Knowledge-Based Systems 213 (2021) 106655

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

Top-*k* star queries on knowledge graphs through semantic-aware bounding match scores

Yuxiang Wang [a], Xiaoliang Xu [a,*], Qifan Hong [a], Jiahui Jin [b], Tianxing Wu [b]

[a] School of Computer Science and Technology, Hangzhou Dianzi University, China
[b] School of Computer Science and Engineering, Southeast University, China

ARTICLE INFO

Article history:
Received 14 July 2020
Received in revised form 30 October 2020
Accepted 3 December 2020
Available online 18 December 2020

Keywords:
Top-*k* star query
Bounding match score
Semantic similarity

ABSTRACT

Large-scale knowledge graphs containing millions of entities are very common nowadays. Querying knowledge graphs is essential for a wide range of emerging applications, e.g., question answering and semantic search. A star query aims to identify an entity by giving a set of related entities, which is an important query type on knowledge graphs. Answering star queries can be modeled as a graph query problem. Given a query graph Q, the graph query finds subgraphs in a knowledge graph G that match Q. We face two challenges on graph query: (1) existing graph query methods usually find subgraphs that are structurally similar to Q, which cannot measure whether a subgraph match satisfies the semantics of Q (i.e., real query intention), leading to an effectiveness issue, and (2) querying a large-scale knowledge graph is usually time-consuming because of the large search space. In this paper, we propose a Top-*k* semantic-aware graph query method over knowledge graphs for star queries, which provides semantically similar matches for Q instead of structurally similar matches. The semantic similarity of a match to Q is measured by an online computed bounding match score. By using bounds, we can efficiently prune the unpromising matches with lower semantic similarities without evaluating all matches. Extensive experiments over three real-world knowledge graphs confirm the effectiveness and efficiency of our solution.

1. Introduction

With the vigorous development of the internet-driven applications, massive data have been produced, including web data, encyclopedia data, scientific data, etc. Distilling the meaningful information from these massive data has become an urgent demand [1]. Therefore, knowledge graphs (such as DBpedia [2], Yago [3], and Freebase [4]) have been constructed for managing these real-world data in recent years [5]. In a knowledge graph, each node and edge represent an entity with attributes and a relationship between two entities, respectively. For instance, Freebase contains more than 43.9 million entities, interconnected by 2.4 billion relations [6]. It is important to be able to query these knowledge graphs and retrieve valuable information, this is essential for a wide range of emerging applications, e.g., question answering and semantic search [7].

Star query is a common but important query type on knowledge graphs [8]. As observed in [9,10], most real-life queries on knowledge graphs are star-like, which aim to identify a target entity, given a set of specific entities and predicates. Graph query is a widely used method for answering star queries on knowledge graphs [10]. For example, consider that a user wants to find *all Spanish soccer players who play for a England soccer club*. One can come up with a reasonable graph representation of this query as a query graph Q that consists of nodes (i.e., entities) and edges (i.e., relations), and identify subgraph matches of Q in a knowledge graph G using graph query models [8,11–14]. Fig. 1(a) shows a query graph of the above query example, and this is a typical star query where v_1 (name: *England*, type: *Country*), v_4 (name: *Spain*, type: *Country*) are specific entities, and v_3 (type: *Person*) is the target entity of interest, while e_1 (predicate: *country*), e_2 (predicate: *team*), and e_3 (predicate: *nationality*) are specific predicates. Many efforts have been made to support graph query in knowledge graphs, which can be concluded as the following three categories.

Graph pattern matching. Graph pattern matching is typically defined in terms of subgraph isomorphism [11,12,15,16], which is NP-complete and often too restrictive to capture sensible matches [17]. These methods requires all the nodes and edges in the matches to be exactly the same as the ones in Q. As a result, we can get some answers such as *Dani_Ceballos* (① in Fig. 1(b)), whose nationality is *Spain* and plays for *Arsenal*. Unfortunately,

* Correspondence to: Room 401, Building 1, Hangzhou Dianzi University , Zhejiang, 310018, China.
E-mail addresses: lsswyx@hdu.edu.cn (Y. Wang), xxl@hdu.edu.cn (X. Xu), qfhong@hdu.edu.cn (Q. Hong), jjin@seu.edu.cn (J. Jin), wutianxing@seu.edu.cn (T. Wu).

https://doi.org/10.1016/j.knosys.2020.106655
0950-7051/

exact matches are not enough for querying knowledge graphs, because the same kind of knowledge can be represented in diverse graph patterns. For example, *Héctor_Bellerín* (② in Fig. 1(b)) is also a correct answer but with different graph pattern from Q. These subgraph isomorphism methods, however, cannot return such answers that approximately match the query graph Q.

Graph similarity search. In addition to subgraph isomorphism methods, several other works can return similar matches to Q based on different similarity metrics: (1) structural similarity [8,13,18], (2) graph edit distance [19,20], and (3) weak semantic similarity [14,21–23]. For structurally similarity based methods, they assume that an entity in G is more likely to be the target entity in Q, if it is closer to the specific entity in Q. In another word, they only consider the distance between entities (path length) to measure the structural similarity and ignore the semantics of edges that are provided by specific predicates. For example, we may find *Dani_Ceballos* (① in Fig. 1(b)) and *Unai_Emery* (③ in Fig. 1(b)) as the answers except *Héctor_Bellerín* (② in Fig. 1(b)) through GraB [8] and NeMa [13], because *Dani_Ceballos* and *Unai_Emery* have shorter distances to *Spain* than *Héctor_Bellerín*. In fact, *Héctor_Bellerín* should be a correct answer instead of *Unai_Emery*, because *Unai_Emery* is a manager of *Arsenal* not a player. To be more precise, the semantics of predicates "manage" and "team" are ignored by GraB and NeMa, thus returning a wrong answer. Similar to structural similarity based methods, graph edit distance based methods consider the number of steps of transforming a subgraph match to Q. They still lack of the ability of returning semantic similarity answers, because they also ignore semantics of edges. Unlike the above methods, weak semantic similarity based methods can return some semantically similar answers, but still leave us opportunities for further accuracy improvement. Let us take S4 [14] as an example, it returns the n-hop matches through *string edit distance of entities*. However, a wrong answer *Nemanja_Matić* (④ in Fig. 1(b)) could also be returned, if S4 sets an inappropriate similarity threshold (e.g., 65%). This is because the combined string *England+Person+Serbia* has a similarity of 68.4% to *England+Person+Spain*. Obviously, the string edit distance cannot well represent the real semantics of Q. Moreover, [21,22] relies on some external knowledge to compute the semantic similarity. For example, [21] presents a query engine that integrates a set of transformation tables to capture semantics such as "synonym" and "abbreviation" (e.g., the 2-hop path *birthPlace+city* in ② can be transformed to its synonym, that is, the specific predicate *nationality* in Q). While for [22], it computes the similarity between two entity labels based on the additional ontology information. Without loss generality, both the transformation tables and ontology information are external knowledge, which determines the quality of returned matches to some extent.

Other methods to query knowledge graph. Knowledge graph search can also be conducted by the following query forms: (1) keywords search [24,25], (2) SPARQL search [12,26,27], and (3) natural language search [28–30]. Most of them transform the input texts to query graphs for graph searching (by using above two types of graph query methods), so the limitations of existing graph query methods still exist.

Despite its importance for retrieving answers that approximately match the query graph Q, none of the state-of-the-art work can well support the approximate graph query over knowledge graphs because the following two reasons: (1) lack of consideration of the semantics of the query graph Q, and (2) rely on and sensitive to the external knowledge. For example, S4 [14] cannot return high-quality graph query results, if the quality of external knowledge (instance pairs for a given query graph provided by PATTY [31]) is poor. To overcome these limitations, we propose a semantic-aware graph query method over knowledge graphs for star queries. Unlike the existing methods, our method provides the Top-k answers that semantically similar to Q rather than structurally similar ones, without relying on the external knowledge. Of course, if external knowledge (e.g., semantic rules extracted from ontologies information) is available, we can also incorporate it in our solution to further improve the quality [32,33]. Note that, the exact graph query (i.e., graph isomorphism methods) can be viewed as a special case of our approach when our defined semantic similarity equals to 1.0. The answers are ranked by match scores quantifying the answers' relevance to the given Q. Instead of ranking all the answers after computing their exact match scores, our method presents a bounding technique to accelerate query processing. To be more precise, we can efficiently prune the low quality answers via bounding match scores without evaluating all the candidate answers, thus returning the Top-k answers early. We summarize the key contributions as follows.

- We propose a semantic-aware graph query method over knowledge graphs for star queries, to return semantically similar answers instead of structurally similar ones. We define the match score of an answer based on a knowledge graph embedding model, to measure an answer's semantic relevance to a query graph.
- We present a bounding technique for computing match scores of candidate answers, to accelerate query processing. By using the match score bound, we can efficiently prune the unpromising answers and return the Top-k semantically similar answers without evaluating all the possible candidate answers.
- We evaluate the performance of our method on real-world and large-scale knowledge graphs to confirm the superiority on effectiveness and efficiency.

The reminder of this paper is organized as follows. In Section 2, we first formalize the problem studied in this paper, then present the overview of our approach. Section 3 provides the details of our semantic-aware graph query for star queries through bounding match scores. In Section 4, we present the evaluation of our approach. We discuss related work in Section 5 an conclude the paper in Section 6.

2. Preliminaries and overview

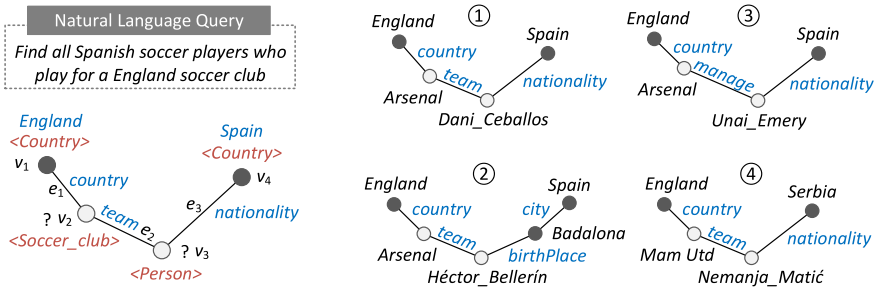
We first formalize the problem studied in this paper. Then we present the overview of our approach. Table 1 lists the frequently used notations in this paper.

2.1. Background

Definition 1 (*Knowledge Graph [13,23]*)**.** We define a knowledge graph as $G = (V, E, L)$, with a node set V, edge set E, and a label function L, where (1) each node $u \in V$ represents an entity, (2) each edge $e \in E$ denotes a relationship between two entities, and (3) L assigns a name and a type on each node, and a predicate on each edge.

Example 1. We assume each node u in G is associated with a type and a unique name [23,30], e.g., $L(u).type = Country$ and $L(u).name = Spain$. For each edge e, it has a predicate such as $L(e) = nationality$. If the type of a node in G is unknown, we employ a probabilistic model-based entity typing method to assign a type on it [34].

A star query aims to discover a target entity from the knowledge graph G by providing specific entities and predicates. We define the star query graph as follows.



(a) Query graph (b) Subgraph matches

Fig. 1. An example of a query graph and some subgraph matches.

Table 1
Frequently used notations.

Notation	Description
G	A knowledge graph
Q^*	A star query graph
$V^s_{Q^*}$	A set of specific nodes in Q^*, each $v^s \in V^s_{Q^*}$ is a specific node
$V^t_{Q^*}$	A set of target nodes in Q^*, each $v^t \in V^t_{Q^*}$ is a target node
v^p	The pivot node in Q^*
$\phi^s(v^s)$	The anchor node of a specific node $v^s \in V^s_{Q^*}$
$\phi^t(v^p)$	A set of answers to Q^*
$\varphi(u_i, \phi^s(v^s))$	The closeness score of an answer $u_i \in \phi^t(v^p)$ to $\phi^s(v^s)$
$\overline{\varphi}$, $\underline{\varphi}$	The upper and lower bound of φ
$S(u_i)$	The match score of an answer $u_i \in \phi^t(v^p)$
$\overline{S}$, $\underline{S}$	The upper and lower bound of S
A_{Q^*}	The Top-k answers to Q^*

Definition 2 (*Star Query Graph [8,35]*)**.** We define a star query graph as $Q^* = (V_{Q^*}, E_{Q^*}, L_{Q^*}, v^p)$, which can be viewed as a join of several sub-query graphs at a pivot node v^p. Specifically, a sub-query graph is defined as a graph $Q = (V_Q, E_Q, L_Q)$, with a query node set V_Q, query edge set E_Q, and a label function L_Q, where $V_Q = V^s_Q \cup V^t_Q$ consists of two subsets of which $V^s_Q = \{v^s\}$ is a set of *specific nodes* (both the type and name of a specific node are known), and $V^t_Q = \{v^t\}$ refers to *target nodes* (only the type of a target node is known). Given these sub-query graphs, we have (1) $V_{Q^*} = V^s_{Q^*} \cup V^t_{Q^*}$, where $V^s_{Q^*} = \bigcup V^s_Q$, $V^t_{Q^*} = \bigcup V^t_Q$, (2) $E_{Q^*} = \bigcup E_Q$, (3) $L_{Q^*} = L_Q$, and (4) all the sub-query graphs intersect at a same target node (called pivot node), denoted by v^p.

Example 2. Fig. 1(a) shows a star query graph consisting of two sub-query graphs: (1) find a soccer player who plays for a England soccer club ($Q_1 : \langle v_1\text{-}e_1\text{-}v_2\text{-}e_2\text{-}v_3\rangle$), (2) find a Spanish soccer player ($Q_2 : \langle v_4\text{-}e_3\text{-}v_3\rangle$), and (3) two sub-query graphs intersect at a same target node v_3 and we call v_3 a pivot node of this star query graph.

Definition 3 (*Star Query Graph Match*)**.** Given a knowledge graph $G = (V, E, L)$ and a star query graph $Q^* = (V_{Q^*}, E_{Q^*}, L_{Q^*}, v^p)$. A star query graph match needs to satisfy: (1) there is a injection ϕ^s: $V^s_{Q^*} \to V$ for each specific node $v^s \in V^s_{Q^*}$, that is, $\phi^s(v^s) = u | u \in V$ (s.t. $L(u).type = L_{Q^*}(v^s).type$, $L(u).name = L_{Q^*}(v^s).name$). We call $\phi^s(v^s)$ an *anchor node* in G, (2) there is a one-to-many relation ϕ^t: $V^t_{Q^*} \to V$ for each target node $v^t \in V^t_{Q^*}$, that is, $\phi^t(v^t) = \{u_i | u_i \in V\}$ (s.t. $L(u_i).type = L_{Q^*}(v^t).type$). We call $u_i \in \phi^t(v^t)$ an candidate *answer* of v^t, and (3) for each path $v^s v^t$ in Q^*, $\overline{\phi^s(v^s)u_i}$ ($u_i \in \phi^t(v^t)$) must be a path in a star query graph match.

For instance, all subgraph matches shown in Fig. 1(b) are star query graph matches to the star query graph provided in Fig. 1(a).

Definition 4 (*A candidate Answer to a Star Query Graph*)**.** Given a star query graph $Q^* = (V_{Q^*}, E_{Q^*}, L_{Q^*}, v^p)$, we define an answer of the pivot node v^p, denoted by $u_i \in \phi^t(v^p)$, as an answer of the star query graph Q^*.

Example 3. In Fig. 1(b), all answers of pivot node v_3 (such as *Dani_Ceballos*, *Unai_Emery*, etc.) are candidate answers of the star query graph in Fig. 1(a).

Given a star query graph Q^*, we try to identify the best k answers to Q^*. In this paper, we define a match score of an answer to quantify the semantic similarity of an answer to Q^*. Intuitively, an answer $u_i \in \phi^t(v^p)$ is better than others if u_i is closer to all anchor nodes $\{\phi^s(v^s)\}$ than other answers. The term "closer" not only means the shorter distance between $\phi^s(v^s)$ and u_i, but also indicates the closer semantics of path $\overline{\phi^s(v^s)u_i}$ in G to path $\overline{v^s v^p}$ in Q^* ($u_i \in \phi^t(v^p)$). Similar to [6], we first define the *closeness score* of an answer as follows, based on which we then provide the *match score* of an answer.

Definition 5 (*Closeness Score*)**.** Given an answer $u_i \in \phi^t(v^p)$ to a star query graph Q^* and an anchor node $\phi^s(v^s)$, we denote the closeness score of u_i to $\phi^s(v^s)$ by $\varphi(u_i, \phi^s(v^s))$. In this paper, we introduce a semantic-based metric for computing $\varphi(u_i, \phi^s(v^s))$ and the details are discussed in Section 3.3.

Definition 6 (*Match Score*)**.** Given an answer $u_i \in \phi^t(v^p)$ to Q^* with the closeness score $\varphi(u_i, \phi^s(v^s))$ to each anchor node $\phi^s(v^s)$, we define the match score of u_i to Q^*, denoted by $S(u_i)$, as the sum of all closeness scores as Eq. (1).

$$S(u_i) = \sum_{v^s \in V^s_{Q^*}} \varphi(u_i, \phi^s(v^s)) \tag{1}$$

A greater match score indicates that an answer is more similar to the star query graph Q^* than others, so the best answer is the one with the greatest match score. Given these definitions above, we formalize the problem studied in this paper as follows.

Top-k approximate star query. *Given a star query graph Q^* and a knowledge graph G, we find the Top-k approximate answers A_{Q^*} based on the match score $S(u_i)$. To be more precise, the k answers in A_{Q^*} must have a greater $S(u_i)$ than other answers.*

$$A_{Q^*} = \{\arg\max_{u_i} S(u_i)\} \tag{2}$$
$$s.t. \quad |A_{Q^*}| = k \quad \& \quad u_i \in \phi^t(v^p)$$

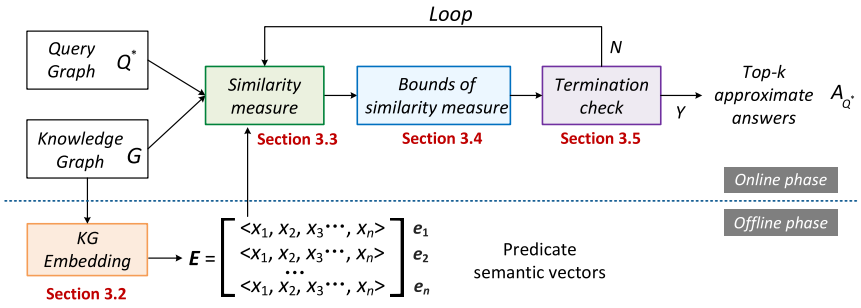


Fig. 2. Pipeline of our method.

2.2. Overview of our approach

We present a semantic-aware graph query method over knowledge graphs for star queries, based on bounding match scores. Fig. 2 shows the pipeline of our method.

Step 1. We leverage a knowledge embedding model to represent the predicates of a knowledge graph G in a vector space offline, denoted by $\boldsymbol{E} = \{\boldsymbol{e_1}...\boldsymbol{e_n}\}$, where $\boldsymbol{e}$ is the predicate vector of an edge $e \in E$. Hence, we can use the similarity between two predicate vectors to measure the semantic similarity between two edges (Section 3.2).

Step 2. We take a predicate semantic space $\boldsymbol{E}$ and a star query graph Q^* as the input to compute the closeness score and match score as similarity measures to evaluate how semantically similar an answer is to Q^* (Section 3.3).

Step 3. We propose a bounding technique for match score computation to efficiently prune unpromising answers, improving the efficiency of our method (Section 3.4).

Step 4. We determine the final Top-k approximate answers to Q^* according to the bounding match scores (Section 3.5). If there are no sufficient answers (e.g., less than k) are found, then we repeat above operations until the best k answers are returned.

3. Semantic-aware graph query through bounding match scores

In this section, we first provide an algorithm overview of our semantic-aware graph query method. Then we discuss four major components of our approach in detail.

3.1. Algorithm overview

Given a star query graph Q^*, our semantic-aware graph query method (shown in Algorithm 1) is divided to an offline and an online phase, respectively.

Offline phase. We capture the semantics of query edges in Q^* via a knowledge graph embedding model, so the predicate semantic space $\boldsymbol{E}$ is computed as an output (line 2). Many knowledge graph embedding models can be selected as the parameter *model* in line 2, we show the effect of different models on our method in Section 4.4.

Online phase. The basic idea can be concluded as follows: (1) we detect the candidate answers $\phi^t(v^p) = \{u_1...u_n\}$ through a multi-sources BFS starting from all anchor nodes $S_a = \{\phi^s(v^s)\}$ (lines 5–6), (2) we calculate the bounds of closeness score of each answer $u_i \in \phi^t(v^p)$ to each anchor node $\phi^s(v^s)$ based on the predicate semantic space $\boldsymbol{E}$, denoted by $\overline{\varphi}(u_i, \phi^s(v^s))$ (upper bound) and $\underline{\varphi}(u_i, \phi^s(v^s))$ (lower bound) (lines 7–8), (3) we compute the bounding match score of each answer u_i to a star query graph Q^*, denoted by $\overline{S}(u_i)$ (upper bound) and $\underline{S}(u_i)$ (lower bound) (lines 9–10), and (4) we determine the approximate Top-k answers according to the bounding match scores (line 4) and return them to users (line 11). We discuss the details of the fourth step (*terminationCheck*(A_{Q^*})) in Algorithm 2, Section 3.5.

Algorithm 1: Semantic-aware Top-k graph query

Data: Query graph Q^*, Knowledge graph G
Result: Answer set A_{Q^*}
// Initialization
1 $A_{Q^*} = \{v | v \in \phi^t(v^p)\}$;
// Offline: KG embedding
2 $\boldsymbol{E} = \{\boldsymbol{e} | e \in E\} = KGEmbedding(G, model)$;
// Online: Main procedure of Top-k graph query
3 $S_a = getAnchorNodes(V_{Q^*})$;
4 **while** *terminationCheck*(A_{Q^*})! = *true* **do**
  // multi-sources BFS
5 **for** $\forall \phi^s(v^s) \in S_a$ **do**
6 1-step BFS iteration from $\phi^s(v^s)$;
  // Closeness score with bounds
7 **for** $\forall u_i \in A_{Q^*}$ **do**
8 $\langle \overline{\varphi}, \underline{\varphi} \rangle = getClosenessScore(u_i, \phi^s(v^s), \boldsymbol{E})$;
9 **for** $\forall u_i \in A_{Q^*}$ **do**
  // Match score with bounds
10 $\langle \overline{S}(u_i), \underline{S}(u_i) \rangle = getMatchScoreBound(\{\langle \overline{\varphi}, \underline{\varphi} \rangle\})$;
11 **return** A_{Q^*};

Remarks. In our implementation, we initialize A_{Q^*} as all candidate answers in the knowledge graph G of which each $u_i \in A_{Q^*}$ has the same type as v^p (line 1), then we identify the best k answers from A_{Q^*} through our semantic-aware graph query method. In order to quickly initialize A_{Q^*}, we build a hash index for all types in G, each type in G is recorded as a *key*, and the *value* refers to a list of entities in G that stored in lexicographical order of their names and have the same type as *key*. Given a star query Q^*, we can use this hash index to retrieve all the entities with the same type as the pivot node v^p to form A_{Q^*}. Because we focus on the semantic-aware graph query method in this paper, we skip the details of the index construction, which is very efficient by using a distributed data processing framework, such as Spark [36], Flink [37], etc.

In the following sections, we will introduce the details of (1) knowledge graph embedding (offline phase), (2) similarity measure (closeness score and match score), (3) bounds of similarity measure, and (4) termination check.

3.2. Knowledge graph embedding

In this paper, we leverage a knowledge graph embedding model to capture the semantics of query edges in a star query

graph Q^*. The basic idea of knowledge graph embedding is to represent each predicate and entity in a knowledge graph G as an n-dimensional semantic vector, such that the original structures and relations in G are preserved in these learned semantic vectors [5]. Briefly, it takes a knowledge graph G as input, and return a predicate semantic space and an entity semantic space as output. We summarize the main idea of most existing knowledge graph embedding models [38–40] as follows: (1) initialize the vectors of head entity h, tail entity t, and predicate r in a triple $\langle h, r, t\rangle$, denoted by $\langle \boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}\rangle$, (2) find a function $g()$ of $\boldsymbol{h}, \boldsymbol{r}$ and optimize $g()$ to satisfy $\boldsymbol{t} \approx g(\boldsymbol{h}, \boldsymbol{r})$ (e.g., $\boldsymbol{h}+\boldsymbol{r} \approx \boldsymbol{t}$ for TransE [39]). As a result, we can finally obtain embeddings of a knowledge graph.

Example 4. Fig. 3 shows an example of TransE [39]. Several triples (⟨subject, predicate, object⟩) in a knowledge graph G are provided in the left part of Fig. 3. Considering the embeddings of G in the right part of Fig. 3, where red arrows indicate the vectors of predicates. Note that, the predicate "product" is more semantically similar to the predicate "assembly" than the predicate "language", because the vectors of "product" and "assembly" have a smaller angle than the one between vectors of "product" and "language". This can be explained as that "product" and "assembly" have the same head entity "Germany" and similar tail entities "BMW_320" and "BMW_X6" (both tail entities have the same type ⟨Automobile⟩, while "language" indicates a relationship between ⟨Country⟩ and its ⟨Language⟩ not a relationship between ⟨Country⟩ and ⟨Automobile⟩.

As shown in the above example, the smaller the angle between two predicate vectors is, the two predicate vectors are more similar, so the two predicates are more semantically similar. Therefore, an effective way to measure the semantic similarity between two predicates is to take the cosine similarity between two predicate vectors as the metric. Given the predicate semantic space $\boldsymbol{E} = \{\boldsymbol{e_1}...\boldsymbol{e_n}\}$ obtained through a knowledge graph embedding model, we compute the cosine similarity between two predicate vectors by Eq. (3) to measure the semantic similarity between two predicates, e.g., *sim*(*nationality*, *birthPlace*).

$$sim(L_Q(e), L(e')) = \frac{\boldsymbol{e} \cdot \boldsymbol{e}'}{\|\boldsymbol{e}\| \times \|\boldsymbol{e}'\|} \tag{3}$$

By using the predicate similarity defined in Eq. (3), we can view a subgraph match to a star query graph Q^* in another perspective, that is, a weighted subgraph match. Given the query graph provided in Fig. 4(a), Fig. 4(b) and Fig. 4(c) show its subgraph match and weighted subgraph match, respectively. A subgraph match can change to a weighted subgraph match by replacing predicates on edges by semantic similarities between edges, e.g., *sim*(*nationality*, *birthPlace*) = 0.98, and *sim*(*nationality*, *city*) = 0.92.

Therefore, the problem of identifying whether a subgraph match is semantically similar to Q^* is transformed to the problem of evaluating the semantic similarity of a weighted subgraph match to Q^*. In this paper, we use the closeness score of an answer and the match score based on the closeness score as the similarity measure to evaluate the semantic similarity of a weighted subgraph match to Q^*. If the answer *Héctor_Bellerín* in Fig. 4(c) has greater closeness scores to both anchor nodes *England* and *Spain*, then it has a greater match score, which indicates that *Héctor_Bellerín* is likely to be a correct answer to the query graph in Fig. 4(a). We next introduce the similarity measure (closeness score and match score) in Section 3.3.

3.3. Similarity measure

Given a candidate answer $u_i \in A_{Q^*}$ and an anchor node $\phi^s(v^s)$ from a knowledge graph G, we now discuss how to compute the closeness score $\varphi(u_i, \phi^s(v^s))$ between u_i and $\phi^s(v^s)$. According to some previous related works, the closeness score can be modeled based on the path length between u_i and $\phi^s(v^s)$ [6,8,13,18,35]. The basic idea is that the shorter the path length is, the higher the closeness score can achieve. An obvious limitation of this method is that it is difficult to measure whether two nodes are semantically close regarding a certain predicate. Considering two possible paths between a person and a country and a given predicate *nationality*, for example, ⟨*Person*⟩-*live_in*-⟨*Country*⟩ and ⟨*Person*⟩-*birthPlace*-⟨*City*⟩-*city*-⟨*Country*⟩. Although the former is shorter in length, the semantic information expressed by the latter is closer to the given predicate *nationality*. This motivates us to measure the closeness score by considering both the path length and semantics of this path. For the convenience of introduction, we use an example shown in Fig. 5 to express our basic intuitions.

Intuition. We compute the closeness score $\varphi(u_i, \phi^s(v^s))$ between a candidate answer u_i and an anchor node $\phi^s(v^s)$ based on the following intuitions.

- **I1**: Given an n-hop path between u_i and $\phi^s(v^s)$, each edge has a predicate similarity (Eq. (3)) to one query edge in the given query graph Q^*, which can be viewed as a weight w of this edge. Hence, the closeness score of u_i to $\phi^s(v^s)$ should be a function $f(w_1...w_n)$ of all predicate similarities appearing at the path $\overline{\phi^s(v^s)u_i}$.
  For example, the third subfigure in Fig. 5 shows a 2-hop path between a ⟨*Person*⟩ and a ⟨*Country*⟩, with two predicate *birthPlace* and *city*. Each predicate has a predicate similarity to a given predicate *nationality*, that are, 0.98 and 0.92, respectively. Note that, any single predicate similarity (0.98 or 0.92) cannot represent the overall semantic similarity of the entire path to *nationality*, and two predicate similarities must be considered at the same time to measure how semantically similar the *birthPlace+city* is to *nationality*.
- **I2**: As mentioned above, the distance between u_i and $\phi^s(v^s)$ is another factor that affects the closeness score. In fact, the closeness score $f(w_1...w_n)$ should be sensitive to the path length, especially when different paths show the similar meaning. To be more precise, it is reasonable that $f(w_1...w_n)$ decreased as path length increasing. This is because the semantic information carried by a path would attenuate during a long distance transmission [8].
  Let us consider the middle two subfigures in Fig. 5. Although both paths show the semantic meaning that a person's nationality is a certain country, we are more likely to believe that the second subfigure claims a 100% accurate fact compared with the third subfigure. This is because a person who was born in a country can have the possibility of having a different nationality.
- **I3**: For a knowledge graph, it is very common to have multiple paths between u_i and $\phi^s(v^s)$. Each path has a semantic similarity to the given predicate, we should consider a comprehensive closeness score based on all paths' semantic similarities.
  As the fourth subfigure shows in Fig. 5, there are two paths between a ⟨*Person*⟩ and a ⟨*Country*⟩. We cannot tell whether this person is Spanish or not through any single path, but if the two paths are considered at the same time, then the possibility that this person is Spanish should be increased to a certain extent. Hence, we need to consider the semantic similarities of all paths when computing the closeness score.

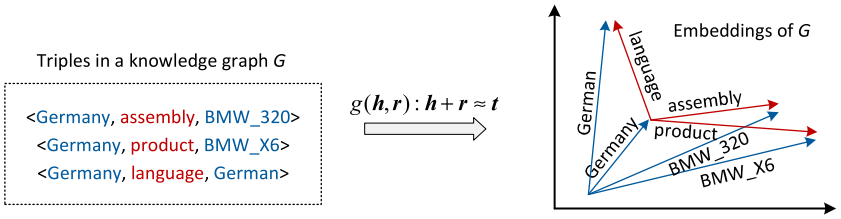
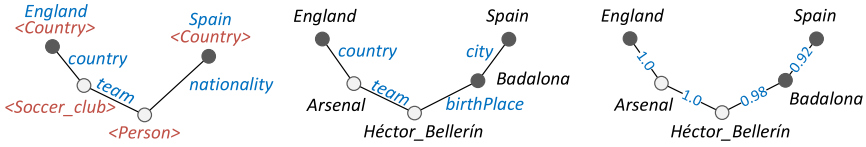


Fig. 3. An example of TransE model.



(a) Query graph (b) A subgraph match (c) A weighted subgraph match

Fig. 4. An example of weighted subgraph match (semantic similarities as weights).

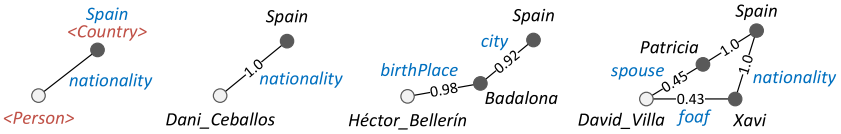


Fig. 5. Examples for closeness score computation.

According to above intuitions, we calculate the closeness score of u_i to $\phi^s(v^s)$ by Eq. (4), which considers two different cases.

$$\varphi(u_i, \phi^s(v^s)) = \begin{cases} \prod_{w_j \in \overline{\phi^s(v^s)u_i}} w_j & \text{Case 1} \\ \min\{\max\{\varphi(u_l, \phi^s(v^s))|u_l \in N(u_i)\}, \\ \quad \sum_{u_l \in N(u_i)} \varphi(u_i, \phi^s(v^s)|u_l)\} & \text{Case 2} \end{cases} \tag{4}$$

Case 1. For the case that only one path between u_i and $\phi^s(v^s)$, we use the weight product of all weights (product of all predicate similarities) appearing at the path $\overline{\phi^s(v^s)u_i}$ as the closeness score (intuition **I1**). Since each weight belongs to a range [0, 1], this closeness score is monotonically decreasing as path length increases (intuition **I2**).

Case 2. While for the case that more than one path between u_i and $\phi^s(v^s)$, we use $\sum_{\forall u_l \in N(u_i)} \varphi(u_i, \phi^s(v^s)|u_l)$ to represent the comprehensive closeness score of u_i to $\phi^s(v^s)$ (intuition **I3**), where $N(u_i)$ is a set of neighbors of u_i and $\varphi(u_i, \phi^s(v^s)|u_l)$ indicates the closeness score calculated based on the path $\overline{\phi^s(v^s)u_i}$ through one neighbor node u_l of u_i. Moreover, in order to ensure that the answer having shorter distance to $\phi^s(v^s)$ has a higher score (intuition **I2**), we constraint the comprehensive closeness score to no greater than the maximal closeness score of u_i's previous-hop neighbor, that is, $\max\{\varphi(u_l, \phi^s(v^s))|u_l \in N(u_i)\}$.

Example 5. In Fig. 5, we show three examples for closeness score computation. There are three soccer players whose nationality might be *Spain*. For the first two persons, we directly compute the closeness score as 1.0 and $0.92 \times 0.98 = 0.901$ (**Case 1**), respectively. While for the last person, there are two paths connecting to *David_Villa*, so we compute the comprehensive closeness score (**Case 2**) as $\min\{\max\{1.0, 1.0\}, 1.0 \times 0.45 + 1.0 \times 0.43\} = 0.88$. Note that, *Davia_Villa*'s wife and friend are Spanish, which indicates that he probably is a Spanish, too. By using the comprehensive closeness score, we can get this probability as 88%. Finally, we output the Top-3 answers in the order of ⟨*Dani_Cebollos, Héctor_Bellerín, David_Villa*⟩.

Analysis. Recall the Algorithm 1, we conduct a BFS from each anchor node $\phi^s(v^s)$ to find the paths to each candidate answer u_i. If we find all paths to u_i from all $\{\phi^s(v^s)\}$, then we can calculate the exact closeness score (Eq. (4)) and match score $S(u_i)$ (Eq. (1)). And the k answers with the greatest $S(u_i)$ are returned as the Top-k answers. This procedure is clear and easy to implement. However, it suffers from the efficiency issue, especially for a large-scale knowledge graph. In a knowledge graph, the path search space is usually large because the high connectivity among nodes. For instance, the average node degree in DBpedia 3.9 dataset is nearly 24, so that if we want to find all paths to an answer within 3 hops, the possible candidate paths would be $24^3 = 13\,824$ on average. The large path search space will affect the efficiency significantly.

This motivates us to present a novel technique for bounding closeness score and match score during runtime. Through this bounding technique, we can determine the Top-k answers earlier without calculating the exact closeness score and match score, by efficiently pruning unpromising answers. We next introduce the bounds of closeness score and match score in Section 3.4.

3.4. Bounds of similarity measure

According to Definition 6, we compute the bounds of match score $\overline{S}(u_i)$ and $\underline{S}(u_i)$ based on the bounds of closeness score $\overline{\varphi}(u_i, \phi^s(v^s))$ and $\underline{\varphi}(u_i, \phi^s(v^s))$ as follows.

$$\begin{cases} \overline{S}(u_i) = \sum_{v^s \in V^s_{Q^*}} \overline{\varphi}(u_i, \phi^s(v^s)) \\ \underline{S}(u_i) = \sum_{v^s \in V^s_{Q^*}} \underline{\varphi}(u_i, \phi^s(v^s)) \end{cases} \tag{5}$$

In order to compute $\overline{S}(u_i)$ and $\underline{S}(u_i)$, we need to compute bounds of closeness score in advance. Hence, we show how to compute $\overline{\varphi}$ and $\underline{\varphi}$ first.

Intuition. In Algorithm 1, we conduct a BFS from each anchor node, so we can detect some nodes at the nth step of BFS. For these detected nodes, the exact closeness scores are computed by Eq. (4). Since we can finally find other undetected nodes through further BFS steps starting from these detected nodes, it is possible to estimate undetected nodes' upper and lower

bounds of closeness scores based on the exact closeness scores of detected nodes computed at the nth step.

Upper bound of closeness score. We denote a set of newly detected nodes in the nth BFS step by D^n. Every node $u_l \in D^n$ has an exact closeness score, denoted by $\varphi^n(u_l, \phi^s(v^s))$. We also introduce a wildcard node u_x to represent all undetected answers after n BFS steps. Note that, there may exist a potential path from $\phi^s(v^s)$ to u_x through u_l. Hence, we use $\overline{\varphi}^n(u_x, \phi^s(v^s)|u_l)$ to denote the upper bound of closeness score of u_x calculated at the nth BFS step from the detected node u_l. We compute it by Eq. (6), where $m(u_l)$ is the maximum semantic similarity of all adjacent edges of u_l.

$$\overline{\varphi}^n(u_x, \phi^s(v^s)|u_l) = \varphi^n(u_l, \phi^s(v^s)) \cdot m(u_l) \tag{6}$$

Given a set of upper bounds of closeness score $\{\overline{\varphi}^n(u_x, \phi^s(v_j^s)|u_l)\}$ for all $u_l \in D^n$, we calculate the upper bound of comprehensive closeness score for u_x as follows.

$$\overline{\varphi}^n(u_x, \phi^s(v^s)) = \min\{\max\{\varphi^n(u_l, \phi^s(v^s))|u_l \in D^n\}, \sum_{u_l \in D^n} \overline{\varphi}^n(u_x, \phi^s(v^s)|u_l)\} \tag{7}$$

Example 6. In Fig. 6, we show an example for bounding closeness score. Given a knowledge graph with semantic similarities on the edges as the input (left part), we conduct a BFS from node u_1, and try to compute the upper bound of closeness score for nodes u_4 and u_5. A snapshot after the 1st BFS step is provided in the middle part, where $D^1 = \{u_2, u_3\}$. Since each node in D^1 have a chance to connect to arbitrary node $u_x|x = 4, 5$, so we can compute $\overline{\varphi}^1(u_x, u_1|u_l)$ for each $u_l \in D^1$. Then we obtain the upper bound of comprehensive closeness score of u_x as $\overline{\varphi}^1(u_x, u_1) = 0.74$. Actually, if we continue to the 2nd BFS step, then we can get the exact comprehensive closeness score of u_4 and u_5 as 0.59 and 0.74, respectively. Both of them are bounded by the upper bound of 0.74.

Lower bound of closeness score. Since we might not find a path from an anchor node $\phi^s(v^s)$ to arbitrary node u_x through a detected node $u_l \in D^n$ at nth BFS step, we just define the lower bound of comprehensive closeness score of u_x as $\underline{\varphi}^n(u_x, \phi^s(v^s)) = 0$.

Bounds refinement. During the graph querying, we keep updating the upper and lower bounds of closeness score for all candidate nodes in A_{Q^*} at each BFS step. If a candidate answer u_i is detected at the nth iteration. Then we have $\overline{\varphi}^n = \underline{\varphi}^n = \varphi$ (φ is the exact score computed by Eq. (4)). After we obtain the bounds of closeness score, we can update the bounds of match score $\overline{S}$ and $\underline{S}$ at each BFS step (Eq. (5)). Given the bounds of match score for each candidate answer, we can determine the Top-k answers through the *termination check* procedure. We discuss it in Section 3.5.

Convergence of bounding matching score. We need to ensure that the graph querying based on the bounding matching score is convergent, which means that the upper and lower bounds of match score will shrink as the number of BFS steps increases. To be precise, the bounds of match score satisfy $\overline{S}^n \geq \overline{S}^{n+1}$ and $\underline{S}^n \leq \underline{S}^{n+1}$.

Lemma 1. *Suppose that we are now at the nth BFS step, then the upper bound of closeness score of an arbitrary node u_x, denoted by $\overline{\varphi}^n(u_x, \phi^s(v^s))$, is bounded ($\leq$) by the value of $\max\{\varphi^n(u_l, \phi^s(v_s))|u_l \in D^n\}$.*

Proof. We prove Lemma 1 based on Eq. (7). To make the proof more clear, we denote $\max\{\varphi^n(u_l, \phi^s(v^s))|u_l \in D^n\}$ by a symbol A, and denote $\sum_{u_l \in D^n} \overline{\varphi}^n(u_x, \phi^s(v^s)|u_l)$ by a symbol B. We first assume that $B \geq A$, then we have $\overline{\varphi}^n(u_x, \phi^s(v^s)) = A$ based on Eq. (7). In another case, if $B \leq A$ then we have $\overline{\varphi}^n(u_x, \phi^s(v_j^s)) = B \leq A$. In summary, $\overline{\varphi}^n(u_x, \phi^s(v^s)) \leq A = \max\{\varphi^n(u_l, \phi^s(v_s))|u_l \in D^n\}$ holds for both cases.

Lemma 2. *Given a set D^n is obtained at the nth BFS step, then we have $\max\{\varphi^n(u_l, \phi^s(v^s))|u_l \in D^n\}$ is bounded ($\leq$) by the upper bound of closeness score for u_x calculated at the $(n-1)$th BFS step, denoted by $\overline{\varphi}^{n-1}(u_x, \phi^s(v^s))$.*

Proof. Since some of the undetected nodes (u_x) in the (n-1)th BFS step will be detected in the nth BFS iteration and are recorded in D^n, it is natural that we have $\overline{\varphi}^{n-1}(u_x, \phi^s(v^s)) \geq \varphi^n(u_l, \phi^s(v^s))$, where $u_l \in D^n$. Hence, $\overline{\varphi}^{n-1}(u_x, \phi^s(v^s)) \geq \max\{\varphi^n(u_l, \phi^s(v^s))|\forall u_l \in D^n\}$ holds.

Theorem 1. *Given a certain undetected node u_x, the upper bound of closeness score of u_x that computed at each BFS step is decreasing as more BFS steps processed. To be precise, we have $\overline{\varphi}^n(u_x, \phi^s(v^s)) \leq \overline{\varphi}^{n-1}(u_x, \phi^s(v^s))$.*

Proof. According to Lemma 1, we have $\overline{\varphi}^n(u_x, \phi^s(v^s)) \leq \max\{\varphi^n(u_l, \phi^s(v^s))|u_l \in D^n\}$. According to Lemma 2, we have $\overline{\varphi}^{n-1}(u_x, \phi^s(v^s)) \geq \max\{\varphi^n(u_l, \phi^s(v^s))|u_l \in D^n\}$. Hence, $\overline{\varphi}^n(u_x, \phi^s(v^s)) \leq \overline{\varphi}^{n-1}(u_x, \phi^s(v^s))$ holds.

Theorem 2. *The graph querying based on the bounding match score is convergent.*

Proof. Since the upper bound of closeness score $\overline{\varphi}$ monotonically decreases as the BFS step increases (Theorem 1), the upper bound of match score $\overline{S}$ is also monotonically decreasing according to Eq. (5) ($\overline{S}^n \geq \overline{S}^{n+1}$). Moreover, the lower bound of closeness score $\underline{\varphi}$ is monotonically increasing because it will increase to the exact value φ from 0 when an answer is detected. Hence, the lower bound of match score $\underline{S}$ is monotonically increasing too ($\underline{S}^n \leq \underline{S}^{n+1}$). Therefore, the upper and lower bounds of match score will shrink as the BFS step increases until $\overline{S} = \underline{S} = S$, which indicates that the graph querying based on the bounding match score is convergent.

Remarks. Our bounding technique can be viewed as a prune strategy for computing the match score. The computational complexity is dominated by three parameters, that are, the number of anchor nodes for a given query graph, denoted by m, the average degree of each explored node, denoted by d, and the upper bound of the path length between a candidate answer and an anchor node, denoted by n. Since graph queries usually exhibit strong access locality [41] and most answers could be found in an n-bounded space of a given anchor node, thus we assume n as a upper bound of the path length. As a result, we have the computational complexity in the worst case as $O(m \cdot d^n)$. By using our bounding technique, the efficiency can be improved without materializing the entire path between a candidate answer and each anchor node, which means we can terminate the match score computation early once a certain condition is satisfied. We next introduce such terminate condition and discuss how to return the Top-k answers based on the bounding match score in detail.

3.5. Termination check

The termination check, as shown in Algorithm 2, determines whether the Top-k answers are identified. Assume that we already compute upper and lower bounds of the match score

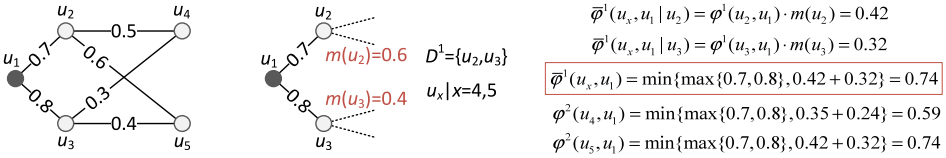


Fig. 6. An example of bounding closeness score.

Algorithm 2: terminationCheck(A_{Q^*})

Data: Answer set A_{Q^*}
Result: Return *true* if find k answers or *false* otherwise
1 Get the kth largest lower bound of match score $\underline{S}(u_k)$;
2 **for** $\forall u_i \in A_{Q^*}$ **do**
3 **if** $\overline{S}(u_i) < \underline{S}(u_k)$ **then**
4 A_{Q^*}.*remove*(u_i);
5 **if** $|A_{Q^*}| = k$ **then**
6 **return** *true*;
7 **else**
8 **return** *false*;

Table 2
The bounds of closeness score for nodes u_6, u_7, u_{10} in Fig. 7.

u_x	$\overline{\varphi}^2(u_x,u_1)$	$\underline{\varphi}^2(u_x,u_1)$	$\overline{\varphi}^1(u_x,u_9)$	$\underline{\varphi}^1(u_x,u_9)$
u_6	0.67	0.67	0.87	0
u_7	0.327	0	0.9	0.9
u_{10}	0.327	0	0.3	0.3

($\overline{S}$ and $\underline{S}$) for each candidate answer in A_{Q^*}, the Top-k termination check consists of two main steps:
Step 1. We find the kth largest $\underline{S}(u_k)$ among all candidates (line 1). $\underline{S}(u_k)$ is the lower bound of match scores of the Top-k answer. In another word, the match score of each Top-k answer must be greater than or equal to $\underline{S}(u_k)$.
Step 2. We remove all candidates having $\overline{S}(u_i)$ less than $\underline{S}(u_k)$ from A_{Q^*}, because they definitely are not the Top-k answers (lines 2–4). The termination check stops when $|A_{Q^*}| = k$. Otherwise, the upper/lower bound should be further refined (lines 5–8).

Example 7. Fig. 7 shows an example of Top-2 answers graph query. In the left part, it is a knowledge graph with semantic similarities on the edges. We aim to find Top-2 answers from three candidates (A_{Q^*} is initialized as $\{u_6, u_7, u_{10}\}$) by conducting two BFS from two anchor nodes u_1 and u_9, respectively. The solid lines and nodes represent the edges and entities that have been detected in the BFS, and the dash lines and nodes represent the undetected parts of the graph. In the 2nd BFS step from the anchor node u_1, we have $D^2 = \{u_5, u_6\}$. While in the 1st BFS iteration from the anchor node u_9, we have $D^1 = \{u_7, u_8, u_{10}\}$. Given D^2 and D^1, the upper and lower bounds of closeness score for candidates u_6, u_7, u_{10} are provided in Table 2. Given these bounds of closeness score, we then compute the bounds of match score for all candidates as $\langle\overline{S}(u_6), \underline{S}(u_6)\rangle = \langle 1.54, 0.67\rangle$, $\langle\overline{S}(u_7), \underline{S}(u_7)\rangle = \langle 1.227, 0.9\rangle$, and $\langle\overline{S}(u_{10}), \underline{S}(u_{10})\rangle = \langle 0.627, 0.3\rangle$. Therefore, we can obtain the Top-2 answers u_6 and u_7 without exploring the whole graph.

4. Experimental study

We present experiment results over real-world knowledge graphs to evaluate (1) effectiveness and efficiency, (2) user study, and (3) effect of KG embedding models.

4.1. Experimental setup

Datasets. We used three real-world knowledge graphs as datasets. (1) DBpedia [2] is an open-domain knowledge base, which is constructed from Wikipedia. (2) Freebase [4] is a knowledge base mainly composed by communities. Since we assume that each entity has a name, we used a Freebase-Wikipedia mapping file [42] to filter 5.7M entities, each entity has a name from Wikipedia. (3) YAGO2 [3] is a knowledge base with information from the Wikipedia, WordNet and GeoNames. In this paper, we only used the CORE portion of Yago (excluding information from GeoName) as our dataset.
Query workload. We used three query workloads to construct the query graphs. (1) QALD-4 [43] is a benchmark for DBpedia. It provides answers for each query. (2) WebQuestions [44] is a benchmark for Freebase. It provides a set of questions, denoted by a quadruple $\langle qText, freebaseKey, relPaths, answers\rangle$. We took the entities and relations from *freebaseKey* and *relPaths* to form query graphs. (3) RDF-3x [45] contains queries for YAGO dataset. It provides SPARQL expressions, but does not provide the answers. To obtain the validation set, we imported YAGO2 to the graph database Neo4j [46] and executed the queries through a sparql-plugin.
Metrics. We adopted three metrics to measure the effectiveness. ***Precision*** (P) is the ratio of correctly discovered answers over all discovered top-k answers. ***Recall*** (R) is the ratio of correctly discovered answers over all correct answers. In addition, we also employed ***F1-measure*** to combine the precision and recall as $F1 = \frac{2}{1/P+1/R}$. For the efficiency, we used the ***average response time*** T of all queries as the metric.

Moreover, we used the ***average performance improvement ratio*** ($p\%$) to evaluate how much our approach outperforms others competitors in the terms of effectiveness metrics as Eq. (8), where $\frac{m_k - m'_k}{m'_k}$ is the performance improvement ratio for each k ($k = \{20, 40, 100, 200\}$), m_k is the metric value of our method and m'_k is the one for others (m_k could be the P, R and $F1$ at arbitrary k). We also evaluate the ***speedup*** of our method to measure the efficiency improvement.

$$p\% = \sum_k \frac{m_k - m'_k}{m'_k} \tag{8}$$

Comparing methods. Since the latest work S4 [23] shows that it outperforms NeMa [13], gStore [12] and BLINK [47], so we compared our approach (named ***Bound***) with ***S4***. Besides, we compared with ***p-hom*** [48] and ***GraB*** [8], the former considers the node similarity and the latter considers the structural similarity for graph query. Moreover, we compared with ***QGA*** [25], which is a keyword-based graph query method.

4.2. Effectiveness and efficiency evaluation

Effectiveness. In this test, we compared our method Bound with other competitors in the terms of P, R and $F1$-measure. We show the results in Figs. 8–10(a, b, c) for a varied k (from $k =$ 20 to $k =$ 200). Observe that, Bound is better than others in the precision metric. This is because our method can find the

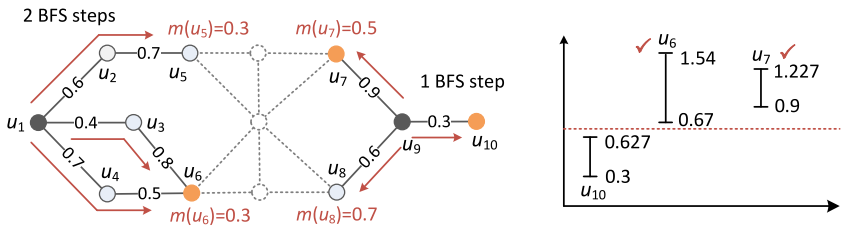


Fig. 7. Case study for Top-2 answers graph query.

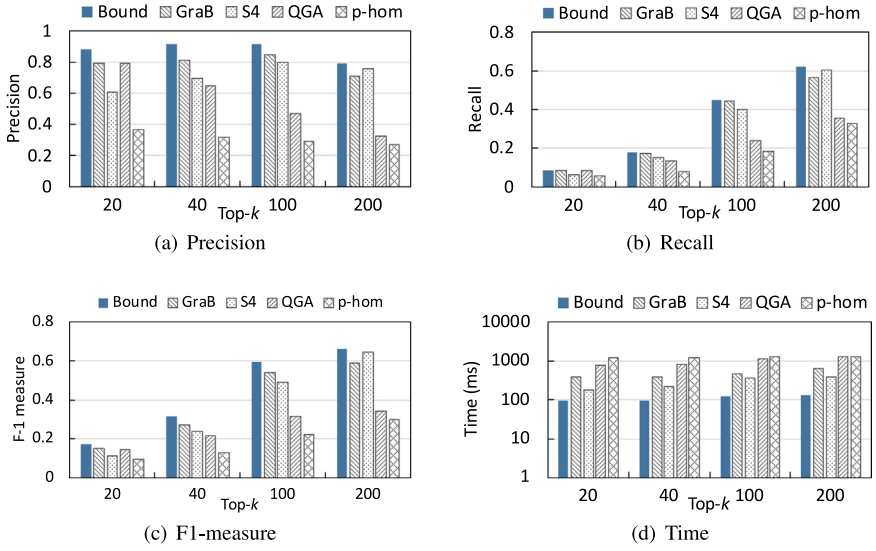


Fig. 8. Effectiveness and efficiency over DBpedia.

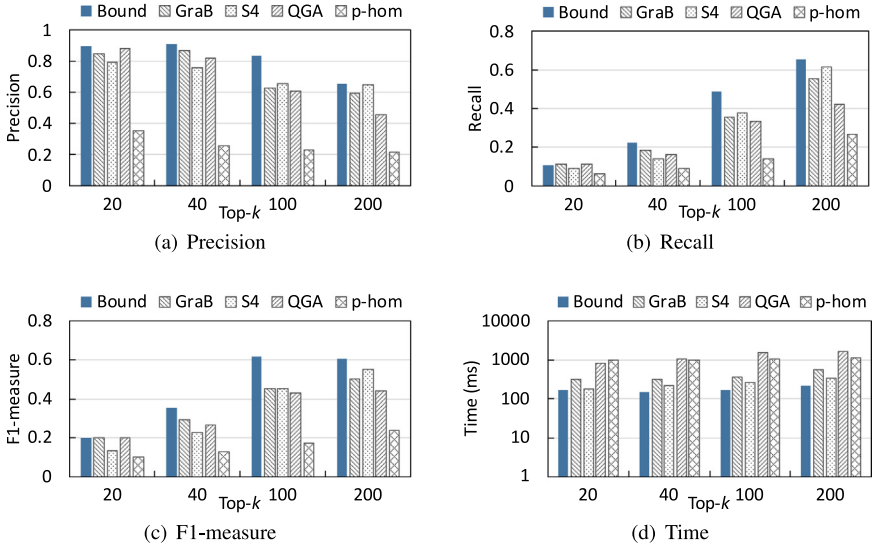


Fig. 9. Effectiveness and efficiency over Freebase.

Table 3
The p% of precision and *speedup* of average response time (DBpedia).

Top-k	**Bound vs. others** (p% of P)				**Bound vs. others** (*speedup* of *Time*)			
	Grab	S4	QGA	p-hom	GraB	S4	QGA	p-hom
$k = 20$	0.177	0.525	0.175	1.536	3.995	1.939	8.223	12.676
$k = 40$	0.184	0.388	0.494	2.047	4.038	2.349	8.598	12.728
$k = 100$	0.142	0.211	1.070	2.293	3.914	2.928	9.379	10.345
$k = 200$	0.119	0.047	1.411	1.896	4.998	2.952	9.995	9.737
AVG	0.156	0.293	0.787	1.943	4.236	2.542	9.049	11.372

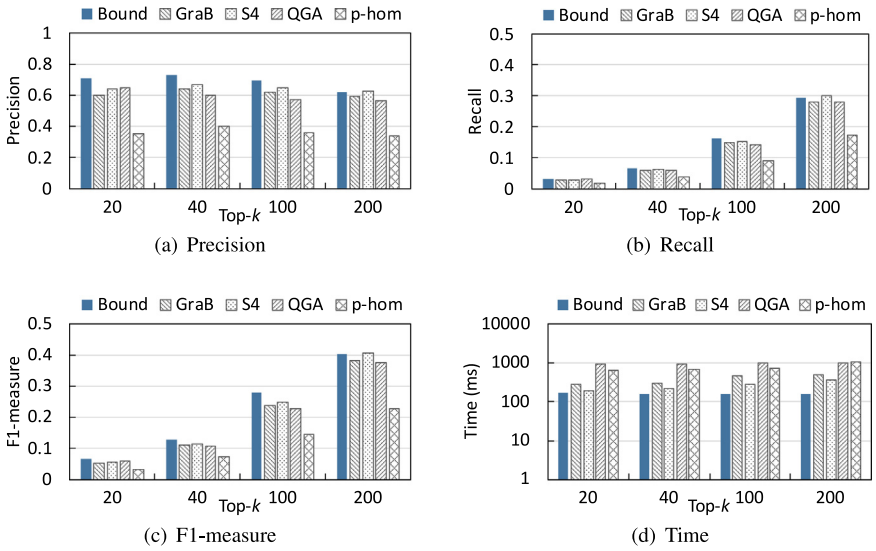


Fig. 10. Effectiveness and efficiency over Yago.

semantically similar answers by considering the semantics of the query edges. Let us take the result over DBpedia as an example to discuss the differences of different methods. Among the other methods, GraB can achieve a better precision for a smaller k ($k = \{20, 40, 100\}$). This is because there are lots of ground truths in QALD-4 benchmark are defined within 2-hop,[1] and GraB uses the path length to indicate the similarity of an answer to the query graph, so it can find more correct answers for the cases of a smaller k. For S4, we show the average precision result of different cases taking different prior knowledge as the input. We note that its precision is relatively lower than Bound, this is because the precision of S4 are significantly affected by the quality of input prior knowledge. S4 performs well in the cases with a high-quality input prior knowledge (the prior knowledge covers the majority of predefined schemas in a knowledge graph), but performs poorly in other cases. S4 is better than GraB for $k = 200$, because many multi-hops answers are found for $k = 200$, and GraB cannot ensure that they are semantically similar to the query graph. Moreover, QGA can only find 1-hop answers, resulting in a rapid reduction in precision as k increases. And p-hom ranks all answers only based on the node similarity, so that many multi-hops wrong answers could be ranked in the Top-k list instead of some 1-hop right answers, leading to a lower precision.

Furthermore, we can see the similar trend in the recall and F1-measure. Note that, recall and F1-measure are very small for $k = \{20, 40\}$, because of the size of validation set is usually larger than k. For example, the query Q117 from QALD-4 has 596 correct answers, so that the recall is very small when we set $k = 20$ (at most $\frac{20}{596} = 3.35\%$).

Efficiency. In this test, we evaluated the efficiency of all methods for a varied k (from $k = 20$ to $k = 200$). The results are given in 8–10(d), we can find that the response time for all methods increases with the increasing of k. This is natural that a larger search space is usually required for achieving more answers. However, our Bound method is the most efficient, because we use the bounding technology to efficiently prune the unpromising candidates during runtime. Among other methods, S4 is better than others, because it uses a summarizing graph to accelerate the query processing. GraB and p-hom perform worse than S4, because their search space becomes too large to converge quickly as the increasing of k. Moreover, QGA takes more time on the query graph assembly, which significantly affects the overall efficiency. From this result, we can conclude that our Bound is more scalable to the value of k.

Performance improvement. In the above tests, we can see that our Bound outperforms other methods in both effectiveness and efficiency metrics. Tables 3–5 provides the details of the performance improvement on *precision* and speedup of the *average response time*. For example, as shown in Table 3, we achieve at least 15.6% improvement on average for precision. And we are at least 2.5 times faster than other methods.

4.3. User study

Since our method returns the Top-k answers to users, we wanted to know if the users are satisfied with the highly-ranked answers (even though the answers are already in the validation set). In other words, we expect that an answer, which is more familiar to the user, must have a higher rank returned by our method. Therefore, we conducted a user study through the *Baidu Data CrowdSourcing Platform*[2] to evaluate the correlation between the Top-k answers returned via our method and the users' preferences. Similar to [49], we used the Pearson Correlation Coefficient (PCC) as the metric to measure such correlation. We did the test according to the steps in [49]. We selected 12 queries from the workloads in this user study. For each query, we set k as the size of validation set. Given Top-k answers, we divided them into several groups according to the match scores, and then generated 30 random pairs of these answers. In order to avoid evaluating two answers with the same match score, we selected answers in a pair from different groups. We presented each pair to 10 annotators and asked for their preference between the two answers (Table 6 for example). Given an answer pair ⟨Opel_Super_6, BMW_320⟩, our method provides the rankings as 62 and 10 for both answers. If more higher ranked answers (e.g., BMW_320) are preferred by more users (shown with ✔), then we can say that the Top-k answers and users' preferences are positively correlated.

Finally, we obtained 12*30*10 = 3600 opinions. For each query, we constructed two lists X and Y based on these opinions to represent our method and annotators' preference of 30 answer pairs. For each answer pair, the value in X is the difference between the two answers' ranks given by our method (e.g., $X_1 =$

[1] For example, there are 234 out of 596 answers of Q117 are 1-hop.

[2] https://zhongbao.baidu.com/?language=en

Table 4
The p% of precision and *speedup* of average response time (Freebase).

Top-k	**Bound vs. others** (p% of P)				**Bound vs. others** (*speedup* of *Time*)			
	Grab	S4	QGA	p-hom	GraB	S4	QGA	p-hom
$k = 20$	0.056	0.131	0.015	1.519	1.873	1.103	4.820	5.914
$k = 40$	0.043	0.201	0.111	2.545	2.153	1.405	6.825	6.736
$k = 100$	0.331	0.276	0.382	2.668	2.069	1.567	9.068	6.231
$k = 200$	0.095	0.007	0.427	2.029	2.535	1.535	7.211	5.054
AVG	0.131	0.154	0.234	2.187	2.158	1.403	6.981	5.984

Table 5
The p% of precision and *speedup* of average response time (Yago).

Top-k	**Bound vs. others** (p% of P)				**Bound vs. others** (*speedup* of *Time*)			
	Grab	S4	QGA	p-hom	GraB	S4	QGA	p-hom
$k = 20$	0.181	0.111	0.090	1.025	1.728	1.144	5.592	3.747
$k = 40$	0.133	0.091	0.213	0.819	1.941	1.363	6.102	4.451
$k = 100$	0.118	0.079	0.221	0.934	3.037	1.784	6.376	4.655
$k = 200$	0.051	0.011	0.102	0.832	3.208	2.359	6.592	6.901
AVG	0.121	0.073	0.157	0.903	2.478	1.662	6.166	4.939

Table 6
An example of answer pairs for user study (Q117 in QALD-4)

Answer1	Our rank	User	Answer2	Our rank	User
Opel_Super_6	62		BMW_320	10	✔
Volkswagen_Passat	72	✔	30_PS	26	

Table 7
Pearson Correlation Coefficient (*PCC*) between our method and Baidu's annotators.

Query	PCC	Query	PCC	Query	PCC	Query	PCC
Q1	0.42	Q4	0.71	Q7	0.63	Q10	0.67
Q2	0.51	Q5	0.69	Q8	0.34	Q11	0.65
Q3	0.56	Q6	0.68	Q9	0.41	Q12	0.74

52 for the first row in Table 6), and the value in Y is the difference between the numbers of annotators preferring the two answers (e.g., $Y_1 = 8$ for the first row in Table 6, if 9 annotators prefer BMW_320 and 1 annotator prefers Opel_Super_6). Given the lists X and Y, we then can calculate the PCC for each query as follows.

$$PCC = \frac{E(XY) - E(X) \cdot E(Y)}{\sqrt{E(X^2) - E(X)^2} \cdot \sqrt{E(Y^2) - E(Y)^2}} \tag{9}$$

The PCC value shows the degree of correlation between the ranks given by our method and the preferences given by annotators. According to [50], the value range of PCC is $[-1, 1]$ and a PCC value in the ranges of [0.5, 1.0], [0.3, 0.5) and [0.1, 0.3) indicates a strong, medium and small positive correlation, respectively. Table 7 shows that our method achieved strong and medium positive correlations with the annotators on 9 and 3 queries, respectively, out of total 12 queries, which indicates that the users are satisfied with the semantically similar answers identified via our method. In another word, a high ranked answer is also much more likely to be preferred by more users.

4.4. Effect of knowledge graph embedding models

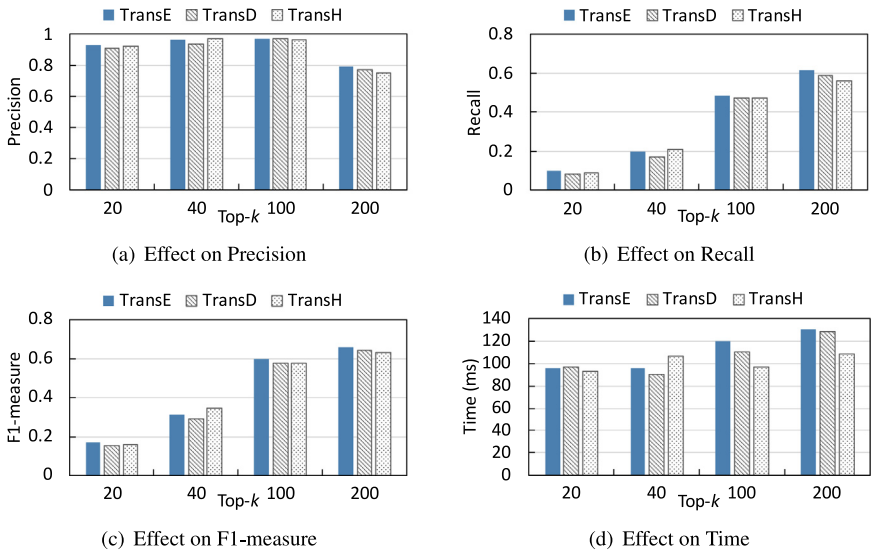
In this test, we evaluate the effect of different knowledge graph embedding models on the effectiveness and efficiency metrics. Since the results over three knowledge graphs show the similar trends, we only show the result over DBpedia. Here, we consider three very common knowledge graph embedding models, TrasE [39] and its two extensions TransD [40] and TransH [38]. Fig. 11(a-c) shows the effectiveness results for three models. Note that, the effectiveness results are less affected by the embedding models. So we can say our method is scalable for the three models. This is because our method is more concerned with whether the model correctly expresses the semantic relationship of two predicates, rather than whether the semantic similarity between two predicates is more accurate. For example, predicate *manufacturer* is a semantically similar predicate to *assembly* in all models (e.g., with semantic similarity at least 90% in all models), then the automobiles are manufactured in Germany will be returned as the results with higher probability than other automobiles designed by German designer. It does not matter if the semantic similarity of $\langle$*manufacturer*, *assembly*$\rangle$ is 93% or 95%, both value would return automobiles are manufactured in Germany as the answers. This is because we always have *manufacturer* is more similar to *assembly* than *designer* in all models. Moreover, Fig. 11(d) shows the effect of different models on the efficiency. The processing time is less affected by the Trans models for each k. In a summary, we only need to select the model with the fastest training speed regardless of the effect of models on the effectiveness and efficiency.

5. Related work

According to how previous approaches search knowledge graph, we categorize related work as follows.

Graph pattern matching. Graph pattern matching is typically defined in terms of subgraph isomorphism [11,12,15,16], which is NP-complete and often too restrictive to capture sensible matches [17]. These methods requires all the nodes and edges in the matches to be exactly the same as the ones in Q. Therefore, we cannot find the approximately similar matches through these methods.

Graph similarity search. Many efforts have been made for the graph similarity search based on different similarity metrics: (1) structural similarity [8,13,18,51], (2) graph edit distance [19,20], and (3) weak semantic similarity [14,21–23]. kGPM [51] studies Top-k graph query for cyclic and complex queries. It allows an edge-to-path mapping, but restricts that the vertex/edge labels specified in the query graph Q should be exactly matched. Besides, Ness [18] and NeMa [13] utilize h-hop neighborhood indexing to unify both the label matching cost and neighborhood matching cost. But it is difficult to predefine a reasonable h for every query and suffer from high index building overhead. GraB [8] is a recent work which utilizes the bound-based Top-k detection mechanism to obtain the answers without indexing. However, these approaches are difficult to find the semantically similar matches to the given query graph. Moreover, KMatch [22] and SLQ [21] can be viewed as concept-level semantic similarity



(a) Effect on Precision (b) Effect on Recall

(c) Effect on F1-measure (d) Effect on Time

Fig. 11. Effect of knowledge graph embedding models.

query. KMatch is an ontology based sub-graph query which can estimate the similarity between two vertex labels. SLQ presents a query engine that integrates the distance based structural transformation and ontology based semantic transformation, but it is hard to deal with more complicated cases. To make use of more specific semantic information, Zheng et al. propose *S*4 [23] to accept some semantic instances as external knowledge and mine the semantic meaning equivalent query pattern. All above approaches do not consider the real knowledge semantics in the query processing, leaving the opportunity for effectiveness improvement.

Query-by-examples. Query-by-Example (QBE) aims to allow users to express their search intentions with examples. GQBE [49] and Exemplar [52] are proposed for searching matches that are same as their counterparts from the examples. Moreover, [53,54] are proposed to pose exemplars characterized by tuple patterns, and identify answers close to exemplar. Our approach can extend these QBE methods by returning more semantically similar answers to the given exemplar queries.

Other methods to query knowledge graph. The knowledge graph search can also be conducted by the following query forms: (1) keywords search [24,25], (2) SPARQL search [12,26,27], and (3) natural language search [28–30]. Most of these methods transform the input texts to query graphs for graph searching, so our graph query approach can be used to improve their performance.

Since we present a novel semantic similarity metric to measure how a subgraph match is semantically similar to the given query graph Q, a literature review on semantic similarity metrics is required an provided as follows.

Corpus-based semantic similarity. Corpus-based approaches [55–58] leverage the information extracted from large corpora such as Wikipedia to measure the semantic similarity [59]. Some works exploit concept associations such as Pointwise Mutual Information [55] or Normalized Google Distance [56]. While other works represent the words with low-dimensional vectors, such as Word2Vec [57], GLOVE [58]. These approaches work well for the text corpora but cannot be easily extend to support knowledge graphs. This is because a knowledge graph is maintained as a big graph, we not only need to consider contextual information but also structural features when measuring the semantic similarity.

Knowledge-based semantic similarity. Knowledge-based approaches use the semantic information contained in the knowledge graph to measure the semantic similarity between concepts, entities, and predicates. A commonly used semantic information is the semantic distance between concepts (e.g., the path between two concepts), the shorter the path is, the ore similar they are [60–64]. Another line of literature represents each entity and predicate in a knowledge graph as an n dimensional semantic vector (i.e., knowledge graph embedding), such that the original structures and relations in the knowledge graph are preserved in these learned semantic vectors [38–40]. To the best of our knowledge, in this paper, we are the first to combine both the distance-oriented and embedding-based semantic similarities to measure how semantic similarity an answer is to the given query graph.

6. Conclusions and future work

In this paper, we proposed a semantic-aware graph query method over knowledge graphs for star queries, which can provide matches that semantically similar to a given query graph Q rather than structurally similar matches. The semantic similarity of a match to Q is measured by a bounding match score that is computed online. By using bounds, we can efficiently prune the unpromising matches that have low semantically similarities earlier without evaluating all possible candidate matches, improving the efficiency of our approach. The experimental results on three real-world knowledge graphs confirm the effectiveness and efficiency of our approach. In the future, we shall take our solution to star queries on knowledge graphs as a building block to support: (1) more complex queries with different graph shapes, and (2) aggregate queries which focus on some statistical function (e.g., SUM, AVG, MAX/MIN, etc.) on the returned factoid answers.

CRediT authorship contribution statement

Yuxiang Wang: Conceptualization, Methodology, Investigation, Writing - original draft. **Xiaoliang Xu:** Supervision, Writing - review & editing. **Qifan Hong:** Software, Data curation, Validation. **Jiahui Jin:** Methodology, Writing - review & editing. **Tianxing Wu:** Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the National key Research & Development Program under grants 2017YFC0820503, the National Natural Science Foundation of China under grants 62072149 and 62072099; the Primary Research & Development Plan of Zhejiang Province under grant 2021C03156; the Public Welfare Research Program of Zhejiang under grant LGG19F020017; and the Natural Science Foundation of Zhejiang under grant LY19F030021.

References

[1] Y. Shi, X. Meng, F. Wang, Y. Gan, You can stop early with COLA: Online processing of aggregate queries in the cloud, in: CIKM, 2012, pp. 1223–1232.
[2] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P.N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, C. Bizer, DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia, Semant. Web 6 (2) (2015) 167–195.
[3] J. Hoffart, F.M. Suchanek, K. Berberich, G. Weikum, YAGO2: A spatially and temporally enhanced knowledge base from wikipedia, Artificial Intelligence 194 (2013) 28–61.
[4] K.D. Bollacker, C. Evans, P. Paritosh, T. Sturge, J. Taylor, Freebase: A collaboratively created graph database for structuring human knowledge, in: SIGMOD, 2008, pp. 1247–1250.
[5] X. Huang, J. Zhang, D. Li, P. Li, Knowledge graph embedding based question answering, in: WSDM, 2019, pp. 105–113.
[6] J. Jin, J. Luo, S. Khemmarat, et al., Querying web-scale knowledge graphs through effective pruning of search space, IEEE Trans. Parallel Distrib. Syst. 28 (8) (2017) 2342–2356.
[7] R.V. Guha, R. McCool, E. Miller, Semantic search, in: WWW, 2003, pp. 700–709.
[8] J. Jin, S. Khemmarat, L. Gao, J. Luo, Querying web-scale information networks through bounding matching scores, in: WWW, 2015, pp. 527–537.
[9] M. Arias, J.D. Fernández, M.A. Martínez-Prieto, P. de la Fuente, An empirical study of real-world sparql queries, arXiv:1103.5043.
[10] S. Yang, F. Han, Y. Wu, X. Yan, Fast top-k search in knowledge graphs, in: ICDE, 2016, pp. 990–1001.
[11] J. Cheng, J.X. Yu, B. Ding, S.Y. Philip, H. Wang, Fast graph pattern matching, in: ICDE, 2008, pp. 913–922.
[12] L. Zou, J. Mo, L. Chen, et al., gstore: Answering sparql queries via subgraph matching, Proc. VLDB Endow. 4 (8) (2011) 482–493.
[13] A. Khan, Y. Wu, C.C. Aggarwal, X. Yan, NeMa: Fast graph search with label similarity, Proc. VLDB Endow. 6 (3) (2013) 181–192.
[14] W. Zheng, H. Cheng, L. Zou, J.X. Yu, K. Zhao, Natural language question/answering: Let users talk with the knowledge graph, in: CIKM, 2017, pp. 217–226.
[15] L. Zou, L. Chen, M.T. Özsu, Distance-join: Pattern match query in a large graph database, Proc. VLDB Endow. 2 (1) (2009) 886–897.
[16] H. Tong, C. Faloutsos, B. Gallagher, T. Eliassi-Rad, Fast best-effort pattern matching in large attributed graphs, in: SIGKDD, 2007, pp. 737–746.
[17] J. Li, Y. Cao, S. Ma, Relaxing graph pattern matching with explanations, in: CIKM, 2017, pp. 1677–1686.
[18] A. Khan, N. Li, X. Yan, Z. Guan, S. Chakraborty, S. Tao, Neighborhood based fast graph search in large networks, in: SIGMOD, 2011, pp. 901–912.
[19] W. Zheng, L. Zou, X. Lian, D. Wang, D. Zhao, Graph similarity search with edit distance constraint in large graph databases, in: CIKM, 2013, pp. 1595–1600.
[20] Z. Zeng, A.K. Tung, J. Wang, J. Feng, L. Zhou, Comparing stars: On approximating graph edit distance, Proc. VLDB Endow. 2 (1) (2009) 25–36.
[21] S. Yang, Y. Wu, H. Sun, X. Yan, Schemaless and structureless graph querying, Proc. VLDB Endow. 7 (7) (2014) 565–576.
[22] Y. Wu, S. Yang, X. Yan, Ontology-based subgraph querying, in: ICDE, 2013, pp. 697–708.
[23] W. Zheng, L. Zou, W. Peng, X. Yan, S. Song, D. Zhao, Semantic SPARQL similarity search over RDF knowledge graphs, Proc. VLDB Endow. 9 (11) (2016) 840–851.
[24] M.H. Namaki, Y. Wu, X. Zhang, Gexp: Cost-aware graph exploration with keywords, in: SIGMOD, 2018, pp. 1729–1732.
[25] S. Han, L. Zou, J.X. Yu, D. Zhao, Keyword search on RDF graphs - A query graph assembly approach, in: CIKM, 2017, pp. 227–236.
[26] P. Peng, L. Zou, R. Guan, Accelerating partial evaluation in distributed sparql query evaluation, in: ICDE, 2019, pp. 112–123.
[27] X. Zhang, L. Zou, IMPROVE-QA: An interactive mechanism for rdf question/answering systems, in: SIGMOD, 2018, pp. 1753–1756.
[28] S. Hu, L. Zou, X. Zhang, A State-transition framework to answer complex questions over knowledge base, in: EMNLP, 2018, pp. 2098–2108.
[29] L. Zou, R. Huang, H. Wang, J.X. Yu, W. He, D. Zhao, Natural language question answering over RDF: A graph driven approach, in: SIGMOD, 2014, pp. 313–324.
[30] W. Zheng, J.X. Yu, L. Zou, H. Cheng, Question answering over knowledge graphs: Question understanding via template decomposition, Proc. VLDB Endow. 11 (11) (2018) 1373–1386.
[31] N. Nakashole, G. Weikum, F.M. Suchanek, PATTY: A taxonomy of relational patterns with semantic types, in: EMNLP-CoNLL, 2012, pp. 1135–1145.
[32] G.I. Diaz, A. Fokoue, M. Sadoghi, Embeds: Scalable, ontology-aware graph embeddings, in: EDBT, 2018, pp. 433–436.
[33] S. Guo, Q. Wang, B. Wang, L. Wang, L. Guo, Semantically smooth knowledge graph embedding, in: ACL, 2015, pp. 84–94.
[34] N. Nakashole, T. Tylenda, G. Weikum, Fine-grained semantic typing of emerging entities, in: ACL, 2013, pp. 1488–1497.
[35] J. Jin, S. Khemmarat, L. Gao, J. Luo, A distributed approach for top-k star queries on massive information networks, in: ICPADS, 2014, pp. 9–16.
[36] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, I. Stoica, Spark: Cluster computing with working sets, in: HotCloud, 2010.
[37] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, K. Tzoumas, Apache flink: Stream and batch processing in a single engine, Bull. IEEE Comput. Soc. Tech. Committee Data Eng. 36 (4) (2015) 28–38.
[38] Z. Wang, J. Zhang, J. Feng, Z. Chen, Knowledge graph embedding by translating on hyperplanes, in: AAAI, 2014, pp. 1112–1119.
[39] A. Bordes, N. Usunier, A. García-Durán, J. Weston, O. Yakhnenko, Translating embeddings for modeling multi-relational data, in: NIPS, 2013.
[40] G. Ji, S. He, L. Xu, K. Liu, J. Zhao, Knowledge graph embedding via dynamic mapping matrix, in: ACL, 2015, pp. 687–696.
[41] S. Yang, X. Yan, B. Zong, A. Khan, Towards effective partition management for large graphs, in: SIGMOD, 2012.
[42] Freebase-wikipedia mapping, http://downloads.dbpedia.org/2016-10/core-i18n/en/.
[43] C. Unger, C. Forascu, V. Lopez, A.-C.N. Ngomo, E. Cabrio, P. Cimiano, S. Walter, Question answering over linked data (qald-4), in: Working Notes for CLEF 2014 Conference, 2014, pp. 1172–1180.
[44] J. Berant, A. Chou, R. Frostig, P. Liang, Semantic parsing on freebase from question-answer pairs, in: EMNLP, 2013.
[45] T. Neumann, G. Weikum, Rdf-3x: A risc-style engine for rdf, Proc. VLDB Endow. 1 (1) (2008) 647–659.
[46] Neo4j, https://neo4j.com/.
[47] H. He, H. Wang, J. Yang, P.S. Yu, Blinks: Ranked keyword searches on graphs, in: SIGMOD, 2007, pp. 305–316.
[48] W. Fan, J. Li, S. Ma, H. Wang, Y. Wu, Graph homomorphism revisited for graph matching, Proc. VLDB Endow. 3 (1–2) (2010) 1161–1172.
[49] N. Jayaram, A. Khan, C. Li, X. Yan, R. Elmasri, Querying knowledge graphs by example entity tuples, IEEE Trans. Knowl. Data Eng. 27 (10) (2015) 2797–2811.
[50] J. Cohen, Statistical Power Analysis for the Behavioral Sciences, Routledge, 2013.
[51] J. Cheng, X. Zeng, J.X. Yu, Top-k graph pattern matching over large graphs, in: ICDE, 2013, pp. 1033–1044.
[52] D. Mottin, M. Lissandrini, Y. Velegrakis, T. Palpanas, Exemplar queries: A new way of searching, Proc. VLDB Endow. 25 (6) (2016) 741–765.
[53] M.H. Namaki, Q. Song, Y. Wu, et al., Answering why-questions by exemplars in attributed graphs, in: SIGMOD, 2019, pp. 1481–1498.
[54] Q. Song, M.H. Namaki, Y. Wu, Answering why-questions for subgraph queries in multi-attributed graphs, in: ICDE, 2019, 40–51.
[55] K.W. Church, P. Hanks, Word association norms, mutual information, and lexicography, Comput. Linguist. 16 (1) (1990) 22–29.
[56] R. Gligorov, W. ten Kate, Z. Aleksovski, F. van Harmelen, Using google distance to weight approximate ontology matches, in: WWW, 2007, pp. 767–776.
[57] T. Mikolov, I. Sutskever, K. Chen, et al., Distributed representations of words and phrases and their compositionality, in: NIPS, 2013, pp. 3111–3119.
[58] J. Pennington, R. Socher, C.D. Manning, Glove: Global vectors for word representation, in: EMNLP, 2014, pp. 1532–1543.
[59] G. Zhu, C.A. Iglesias, Computing semantic similarity of concepts in knowledge graphs, IEEE Trans. Knowl. Data Eng. 29 (1) (2017) 72–85.
[60] R. Rada, H. Mili, E. Bicknell, M. Blettner, Development and application of a metric on semantic nets, IEEE Trans. Syst. Man Cybern. 19 (1) (1989) 17–30.
[61] C. Leacock, M. Chodorow, Combining local context and wordnet similarity for word sense identification, WordNet: Electron. Lexical Database 49 (2) (1998) 265–283.
[62] J.J. Jiang, D.W. Conrath, Semantic similarity based on corpus statistics and lexical taxonomy, in: ROCLING, 1997, pp. 19–33.
[63] D. Lin, An information-theoretic definition of similarity, in: J.W. Shavlik (Ed.), ICML, 1998, pp. 296–304.
[64] P. Resnik, Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language, J. Artificial Intelligence Res. 11 (1999) 95–130.