
- We conduct extensive experiments on the real-world dataset to evaluate the effectiveness and efficiency of the framework. These results provide supporting evidence that the framework is powerful in generating effective answers and explanations for failing queries within an acceptable time.

Organization: The remainder of the paper is organized as follows. Section 2 presents the details of the proposed framework. The evaluation of the framework is reported in Sect. 3. The related work is discussed in Sect. 4. Finally, our conclusions and future work are presented in Sect. 5.

2 The Proposed Framework

Before demonstrating the details of our framework, we briefly introduce the notations employed in this paper.

RDF Graph. Let $\mathcal{E}$ be a set of entities, $\mathcal{R}$ be a set of relations between entities. An RDF graph $\mathcal{G} = (\mathcal{E}, \mathcal{R})$ is a finite set of RDF triples in the form $\langle e_h, r, e_t \rangle$, where $e_h, e_t \in \mathcal{E}$ and $r \in \mathcal{R}$. An RDF triple $\langle e_h, r, e_t \rangle$ indicates a directed relation r from the head entity e_h to the tail entity e_t, e.g., $\langle$*Batman*, *director*, *Tim Burton*$\rangle$.

The standard SPARQL [11] contains BGPs and other operations (*UNION*, *OPTIONAL*, *FILTER*, etc.). In this paper, we focus on the SPARQL empty-answer problem caused by over-constrained BGPs, which already yields a non-trivial problem to study.

BGP. Let $\mathcal{V}$ be a set of entity variables. Each variable $v \in \mathcal{V}$ is distinguished by a leading question mark symbol, e.g., *?film*. A triple pattern is similar to an RDF triple but allows the usage of variables for entities[1], e.g., $\langle$*?film*, *director*, *Tim Burton*$\rangle$. A SPARQL BGP $\mathcal{P} = (\mathcal{E}_\mathcal{P} \cup \mathcal{V}_\mathcal{P}, \mathcal{R}_\mathcal{P})$ is a finite set of triple patterns, where $\mathcal{E}_\mathcal{P} \subseteq \mathcal{E}$, $\mathcal{V}_\mathcal{P} \subseteq \mathcal{V}$ and $\mathcal{R}_\mathcal{P} \subseteq \mathcal{R}$.

SPARQL Query. The official standard [11] defines four different forms of queries on the top of BGPs, namely *SELECT*, *ASK*, *CONSTRUCT*, and *DESCRIBE*. Since *SELECT* is the only form which returns the graph matching results to users, we define a SPARQL query Q as an expression of the form *SELECT* $\mathcal{S}$ *FROM* $\mathcal{G}$ *WHERE* $\mathcal{P}$, where $\mathcal{P}$ is a BGP, $\mathcal{G}$ is an RDF graph to be queried, and $\mathcal{S} \subseteq \mathcal{V}_\mathcal{P}$.

[1] To simplify the problem, we do not consider variables on predicates in this paper as such graph patterns are mainly used for exploring RDF schema but rarely used in real-world SPARQL queries [2].

SPARQL Empty-Answer Problem. Given a SPARQL query *SELECT* $\mathcal{S}$ *FROM* $\mathcal{G}$ *WHERE* $\mathcal{P}$, $\mathcal{P}$ is evaluated to match $\mathcal{G}$, and the variables in $\mathcal{P}$ are substituted by entities in $\mathcal{G}$. *SELECT* $\mathcal{S}$ employs the matched RDF graphs to provide the final result set $\mathcal{RS}$. The SPARQL empty-answer problem refers to the scenario where the final result set is empty, i.e., $\mathcal{RS} = \varnothing$.

Given a failing SPARQL query, our goal is to automatically generate top-k answers which approximately meet the original query intention along with corresponding alternative queries. Different from the existing methods [6–9,12–14], our framework solves the empty-answer problem based on a continuous vector space, as introduced in the example of Sect. 1. The proposed framework mainly includes three modules: learning RDF embeddings (described in Sect. 2.1), computing variable embeddings (described in Sect. 2.2), as well as generating approximate answers and alternative queries (described in Sect. 2.3).

2.1 Learning RDF Embeddings

In this module, we aim to embed entities and relations of the underlying RDF graph into a continuous vector space while preserving the inherent structure of the graph. Neural-language-based models, e.g., RDF2vec [20], only generate entity latent representations, and they cannot encode relations between entities. Therefore, we adopt the translation mechanism of TransE [3] to capture the correlations between entities and relations[2]. The translation mechanism in this context represents a relation as a translation operation from the head entity to the tail entity in the continuous vector space. Specifically, if an RDF triple $\langle e_h, r, e_t\rangle \in \mathcal{G}$, our objective is to learn embeddings $\mathbf{e_h}$, $\mathbf{r}$ and $\mathbf{e_t}$ which hold $\mathbf{e_h} + \mathbf{r} \approx \mathbf{e_t}$ ($\mathbf{e_t}$ should be a nearest neighbor of $\mathbf{e_h} + \mathbf{r}$). However, directly adopting TransE does not guarantee that semantically similar entities are close to each other in the continuous vector space since it regards an RDF graph as a set of independent triples during the learning process. Triples in the RDF graph are not independent, and semantically similar entities tend to share common context information, e.g., neighboring entities and their associated relations. Therefore, we propose a novel embedding method which considers the entity context information during the translation-based learning process.

Definition 1 (Entity Context). *For an entity* $e \in \mathcal{E}$*, the context of* e *is a set* $C(e) = \{(r_c, e_c) | r_c \in \mathcal{R}, e_c \in \mathcal{E}, \langle e, r_c, e_c\rangle \in \mathcal{G} \vee \langle e_c, r_c, e\rangle \in \mathcal{G}\}$*, where* r_c *is the relation between* e *and its neighbor* e_c*.*

Given an entity $e \in \mathcal{E}$, our goal is to learn the vector representation $\mathbf{e}$ while preserving its entity context information. To this end, we first define the conditional probability of e given its context $C(e)$ as follows:

$$P(e|C(e)) = \frac{\exp(f_1(e, C(e)))}{\sum_{e' \in \mathcal{E}} \exp(f_1(e', C(e)))}, \tag{1}$$

[2] Other translation-based embedding models, such as TransH [21] and TransR [17], can also be easily adopted for the RDF triple encoding.

where $f_1(e', C(e))$ is a score function that measures the correlation between an arbitrary entity e' and the entity context of e. We define $f_1(e', C(e))$ as:

$$f_1(e', C(e)) = -\frac{1}{|C(e)|} \sum_{(r_c, e_c) \in C(e)} f_2(e', r_c, e_c), \tag{2}$$

where $f_2(e', r_c, e_c)$ is the score function of TransE that measures the correlation between e' and $(r_c, e_c) \in C(e)$. $f_2(e', r_c, e_c)$ is formulated as follows:

$$f_2(e', r_c, e_c) = \begin{cases} \|\mathbf{e'} + \mathbf{r_c} - \mathbf{e_c}\|_2^2, & \text{if } \langle e, r_c, e_c \rangle \in \mathcal{G}, \\ \|\mathbf{e_c} + \mathbf{r_c} - \mathbf{e'}\|_2^2, & \text{if } \langle e_c, r_c, e \rangle \in \mathcal{G}, \end{cases} \tag{3}$$

where $\langle e, r_c, e_c \rangle$ and $\langle e_c, r_c, e \rangle$ indicate the directions of r_c. Intuitively, if two entities share more common context information, their embeddings tend to be more similar according to the above equations. By maximizing the joint probability of all entities in $\mathcal{G}$, we define the final objective function as:

$$O = \sum_{e \in \mathcal{E}} \log P(e|C(e)). \tag{4}$$

Considering the over-sized RDF graph, it is impractical to directly compute Eq. (1). Hence, we follow [18] to approximate Eq. (1) based on negative sampling, as formulated in Eq. (5).

$$P(e|C(e)) \approx \sigma(f_1(e, C(e))) \cdot \prod_{e' \in \mathcal{E}^e_{\mathcal{N}}}^{n} \sigma(f_1(e', C(e))), \tag{5}$$

where n is the number of negative examples, $\sigma(\cdot)$ is the sigmoid function, and e' is the negative entity which is obtained by sampling entities from a uniform distribution over the negative entity set $\mathcal{E}^e_{\mathcal{N}}$. For each negative entity $e' \in \mathcal{E}^e_{\mathcal{N}}$, the precondition is $C(e') \bigcap C(e) = \varnothing$.

Based on the above process, entities are encoded into the continuous vector space with their context information such that semantically similar entities are close to each other. The relations between entities are simultaneously captured by the translation mechanism. It is worth mentioning that the generation of embeddings is independent of the rest phases of the framework. Once the RDF embeddings have been learned, we can use them in SPARQL empty-answer problems solving without frequent modification.

2.2 Computing Variable Embeddings

We assume that the initial SPARQL is free of spelling/syntactic errors. In this module, we aim to compute embeddings of variables in the original SPARQL query. Similar to the correlation between an entity and its context, a variable is determined by its neighbors in the BGP of the initial query.

The neighbor of a variable could be a specified entity or any variable else (a BGP may contain multiple variables). Given a failing SPARQL query, we first partition its BGP into several sub-basic graph patterns (sBGPs), each of which contains only one variable connected with a set of specified entities.

Definition 2 (Sub-Basic Graph Pattern). *Given a basic graph pattern* $\mathcal{P} = (\mathcal{E}_\mathcal{P} \cup \mathcal{V}_\mathcal{P}, \mathcal{R}_\mathcal{P})$*, a sub-basic graph pattern (sBGP) for a variable* $v_s \in \mathcal{V}_\mathcal{P}$ *is a set* $\mathcal{SP} = \{\langle v_s, r_s, e_s\rangle | r_s \in \mathcal{R}_\mathcal{P}, e_s \in \mathcal{E}_\mathcal{P}, \langle v_s, r_s, e_s\rangle \in \mathcal{P}\} \cup \{\langle e_s, r_s, v_s\rangle | r_s \in \mathcal{R}_\mathcal{P}, e_s \in \mathcal{E}_\mathcal{P}, \langle e_s, r_s, v_s\rangle \in \mathcal{P}\}$*, where* r_s *is a relation between* v_s *and its neighbor* e_s*.*

For instance, the left of Fig. 2 illustrates a SPARQL BGP that retrieves American drama films directed by Tim Burton and there is a star actor who was born in New York. We can partition the BGP into two sBGPs, i.e., $sBGP_1$ and $sBGP_2$ for *?film* and *?actor*, respectively.

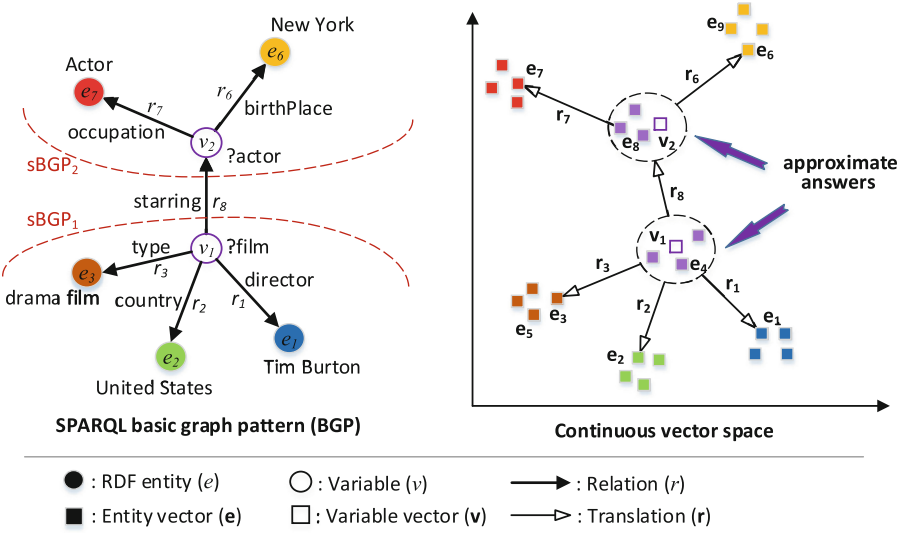

Fig. 2. Failing SPARQL basic graph pattern and RDF graph embeddings.

Then, given a variable v_s and the corresponding sBGP $\mathcal{SP}$, we utilize specified entities in $\mathcal{SP}$ to estimate the embedding of v_s. For a single triple pattern $\langle v_s, r_s, e_s\rangle$ or $\langle e_s, r_s, v_s\rangle$ in $\mathcal{SP}$, we can obtain a preliminary embedding $\tilde{\mathbf{v}}_\mathbf{s}$, computed as follows:

$$\tilde{\mathbf{v}}_\mathbf{s} = \begin{cases} \mathbf{e_s} - \mathbf{r_s}, \text{ if } \langle v_s, r_s, e_s\rangle \in \mathcal{SP}, \\ \mathbf{e_s} + \mathbf{r_s}, \text{ if } \langle e_s, r_s, v_s\rangle \in \mathcal{SP}. \end{cases} \tag{6}$$

For instance, regarding the variable *?film* (v_1) shown in Fig. 2, we can utilize the triple pattern $\langle$*?film, director, Tim Burton*$\rangle$, i.e., $\langle v_1, r_1, e_1\rangle$, to obtain the preliminary embedding of v_1 according to Eq. (6), i.e., $\tilde{\mathbf{v}}_\mathbf{1} = \mathbf{e_1} - \mathbf{r_1}$ in the continuous vector space.

In the sBGP $\mathcal{SP}$, if the variable v_s is involved in multiple triple patterns, we need to jointly consider these triple patterns in the estimation of the variable embedding. Intuitively, different triple patterns may have different impacts on determining the variable embedding. For example, $sBGP_1$ in Fig. 2 consists of

three triple patterns, i.e., ⟨*?film, director, Tim Burton*⟩, ⟨*?film, country, United States*⟩, and ⟨*?film, type, drama film*⟩. In the underlying RDF graph, e.g., DBpedia, there are respectively 24, 112,336, and 238 RDF triples matching the three triple patterns. Therefore, the triple pattern ⟨*?film, director, Tim Burton*⟩ contains more specific information for estimating the variable embedding of *?film* compared with the other two triple patterns, and it should attract more attention. We define the following attention function $a(v_s, r_s, e_s)$ to measure the attention on a triple pattern $\langle v_s, r_s, e_s \rangle$ when estimating the embedding of v_s.

$$a(v_s, r_s, e_s) = \exp\left(-\frac{|\{e_{v_s} | \langle e_{v_s}, r_s, e_s \rangle \in \mathcal{G} \vee \langle e_s, r_s, e_{v_s} \rangle \in \mathcal{G}\}|}{\sum_{\langle v_s, r'_s, e'_s \rangle \in \mathcal{SP} \vee \langle e'_s, r'_s, v_s \rangle \in \mathcal{SP}} |\{e_{v_s} | \langle e_{v_s}, r'_s, e'_s \rangle \in \mathcal{G} \vee \langle e'_s, r'_s, e_{v_s} \rangle \in \mathcal{G}\}|}\right), \tag{7}$$

where the numerator of the exponent is the number of RDF triples in the underlying RDF graph matching the triple pattern $\langle v_s, r_s, e_s \rangle$. The denominator of the exponent is the number of RDF triples in the underlying RDF graph matching any triple pattern in $\mathcal{SP}$. For instance, according to Eq. (7), the attention scores in $sBGP_1$ are 0.9998, 0.3687, and 0.9979 for the three triple patterns $\langle v_1, r_1, e_1 \rangle$, $\langle v_1, r_2, e_2 \rangle$, and $\langle v_1, r_3, e_3 \rangle$, respectively. And the attention scores in $sBGP_2$ are 0.5967 and 0.6165 for $\langle v_2, r_6, e_6 \rangle$, $\langle v_2, r_7, e_7 \rangle$, respectively.

By examining all neighbors of v_s in a sBGP $\mathcal{SP}$, we further define the preliminary embedding $\hat{\mathbf{v}}_\mathbf{s}$ of v_s as:

$$\hat{\mathbf{v}}_\mathbf{s} = \frac{\sum_{\langle v_s, r_s, e_s \rangle \in \mathcal{SP} \vee \langle e_s, r_s, v_s \rangle \in \mathcal{SP}} a(v_s, r_s, e_s) \cdot \tilde{\mathbf{v}}_\mathbf{s}}{\sum_{\langle v_s, r_s, e_s \rangle \in \mathcal{SP} \vee \langle e_s, r_s, v_s \rangle \in \mathcal{SP}} a(v_s, r_s, e_s)}, \tag{8}$$

where $\tilde{\mathbf{v}}_\mathbf{s}$ is computed for each single triple pattern according to Eq. (6).

For instance, the preliminary embeddings of variable v_1 and v_2 can be computed as $\hat{\mathbf{v}}_\mathbf{1} = 0.4225 \cdot (\mathbf{e_1} - \mathbf{r_1}) + 0.1558 \cdot (\mathbf{e_2} - \mathbf{r_2}) + 0.4217 \cdot (\mathbf{e_3} - \mathbf{r_3})$ and $\hat{\mathbf{v}}_\mathbf{2} = 0.4918 \cdot (\mathbf{e_6} - \mathbf{r_6}) + 0.5082 \cdot (\mathbf{e_7} - \mathbf{r_7})$, respectively.

For the impacts of relations among variable entities in different sBGPs, we introduce a simple and effective method to compute the final variable embeddings based on the preliminary embeddings. For a variable $v_s \in \mathcal{V}_\mathcal{P}$ which is directly linked with other variables in BGP $\mathcal{P} = \{\mathcal{E}_\mathcal{P} \cup \mathcal{V}_\mathcal{P}, \mathcal{R}_\mathcal{P}\}$, we compute its final embedding $\mathbf{v_s}$ as follows:

$$\mathbf{v_s} = \frac{\sum_{\langle v_s, r, v'_s \rangle \in \mathcal{P} \vee \langle v'_s, r, v_s \rangle \in \mathcal{P}} num(v'_s) \cdot f_3(v_s, r, v'_s) + num(v_s) \cdot \hat{\mathbf{v}}_\mathbf{s}}{\sum_{\langle v_s, r, v'_s \rangle \in \mathcal{P} \vee \langle v'_s, r, v_s \rangle \in \mathcal{P}} num(v'_s) + num(v_s)}, \tag{9}$$

where

$$f_3(v_s, r, v'_s) = \begin{cases} \hat{\mathbf{v}}'_\mathbf{s} - \mathbf{r}, \text{ if } \langle v_s, r, v'_s \rangle \in \mathcal{P}, \\ \hat{\mathbf{v}}'_\mathbf{s} + \mathbf{r}, \text{ if } \langle v'_s, r, v_s \rangle \in \mathcal{P}, \end{cases} \tag{10}$$

and $num(\cdot)$ is the number of triple patterns in the sBGP of a variable. The reason we utilize $num(\cdot)$ is that we assume the preliminary embedding of a variable deserves more attention if it is computed based on more triple patterns. It is worth mentioning that the correlations between variables can be characterized

by more sophisticated method, such as an iterative updating algorithm. We will investigate this part in the future.

Finally, we can obtain all variable embeddings of the original query in the continuous vector space. For instance, the final embeddings of variable v_1 and v_2 in Fig. 2 can be computed as $\mathbf{v_1} = 0.4 \cdot (\hat{\mathbf{v_2}} - \mathbf{r_8}) + 0.6 \cdot \hat{\mathbf{v_1}} = 0.4 \cdot [0.4918 \cdot (\mathbf{e_6} - \mathbf{r_6}) + 0.5082 \cdot (\mathbf{e_7} - \mathbf{r_7}) - \mathbf{r_8}] + 0.6 \cdot [0.4225 \cdot (\mathbf{e_1} - \mathbf{r_1}) + 0.1558 \cdot (\mathbf{e_2} - \mathbf{r_2}) + 0.4217 \cdot (\mathbf{e_3} - \mathbf{r_3})]$ and $\mathbf{v_2} = 0.6 \cdot (\hat{\mathbf{v_1}} + \mathbf{r_8}) + 0.4 \cdot \hat{\mathbf{v_2}} = 0.6 \cdot [0.4225 \cdot (\mathbf{e_1} - \mathbf{r_1}) + 0.1558 \cdot (\mathbf{e_2} - \mathbf{r_2}) + 0.4217 \cdot (\mathbf{e_3} - \mathbf{r_3}) + \mathbf{r_8}] + 0.4 \cdot [0.4918 \cdot (\mathbf{e_6} - \mathbf{r_6}) + 0.5082 \cdot (\mathbf{e_7} - \mathbf{r_7})]$, respectively.

2.3 Generating Approximate Answers and Alternative Queries

In this module, our goal is to discover approximate answers based on embeddings of variables in the continuous vector space. For each approximate answer, we also generate an alternative query as the explanation to help the user recognize his expected information and refine the original query.

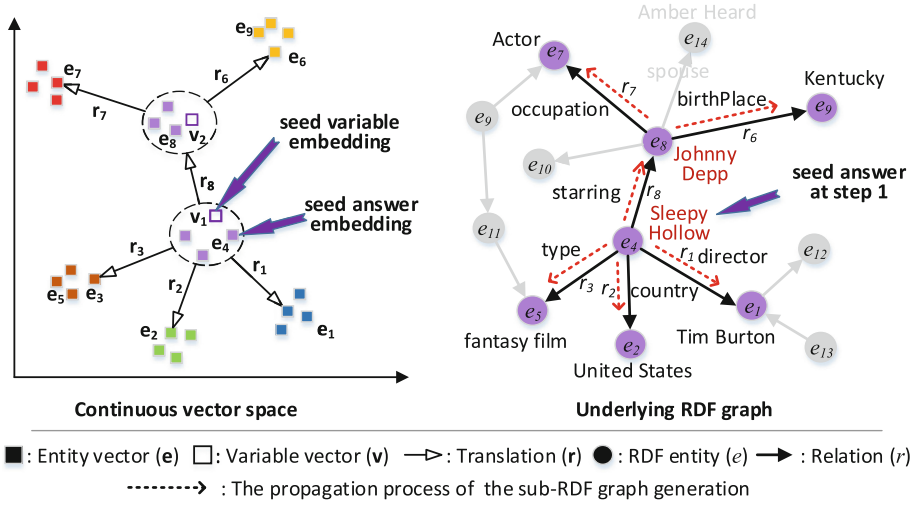

Fig. 3. Approximate answers generation based on the RDF embeddings.

As analyzed in Sect. 2.1, semantically similar entities are close to each other in the continuous vector space. Therefore, given a BGP $\mathcal{P} = (\mathcal{E}_\mathcal{P} \cup \mathcal{V}_\mathcal{P}, \mathcal{R}_\mathcal{P})$, a variable $v_p \in \mathcal{V}_\mathcal{P}$, and the embedding $\mathbf{v_p}$ computed through Eq. (9), we can readily find a semantically similar entity e_i of the RDF graph $\mathcal{G} = (\mathcal{E}, \mathcal{R})$ for v_p by computing the distance between $\mathbf{v_p}$ and $\mathbf{e_i}$ in the continuous vector space. We employ the cosine similarity to measure the distance between $\mathbf{v_p}$ and $\mathbf{e_i}$ as follows:

$$sim(\mathbf{v_p}, \mathbf{e_i}) = \frac{\mathbf{v_p} \cdot \mathbf{e_i}}{\|\mathbf{v_p}\| \|\mathbf{e_i}\|}. \tag{11}$$

According to Eq. (11), we can obtain top-k semantically similar entities $\mathcal{E}_{\mathcal{K}} = \{e_1, ..., e_i, ..., e_k\}$ for v_p.

In a simple case where a failing BGP $\mathcal{P} = (\mathcal{E}_{\mathcal{P}} \cup \{v_p\}, \mathcal{R}_{\mathcal{P}})$ contains only one variable v_p, the semantically similar entity set $\mathcal{E}_{\mathcal{K}}$ implied by Eq. (11) is exactly the set of final approximate answers to v_p. For each approximate answer $e_i \in \mathcal{E}_{\mathcal{K}}$, we can directly extract a sub-RDF graph $\mathcal{SG}_i$ about e_i from $\mathcal{G}$. The extraction of $\mathcal{SG}_i$ is a searching process based on e_i. Specifically, for each triple pattern $\langle v_p, r_p, e_p \rangle \in \mathcal{P}$ (v_p at the head), its ideal corresponding RDF triple is $\langle e_i, r_p, e_p \rangle$. If $\langle e_i, r_p, e_p \rangle \notin \mathcal{G}$, we figure out the corresponding RDF triple $\langle e_i, r, e' \rangle$ which is most similar to $\langle e_i, r_p, e_p \rangle$ among all the RDF triples in $\mathcal{G}$, formulated as follows:

$$\langle e_i, r, e' \rangle = \underset{\langle e_i, r, e' \rangle \in \mathcal{G}}{\arg\max} \left(\frac{\mathbf{r_p} \cdot \mathbf{r}}{\|\mathbf{r_p}\| \|\mathbf{r}\|} + \frac{\mathbf{e_p} \cdot \mathbf{e'}}{\|\mathbf{e_p}\| \|\mathbf{e'}\|} \right). \tag{12}$$

Analogically, for each triple pattern $\langle e_p, r_p, v_p \rangle \in \mathcal{P}$ (v_p at the tail), we can also compute its corresponding RDF triple. These RDF triples containing e_i form the sub-RDF graph $\mathcal{SG}_i$ which can be utilized to generate a modified BGP $\mathcal{P}'$ that expresses the similar query intention to $\mathcal{P}$. For example, assuming that a BGP $\{\langle ?film, type, drama\ film \rangle\}$ returns nothing over an RDF graph, we may obtain an approximate answer *Sleepy Hollow*. Then we can extract a sub-RDF graph $\{\langle Sleepy\ Hollow, type, \textbf{\textit{fantasy film}} \rangle\}$ and return *Sleepy Hollow* along with $\{\langle ?film, type, \textbf{\textit{fantasy film}} \rangle\}$ for the user.

For a BGP with multiple variables, we select a seed variable and obtain its approximate answers as seed answers according to Eq. (12). Specifically, given a SPARQL query *SELECT* $\mathcal{S}$ *FROM* $\mathcal{G}$ *WHERE* $\mathcal{P}$, we select the seed variable from $\mathcal{S}$ (i.e., the expected result expressed by users). If $\mathcal{S}$ contains multiple variables, the seed variable is selected from $\mathcal{S}$ based on the degrees of variables in the BGP $\mathcal{P}$, since a variable at a larger degree usually indicates that the user is more interested in it. For each seed answer, we adopt a propagation process to generate the final returned answer and the alternative query. For example, the variable v_1 at the largest degree in Fig. 3 will be selected as the seed variable. We first find its approximate answer *Sleep Hollow* (e_4) as the seed answer in the continuous vector space. In the first step of the propagation process, we follow Eq. (12) to find the corresponding triples $\{\langle Sleep\ Hollow, type, fantasy\ film \rangle, \langle Sleep\ Hollow, country, United\ States \rangle, \langle Sleep\ Hollow, director, Tim\ Burton \rangle, \langle Sleep\ Hollow, starring, Johnny\ Depp \rangle\}$. After this step, we have determined v_2 as *Johnny Depp*. Then, in the second step, we remove the triple patterns which have already been determined and set *Johnny Depp* (e_8) as a new seed answer for the next step. Repeat the propagation operation until all triple patterns have been determined as illustrated in the right of Fig. 3. Finally, we can generate the final returned answer $\{?film{:}Sleep\ Hollow, ?actor{:}Johnny\ Depp\}$ and the alternative BGP $\{\langle ?film, type, \textbf{\textit{fantasy film}} \rangle, \langle ?film, country, United\ States \rangle, \langle ?film, director, Tim\ Burton \rangle, \langle ?film, starring, ?actor \rangle, \langle ?actor, occupation, Actor \rangle, \langle ?actor, birthplace, \textbf{\textit{Kentucky}} \rangle\}$.

Discussions: We discuss two parts which can be improved during the framework implementation: (1) Given a variable embedding in the vector space, there

is no need to traverse all entity embeddings for the top-k similar entities searching. For example, we can partition the vector space into disjoint subspaces, and differentiate entities of the RDF graph according to the subspaces to which they belong. More sophisticated approaches are provided in [15] for this issue. (2) Currently, we assume that a variable at a larger degree is more important in the sub-RDF graph extraction. The preliminary experiments show some effectiveness, but we still need to improve the scalability in the future. For instance, users can be allowed to determine which variable is most important.

3 Experimental Evaluation

To scrutinize the effectiveness and efficiency of the proposed framework, we performed three types of experiments including: (1) Entity context preserving embedding validation; (2) Quality evaluation of approximate answers and alternative queries; (3) Efficiency evaluation. The results demonstrate that our framework significantly outperforms other baselines.

Dataset: DBpedia [16] is extracted from Wikipedia[3] and has become the core dataset of the LOD. In this paper, we employed the English version of DBpedia[4], which consists of 6.7M entities, 1.4K relations and 583M RDF triples.

Queries: According to our investigation, there is no open domain benchmark query set for SPARQL empty-answer problem. Therefore, twenty queries were constructed over DBpedia for the evaluation. The queries were designed to include basic graph patterns with different topological structures (e.g., star, chain, cycle, and complex) based on joins over variables [10].

Baselines: To validate the effectiveness of the consideration of entity context information in the translation-based model, we compared our embedding model with TransE [3]. To evaluate the empty-answer problem solving, we compared our framework with four state-of-the-art query relaxation models, i.e, similarity-based (SB) [7], rule-based (RB) [14], user-preferences-based (UPB) [5], and cooperative-techniques-based (CTB) [9] models. Meanwhile, we also compared our framework to a lite version with directly TransE plugged in.

3.1 Entity Context Preserving Embedding Validation

Our translation-based embedding model leverages the entity context information to encode semantically similar entities and utilizes the translation mechanism to represent relations between entities. The embedding model was implemented in Java, and the following validation was conducted on a Linux server with an Intel Core i7 3.40 GHz CPU and 126 GB memory running Ubuntu-14.04.1. We determined the optimal parameters using a grid search strategy. And the training instances were conducted over 1,000 iterations. The running time per iteration was 562 s. We report the implementation code and detailed parameters in https://github.com/wangmengsd/re.

[3] https://www.wikipedia.org.
[4] http://wiki.dbpedia.org/develop/datasets.

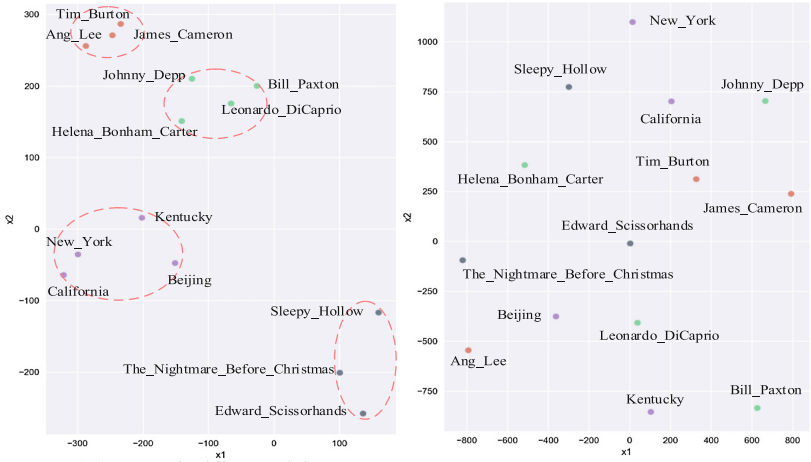

Fig. 4. Visualization of entity embeddings learned by our model and TransE.

For the entity context preserving validation, we projected several sample entity embeddings generated by our embedding method and TransE into two-dimensional spaces using t-SNE[5], as shown in Fig. 4. The result in Fig. 4(a) is consistent with our expectation, where semantically similar entities are close to each other. In contrast, the distribution of entities in Fig. 4(b) does not represent the result we want.

For the translation preserving validation, we followed TransE in [3] and employed *MeanRank* and *Hits@10* as evaluation metrics. Specifically, to test a triple $\langle e_h, r, e_t \rangle$, we removed the head entity e_h or the tail entity e_t. Then we predicted the missing entity e_h or e_t based on $\mathbf{e_t} - \mathbf{r}$ or $\mathbf{e_h} + \mathbf{r}$, and we used the score function Eq. (3) to rank the predictions in a descending order. *MeanRank* denotes the average rank of all correct predictions, and *Hits@10* denotes the proportion of correct predictions ranked in the top-*10*. The *MeanRank* of our embedding model is 8.01 and *Hits@10* is 83.98. Whereas, the *MeanRank* of TransE is 8.00 and *Hits@10* is 84.01. Both the results of *MeanRank* and *Hits@10* proved that our embedding model maintained the effectiveness of the translation mechanism.

3.2 Quality of Approximate Answers and Alternative Queries

In this section, we compared our framework with four state-of-the-art query relaxation models [5,7,9,14]. Note that the efficient approach in [9] only computed Maximal Succeeding Subqueries (XSSs) (a kind of relaxed queries), and it didn't support similarity criteria to rank the multiple XSSs and query answers.

[5] https://lvdmaaten.github.io/tsne/.

To evaluate the quality of generated approximate answers and alternative queries, ten evaluators (graduate students in Computer Science, but none of them knows the details of the methods) were asked to evaluate how well an approximate answer satisfies the original query intention (*Sat*) and how similar an alternative query is to the original one (*Sim*). The two metrics *Sat* and *Sim* are rated on a 5-point scale: 0 corresponding to "negative", 1 to "weakly negative", 2 to "neutral", 3 to "weakly positive", and 4 to "positive". For each empty-answer query, we presented top-k[6] approximate answers and alternative queries generated by our framework and four baselines to the evaluators. We also employed *Pearson Correlation Coefficient* to analyze the correlation between the evaluator ratings and similarity scores calculated by the corresponding models. The *Pearson Correlation Coefficient* is a standard measure of the correlation between two variables. The coefficient value ranges from -1 to $+1$, where -1 represents totally negative correlation, 0 represents no linear correlation, and $+1$ represents totally positive correlation. Table 1 reports the average ratings (Avg.Rating) of all five models and the *Pearson Correlation Coefficients* (PCC). We can make the following observations:

Table 1. Results of overall effectiveness.

	Top-k	Top-1		Top-3		Top-5		Top-10		Top-20	
	Metric	*Sat*	*Sim*	*Sat*	*Sim*	*Sat*	*Sim*	*Sat*	*Sim*	*Sat*	*Sim*
Our method	Avg. rating	**3.5**	**3.25**	**3.15**	**3.03**	**2.72**	**2.61**	1.86	**1.82**	1.41	**1.40**
	PCC	0.54	0.51	0.52	0.48	0.53	0.48	0.52	0.49	0.53	0.49
Lite version with TransE	Avg. rating	0.85	0.33	0.45	0.28	0.3	0.2	0.24	0.09	0.13	0.06
	PCC	0.12	0.07	0.09	−0.03	0.06	−0.05	0.07	0.02	0.05	0.03
SB [7]	Avg. rating	3.2	3.0	3.02	2.75	2.45	2.05	**1.87**	1.77	**1.44**	1.39
	PCC	0.43	0.41	0.43	0.39	0.41	0.36	0.43	0.37	0.43	0.37
RB [14]	Avg. rating	3.05	3.0	2.93	2.68	2.5	2.21	**1.87**	1.74	1.42	1.37
	PCC	0.40	0.39	0.41	0.39	0.42	0.40	0.41	0.38	0.40	0.38
UPB [5]	Avg. rating	2.85	2.75	2.42	2.07	2.11	1.6	1.45	1.21	1.22	0.97
	PCC	0.33	0.29	0.31	0.27	0.32	0.26	0.33	0.27	0.34	0.28
CTB [9]	Avg. rating	2.7	2.75	2.45	2.05	2.16	1.62	1.43	1.18	1.24	0.97
	PCC	-	-	-	-	-	-	-	-	-	-

- Our framework achieved consistently significant improvement on *Sat* and *Sim* compared with all the baselines, which demonstrates the effectiveness of our framework. The reason is that we can directly compare the semantic similarity between expected answers and approximate answers in the continuous vector space. And the entity context preserving embedding model enables our method to generate high quality approximate answers and alternative queries.
- The PCC of our framework is also higher than all other baselines, which indicates that our similarity measuring mechanism (Eqs. (11) and (12)) is more

[6] We set $k \in \{1, 3, 5, 10, 20\}$ in this paper.

identical with the perception of users. The reason is that the embeddings are learned based on entity context information of the underlying RDF graph which contains more precise and richer information than the ontological information and statistical language models employed by other models.
– The performances of all methods were affected when increasing k to 20 because more irrelevant answers were generated. However, our method still has its own advantage. Since our method lists approximate answers in a descending order in terms of the similarity, users can obtain the most approximate answers at the top.

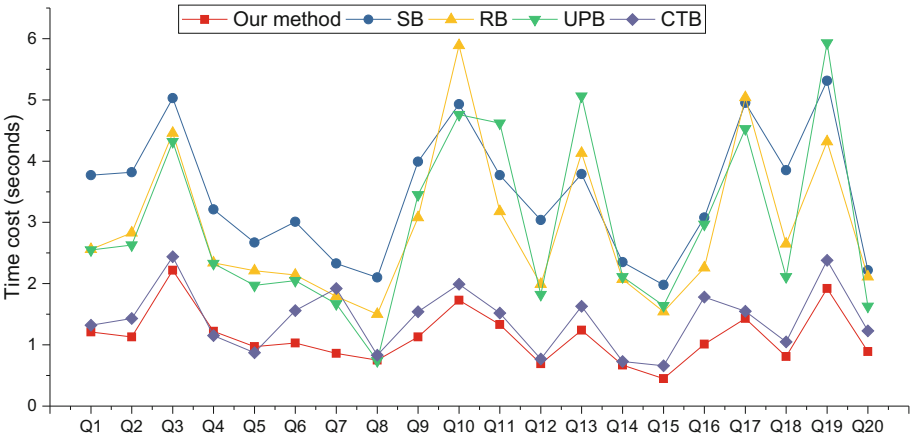

Fig. 5. Time costs of five methods on twenty failing SPARQL queries $Q1 \sim Q20$.

3.3 Efficiency Evaluation

The average time cost of our method to process a failing query is 1.13 s. This amount of time is acceptable for users to obtain approximate answers and alternative queries. We compared the time cost of our framework with other baselines. Figure 5 illustrates the runtime results of solving twenty empty-answer SPARQL queries. We can observe that the time cost of our framework is significantly less than other baselines. The key reason is that our framework is driven and guided by the approximate answer embeddings, which speeds up the generation of possible answers and alternative queries. Another reason is that the similarity computation in the continuous vector space is more efficient than the conventional graph-based computation method over a large RDF graph.

In summary, the evaluation results on effectiveness and efficiency show that our framework can facilitate to generate high-quality approximate answers and alternative queries.

4 Related Work

This section discusses existing related research in the following aspects: RDF query relaxation approaches and RDF graph embedding techniques.

Query relaxation approaches in the RDF context have been proposed to solve the SPARQL empty-answer problem. These methods mainly focus on reformulating the original query into a new relaxed query by removing or relaxing RDF conditions. Four types of models, similarity-based, rule-based, user-preferences-based, and cooperative-techniques-based models are utilized to generate multiple relaxed query candidates. Similarity-based models [6,7] leverage lexical analyses to determine appropriate relaxation candidates. Rule-based models [12–14,19] exploit RDF schema semantics and rewriting rules to perform relaxation. The user-preferences-based model [5] automatically relaxes over-constrained RDF queries based on domain knowledge and user preferences. Cooperative-techniques-based models [8,9] design pruning strategies to reduce the exponential search space of finding Top-k optimal relaxed queries. However, relaxed queries generated by query relaxation approaches may be rather different from initial queries of users because these models cannot consider the expected answers which do not occur in the query results. Over-relaxed queries and irrelevant answers are not effective for the expectation of users.

Existing RDF graphs already include thousands of relation types, millions of entities, and billions of RDF triples [1]. The RDF applications based on conventional graph-based algorithms are compromised by the data sparsity and computational inefficiency. To address these problems, RDF graph embedding techniques [3,4,17,20,21] have been proposed to embed both entities and relations into continuous vector spaces. Among these methods, neural-language-based models [4,20] only generate entity latent representations by training the neural language model of input RDF graphs. As a result, semantically similar entities are close to each other in continuous vector spaces. But we cannot infer relations between entities solely based on entity latent representations. Translation-based models [3,17,21] are effective in modeling relations between entities because of their translation mechanisms. But they do not guarantee that semantically similar entities are close to each other in continuous vector spaces since they regard the RDF graph as a set of independent triples during the learning processes. To sum up, none of the existing models meets the requirements for modeling SPARQL triple patterns in our framework.

5 Conclusions and Future Work

In this paper, we solve the SPARQL empty-answer problem in the continuous vector space. To make semantically similar entities close to each other in the vector space, we propose a novel embedding model which utilizes the translation mechanism to capture the relations between entities while considering the entity context. Then, given a failing SPARQL query, we partition the SPARQL BGP into several parts and compute approximate answers by leveraging RDF embeddings and the translation mechanism. We also generate alternative queries for

approximate answers, which helps users recognize their information needs and refine the original query. We conduct extensive experiments on the real-world RDF dataset to validate the effectiveness and the efficiency of our framework.

In future work, we intend to improve the accuracy of variable embedding computation through an iterative updating algorithm. Another development of our research is to address the SPARQL empty-answer problem on graph patterns which contain operators such as *UNION*, *OPTIONAL*, *MINUS* and so on.

Acknowledgment. This work was supported by National Key Research and Development Program of China (2018YFB1004500), National Natural Science Foundation of China (61532015, 61532004, 61672419, and 61672418), Innovative Research Group of the National Natural Science Foundation of China (61721002), Innovation Research Team of Ministry of Education (IRT_17R86), Project of China Knowledge Centre for Engineering Science and Technology, Science and Technology Planning Project of Guangdong Province (No. 2017A010101029), and Teaching Reform Project of XJTU (No. 17ZX044).

References

1. Bizer, C., Heath, T., Berners-Lee, T.: Linked data-the story so far. Int. J. Semant. Web Inf. Syst. **5**(3), 1–22 (2009)
2. Bonifati, A., Martens, W., Timm, T.: An analytical study of large SPARQL query logs. Proc. VLDB Endow. **11**(2), 149–161 (2017)
3. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: Advances in Neural Information Processing Systems, pp. 2787–2795 (2013)
4. Cochez, M., Ristoski, P., Ponzetto, S.P., Paulheim, H.: Global RDF vector space embeddings. In: d'Amato, C., et al. (eds.) ISWC 2017. LNCS, vol. 10587, pp. 190–207. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68288-4_12
5. Dolog, P., Stuckenschmidt, H., Wache, H., Diederich, J.: Relaxing RDF queries based on user and domain preferences. J. Intell. Inf. Syst. **33**(3), 239 (2009)
6. Elbassuoni, S., Ramanath, M., Schenkel, R., Sydow, M., Weikum, G.: Language-model-based ranking for queries on RDF-graphs. In: Proceedings of the 18th ACM conference on Information and Knowledge Management, pp. 977–986. ACM (2009)
7. Elbassuoni, S., Ramanath, M., Weikum, G.: Query relaxation for entity-relationship search. ESWC 2011. LNCS, vol. 6644, pp. 62–76. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21064-8_5
8. Fokou, G., Jean, S., Hadjali, A., Baron, M.: Cooperative techniques for SPARQL query relaxation in RDF databases. In: Gandon, F., Sabou, M., Sack, H., d'Amato, C., Cudré-Mauroux, P., Zimmermann, A. (eds.) ESWC 2015. LNCS, vol. 9088, pp. 237–252. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18818-8_15
9. Fokou, G., Jean, S., Hadjali, A., Baron, M.: Handling failing RDF queries: from diagnosis to relaxation. Knowl. Inf. Syst. **50**(1), 167–195 (2017)
10. Görlitz, O., Thimm, M., Staab, S.: SPLODGE: systematic generation of SPARQL benchmark queries for linked open data. In: Cudré-Mauroux, P., et al. (eds.) ISWC 2012. LNCS, vol. 7649, pp. 116–132. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35176-1_8
11. Harris, S., Seaborne, A., Prud'hommeaux, E.: Sparql 1.1 query language. W3C recommendation **21**(10) (2013)

12. Hogan, A., Mellotte, M., Powell, G., Stampouli, D.: Towards fuzzy query-relaxation for RDF. In: Simperl, E., Cimiano, P., Polleres, A., Corcho, O., Presutti, V. (eds.) ESWC 2012. LNCS, vol. 7295, pp. 687–702. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30284-8_53
13. Huang, H., Liu, C., Zhou, X.: Approximating query answering on RDF databases. World Wide Web **15**(1), 89–114 (2012)
14. Hurtado, C.A., Poulovassilis, A., Wood, P.T.: Query relaxation in RDF. In: Spaccapietra, S. (ed.) Journal on Data Semantics X. LNCS, vol. 4900, pp. 31–61. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77688-8_2
15. Katayama, N., Satoh, S.: The SR-tree: an index structure for high-dimensional nearest neighbor queries. ACM Sigmod Rec. **26**(2), 369–380 (1997)
16. Lehmann, J., et al.: Dbpedia-a large-scale, multilingual knowledge base extracted from Wikipedia. Semant. Web **6**(2), 167–195 (2015)
17. Lin, Y., Liu, Z., Sun, M., Liu, Y., Zhu, X.: Learning entity and relation embeddings for knowledge graph completion. In: AAAI, vol. 15, pp. 2181–2187 (2015)
18. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems, pp. 3111–3119 (2013)
19. Poulovassilis, A., Wood, P.T.: Combining approximation and relaxation in semantic web path queries. In: Patel-Schneider, P.F., et al. (eds.) ISWC 2010. LNCS, vol. 6496, pp. 631–646. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17746-0_40
20. Ristoski, P., Paulheim, H.: RDF2Vec: RDF graph embeddings for data mining. In: Groth, P., et al. (eds.) ISWC 2016. LNCS, vol. 9981, pp. 498–514. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46523-4_30
21. Wang, Z., Zhang, J., Feng, J., Chen, Z.: Knowledge graph embedding by translating on hyperplanes. In: AAAI, vol. 14, pp. 1112–1119 (2014)