Journal of Southeast University (English Edition) Vol. 40, No. 4, pp. 417 – 424 Dec. 2024 ISSN 1003—7985

Embedding-based approximate query for knowledge graph

Qiu Jingyi[1] Zhang Duxi[2] Song Aibo[1] Wang Honglin[3]
Zhang Tianbo[1] Jin Jiahui[1] Fang Xiaolin[1] Li Yaqi[1]

([1] School of Computer Science and Engineering, Southeast University, Nanjing 211189, China)
([2] Ningbo Power Supply Co., State Grid Zhejiang Electric Power Co., Ltd., Ningbo 315000, China)
([3] School of Artificial Intelligence, Nanjing University of Information Science and Technology, Nanjing 210044, China)

Abstract: To solve the low efficiency of approximate queries caused by the large sizes of the knowledge graphs in the real world, an embedding-based approximate query method is proposed. First, the nodes in the query graph are classified according to the degrees of approximation required for different types of nodes. This classification transforms the query problem into three constraints, from which approximate information is extracted. Second, candidates are generated by calculating the similarity between embeddings. Finally, a deep neural network model is designed, incorporating a loss function based on the high-dimensional ellipsoidal diffusion distance. This model identifies the distance between nodes using their embeddings and constructs a score function. k nodes are returned as the query results. The results show that the proposed method can return both exact results and approximate matching results. On datasets DBLP (DataBase systems and Logic Programming) and FUA-S (Flight USA Airports-Sparse), this method exhibits superior performance in terms of precision and recall, returning results in 0.10 and 0.03 s, respectively. This indicates greater efficiency compared to PathSim and other comparative methods.

Key words: approximate query; knowledge graph; embedding; deep neural network

DOI: 10.3969/j.issn.1003 – 7985.2024.04.011

A knowledge graph (KG) models entities and relations in the real world as nodes and edges within a graph[1]. Large-scale KGs have been constructed to represent factual information, such as YAGO and Wikipedia[2], which are widely used to advance research in areas such as question answering[3-4], recommendation[5], and visualization[6]. However, owing to the massive size of real-world KGs, there is a need to manage the time complexity gap associated with these graphs. KG query systems are crucial for high-performance applications and remain a prevailing topic in KG research. Therefore, designing an effective and efficient query system is a key issue in this research.

It is essential to understand that users prefer approximate queries, which enjoy increasing popularity in academia and industries, for the following two reasons: 1) Exact matching conditions are very strict, often resulting in prolonged search times. Approximate queries can enhance efficiency by narrowing the search space by pruning and applying thresholds; 2) Exact matching can return an empty set, failing to meet user expectations. Approximate queries can return both exact and approximate answers to enhance system generalization. The goal of this paper is to solve an approximate KG query problem: Given a query graph, find k exact or approximate matches in the KG and return the entities that users want.

Historically, there have been two main methods for KG queries: RDF/SPARQL-based modes[7-9] and subgraph isomorphism[10-11]. RDF is an information resource model that uses triples to describe items and their relations. SPARQL is a standard language for RDF queries, which has attracted significant attention for real-world applications. Nevertheless, users need to be familiar with the graph schemas, and RDF queries do not support approximate querying. Those disadvantages cause poor generalization and unbalanced overload on large-scale KGs. Subgraph isomorphism focuses on pruning to narrow the search space, but it remains an NP-complete problem, requiring substantial hardware and distributed calculation for large-scale graphs. Both methods respond by indices on graph structures and self-defined similarity measures for approximate queries, presenting clear challenges in enhancing query and index efficiency. Approximate subgraph matching benefits users as exact subgraph isomorphism is too restrictive to capture user needs[12-14]. Given that embedding provides a vectorization property for KG nodes and allows for a flexible design of score functions in queries, we propose a deep learning-based query model utilizing pre-existing embedding.

The contributions of this paper are listed in the following: First, we formalize query graphs and extract three limitations for approximate KG queries. Second, leveraging the powerful strength of the deep learning tech-

Received 2024-01-23, **Revised** 2024-05-16.

Biographies: Qiu Jingyi (1996—), female, Ph. D. candidate; Song Aibo (corresponding author), male, doctor, professor, absong@seu.edu.cn.

Foundation item: The State Grid Technology Project (No. 5108202340042A-1-1-ZN).

Citation: Qiu Jingyi, Zhang Duxi, Song Aibo, et al. Embedding-based approximate query for knowledge graph[J]. Journal of Southeast University (English Edition), 2024, 40(4): 417 – 424. DOI: 10.3969/j.issn.1003 – 7985.2024.04.011.

niques[15-17], we design a deep neural network (DNN) and loss function for selecting k approximate query results. This model can return both exact results and approximate results, ensuring that exact results are ranked higher than most approximate results within the top k returned results. Finally, we conduct experiments on real-world KGs with different densities. Experimental results demonstrate that our system outperforms other methods in terms of effectiveness and efficiency.

1 Preliminary

Definition 1 (knowledge and query graphs) Given a knowledge graph $G(V, E, T)$; V denotes the set of nodes representing entities; E denotes the set of edges; T is a function for mapping entities to their types. $T(v) \in \Lambda$ is the type of node v, where $v \in V$ and Λ is the infinite alphabet for all node types. Given a query graph $G_Q(V_Q, E_Q, T)$, V_Q denotes the set of nodes in G_Q, while E_Q denotes the set of edges in G_Q.

In a query graph, each node may contain varying amounts of information. Some nodes explicitly indicate the entities they represent, while others only specify their types. Users are primarily interested in matching results for specific subsets of nodes rather than all unmatched nodes. Based on these distinctions, nodes in the query graph are categorized into the following three types: 1) Confirmed nodes C_Q. The nodes correspond to keywords entered by users and have clear corresponding entities in the KG; that is, $\forall c \in C_Q$, $c \in V$. 2) Result nodes R_Q. Unknown nodes in G_Q whose matching entities meet user requirements. Typically, the type of these nodes is specified in the query graph, i.e., $\forall r \in R_Q$, $m(r) \in V$, and $T(m(r)) = T(r)$, where $m(\cdot)$ is the matching function for nodes in G_Q. In this paper, we assume that $|R_Q| = 1$ and R_Q is recorded as r_Q for simplification. We first discuss the case when $|R_Q| = 1$. 3) Other nodes O_Q. The nodes in G_Q are neither confirmed nodes nor result nodes. They lie on the paths between C_Q and R_Q or are neighbors of these nodes.

Classifying nodes in this manner and subsequently applying different levels of approximation to different nodes offer several advantages. First, it prevents over-approximation of crucial result nodes when generating candidate entities in approximate queries, ensuring that the returned results meet user requirements. Second, it allows for fuzzier matching on less critical nodes, reducing computational time spent on searching exact matches and improving query efficiency.

The purpose of this study is to execute approximate KG queries with better performance and higher efficiency. Users primarily seek the result node r_Q. We define a function ψ to quantify the approximation between a returned entity v and r_Q. Consequently, we articulate the problem as follows: Given a query graph $G_Q(V_Q, E_Q, T)$, the approximate query problem aims to find the k entities with the highest ψ values relative to the result node r_Q and return them, i.e., $V_r = \text{argmax}_{|V_r|=k, v \in V} \psi(v, G_Q)$, where V_r is the set of entities representing the output of the approximate query. In the context of the approximate query problem, ψ should consider the following conditions: 1) the returned entity v should have the same type as r_Q, i.e., $T(v) \equiv T(r_Q)$; 2) the distance between v and the set C_Q should be approximately equal to the distance between r_Q and C_Q; 3) for every other node $o \in O_Q$, o should either be adjacent to a confirmed node or result node, or lie on the path from a confirmed node to a result node.

Condition 1 is an inherent requirement that must be met. Condition 2 stems from the characteristic that the query graph is typically smaller and more connected compared to the whole KG. In this case, C_Q is always connected to r_Q and relatively close in distance. Therefore, we can use the distance between C_Q and r_Q as a measure of approximation. To allow some flexibility, we introduce a parameter ϵ to relax the condition. Condition 3 imposes further constraints on the query results based on O_Q. In general, O_Q is not as crucial as C_Q and r_Q. O_Q usually consists of neighbors of C_Q and r_Q or nodes in the paths connecting them. Nodes that do not meet these criteria provide minimal constraints on the query results. Moreover, owing to the small size of G_Q, it is unlikely to encounter such cases. Therefore, we do not consider nodes that have minimal constraints on the query results in Condition 3 for efficiency.

Based on these conditions, it is essential to apply varying approximation metrics to different node categories. However, quantifying the approximation and designing an approximate query function pose challenges. KG embedding can represent nodes or edges as low-dimensional vectors, effectively retaining the graph structure and semantic information, and is often used to measure similarity. Therefore, embedding can be considered for measuring query approximation. This paper seeks to utilize embedding to measure approximation. We define the embedding symbol as f, a vector with the dimension of z for low-dimensional representation.

2 Methodology

2.1 Query-based embedding

In this section, we discuss the process of generating embeddings suitable for approximate querying. Since the node types of both the result node and other nodes need to be considered during approximate queries, we not only embed the nodes in G but also their types. The embedding symbols for the nodes and types are denoted as $\boldsymbol{f}$ and $\boldsymbol{f}_t$, respectively. For each node v in KG G, $\boldsymbol{f}^v$ is a vector of dimension z denoting the embedding of node v. Similarly, for each type in G, $\boldsymbol{f}_t^{\lambda}$ is a vector of dimension z

denoting the embedding of type λ.

Based on the three conditions extracted from the query graph, type, connectivity, and distance, these components should be considered in the embedding process. Given that KGs are typically sparse graphs, the connectivity between nodes is enough for querying purposes. Therefore, we use the Euclidean distance between embeddings to measure both connectivity and distance. According to Condition 2, for $v_i, v_j \in V$, the smaller the distance between v_i and v_j, the smaller $\|\boldsymbol{f}^{v_i} - \boldsymbol{f}^{v_j}\|_2$ should be. Based on this property, we can use any graph embedding method that focuses on the connections between nodes. In this paper, we choose DeepWalk, the most commonly used and efficient embedding method. Existing KG embedding methods often require extensive training, which can be time-consuming and demanding on hardware resources. Moreover, when nodes and edges within the KG dynamically change, the embeddings must be retrained, posing a significant hardware challenge. Additionally, some KGs consist of encrypted data where nodes are represented by regular IDs rather than semantic entities. In such scenarios, DeepWalk is a suitable solution as it fulfills the requirements of our query model by serving as a time-efficient and convenient embedding method. It does not rely on expensive hardware or the inclusion of semantic entities, making it a practical choice. It is important to avoid selecting embedding methods specifically designed for heterogeneous graphs that consider node types. Instead, we embed the types in the KG separately. This is because when performing approximate queries, some types may require an exact match, and using embeddings designed for heterogeneous graphs can easily lead to type-matching errors. Instead, we construct a weighted graph specifically for types. In this graph, the types are treated as nodes, and the edge weights are proportional to the number of connecting edges between the entities corresponding to the types in G. We also choose DeepWalk for implementing type embedding, where the walking probabilities are proportional to the weights of the edges in the type graph.

2.2 Embedding-based approximate query model (EAQM)

In this section, we introduce a deep neural network model to return k best results from candidate generation and the DNN model. We first generate a candidate set by calculating the Euclidean distance between the embeddings of nodes with type $T(r_Q)$ and confirmed nodes.

To address Condition 2, we have developed a DNN-based approximate query model. This model constructs a neural network that takes the embedding vectors of two nodes as input and outputs a vector $\boldsymbol{\theta}$, where the dimension of vector $\boldsymbol{\theta}$ is $2z$, and z is the dimension of the embeddings $\boldsymbol{f}^v$ and $\boldsymbol{f}^{\lambda_t}$ for node v and type λ, respectively.

We posit that embedding vectors in high-dimensional space demonstrate an ellipsoidal shape. Fig. 1 (a) depicts an example of the relationship between embedding vectors in high-dimensional space, showcasing entities connected to v enclosed by different elliptical curves. This example, relying on DBLP, shows node v representing the conference "AAAI." The triangular nodes denote three papers which published at "AAAI," and the circular nodes denote the authors of those papers. The curve equation is represented by S. Nodes at a distance of one hop from entity v are enclosed by curve S_1, while nodes at a distance of two hops are situated between curves S_1 and S_2. Similarly, it can be inferred that node embedding at a distance of n hops from v should lie between curves S_{n-1} and S_n. Hence, we propose the following idea: if we can determine the equation for S, we can use the embedding to estimate the number of hops between two nodes. Furthermore, since the embedding in space expands outward in a certain pattern as the number of hops increases, different curves S_n share the same eccentricity. Given a chosen node v, we establish the equation for curve S_n as follows:

$$S_n(\boldsymbol{x}, v) = \frac{1}{n^2}(\boldsymbol{x} - [\boldsymbol{f}^v \quad \boldsymbol{f}_t^{\lambda^v}])^{\mathrm{T}}(\boldsymbol{x} - [\boldsymbol{f}^v \quad \boldsymbol{f}_t^{\lambda^v}])\boldsymbol{\theta} - 1 \tag{1}$$

where vector $\boldsymbol{x}$ is the input variable with a dimension of $2z$, and the symbol λ^v represents the type of node v. Vector $[\boldsymbol{f}^v \quad \boldsymbol{f}_t^{\lambda^v}]$ is formed by concatenating the node embedding and type embedding of entity v into a vector with a dimension of $2z$. Including type embedding is necessary as the distance between entities often depends on their

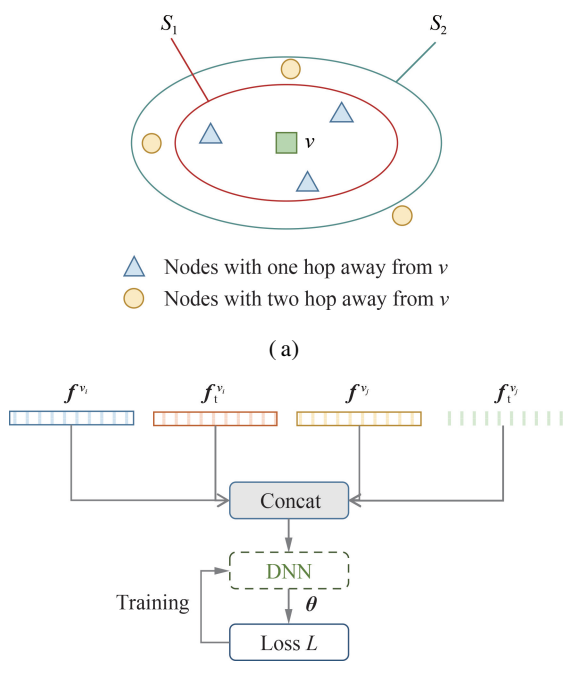

Fig. 1 Embedding-based approximate query model. (a) Embeddings in high-dimensional space; (b) EAQM structure

types. For example, in DBLP, the distance between an "Author" and a "Conference" is at least two hops.

Based on Eq. (1), if we set $\boldsymbol{x} = [\boldsymbol{f}^u \quad \boldsymbol{f}_t^{\lambda^u}]$, we can estimate the number of hops between entities u and v with given $\boldsymbol{\theta}$. Therefore, we design a DNN to generate $\boldsymbol{\theta}$. As illustrated in Fig. 1 (b), the training set will comprise randomly selected entity pairs (v_i, v_j). The input to the DNN will be $[\boldsymbol{f}^{v_i} \quad \boldsymbol{f}_t^{\lambda^{v_i}} \quad \boldsymbol{f}^{v_j} \quad \boldsymbol{f}_t^{\lambda^{v_j}}]$, i.e., the concatenated vectors of the entities. The output will be a vector $\boldsymbol{\theta}$. Subsequently, we devise the loss function as follows:

$$L(v_i, v_j) = L_1(v_i, v_j) + L_2(v_i, v_j) \tag{2}$$

$$L_1(v_i, v_j) = \mathrm{ReLU}(-S_{n-1}([\boldsymbol{f}^{v_i} \quad \boldsymbol{f}_t^{\lambda^{v_i}}], v_j)) \tag{3}$$

$$L_2(v_i, v_j) = \mathrm{ReLU}(S_n([\boldsymbol{f}^{v_i} \quad \boldsymbol{f}_t^{\lambda^{v_i}}], v_j)) \tag{4}$$

where S_{n-1} and S_n represent the $(n-1)$-th and the n-th ellipsoidal curves enclosing node v_j, respectively. As mentioned earlier, for a node v_i at a distance of n hops from v_j, its embedding (represented as $\boldsymbol{f}^{v_i}$) should lie between curves S_{n-1} and S_n. This means that $\boldsymbol{f}^{v_i}$ should be outside the curve S_{n-1} and inside the curve S_n. This condition can be expressed as $S_{n-1} > 0$ and $S_n < 0$, resulting in the loss $L_1(v_i, v_j) = L_2(v_i, v_j) = 0$. If $L_1(v_i, v_j) > 0$, it indicates that $\boldsymbol{f}^{v_i}$ is inside the curve S_{n-1}, while $L_2(v_i, v_j) > 0$ indicates that $\boldsymbol{f}^{v_i}$ is outside the curve S_n. Both cases signify that the trained curve fails to accurately identify the distance between v_i and v_j, requiring further training. It is important to note that there is a negative sign before S_{n-1} in Eq. (3). This negative sign ensures that the loss needs to be minimized, but the position of $\boldsymbol{f}^{v_i}$ with respect to S_{n-1} and S_n is opposite, i.e., $\boldsymbol{f}^{v_i}$ should be outside S_{n-1} and inside S_n. After obtaining $\boldsymbol{\theta}$ through training, we can design ψ as $\psi(v, G_Q) = \sum_{c \in C_Q, v \in V} -L(c, v)$ and return the nodes with the k-th maximum ψ score.

It is not necessary to design an additional model to limit the type of other nodes in Condition 3. In dense graphs, the number of nodes and edges is greater than the number of node types, which causes a large number of meta-paths within the graph structure. For example, in DBLP, the type of nodes that are two hops between "Conference" and "Author" can only be "Paper." In sparse graphs, which have a low density of connections, many node pairs have only one path between them, or nodes have only one neighbor. When the distance is confirmed, the path or neighbor is unique, and the corresponding other nodes are easily matched. Therefore, an approximate result can be searched on Condition 2 without restricting the type of other nodes.

2.3 Optimization

2.3.1 Negative sampling

In a KG with high density and complex relations, it is essential to set appropriate training data. For instance, the loss with $n = 2$ and loss with $n = 6$ may slightly differ when the graph is connected, and its diameter is 7. This can lead to poor performance because a large number of samples could be trained in the range of S_7. Therefore, negative sampling is indispensable to identify between closer and further nodes as they are always connected. We propose setting n as a value greater than the diameter of the graph when $n \geqslant n_M$, where n_M is a threshold.

2.3.2 Multi-Node Query

The above model is built based on the assumption that $|R_Q| = 1$. However, it becomes inefficient to utilize EAQM on each r_Q and enumerate different combinations of $r \in R_Q$ when $|R_Q| > 1$. As a result, we use a greedy algorithm to combine different $r \in R_Q$ for discovering k results.

3 Experiments

3.1 Settings

We use two real-world KGs. DBLP[18] is a bibliographic KG, while Flight-USA-Airports is a cryptographic graph[19]. To compare the performance of datasets with different densities, we extract a part of the edges, denoted as FUA-S. Table 1 lists the statistics of the two datasets, where density δ is defined as $\delta = 2|E|/|V|$.

Table 1 Dataset statistics

Dataset	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert \Lambda\rvert$	δ
DBLP	37 791	170 794	4	9.038 9
FUA-S	1 190	2 358	4	3.963 0

Query graphs are randomly extracted from the main graphs. We assess the effectiveness of EAQM by manually evaluating the approximate query results using the queries listed in Table 2, containing chain-shaped and star-shaped queries. To evaluate response times, we randomly

Table 2 Queries in experiments

Query	Dataset	Shape	Queries
Q1	DBLP	Chain	Find an authorwho published a paper in cooperation with Thomas P. Minka, and one of his papers contains the term "expectation."
Q2	DBLP	Chain	Find a termthat appeared with the terms "piazza" and "are" in two papers, respectively.
Q3	DBLP	Star	Query an authorwho published two papers with Nirmalie Wiratunga and Stewart Massie, respectively, and published a paper about "folding."
Q4	FUA-S	Chain	Find an entity with type "3"that is two hops away from entities "12134" and "13930," and the types of the middle nodes on the two paths are both "3."
Q5	FUA-S	Chain	Query an entity with type "2," that is two hops away from entities "14273" and "14119," and the types of the middle nodes on two paths are both "2."
Q6	FUA-S	Star	Find an entity with type "1"that is two hops away from the entity "12822," "13434," and "14627." The types of middle nodes on the three paths are "1," "1," and "0," respectively.

select 1 000 subgraphs that have the same structure as the queries in Table 2 from each KG. We compare our approach with several baseline methods, including PathSim, Nema[20], VF2[21], SGQ, and VEQ[22]. All experiments are conducted using Python 3.7.

3.2 Evaluating effectiveness

We implement experiments for testing the effectiveness of EAQM. The precision of exact and approximate queries is plotted for different datasets with $k = 1, 5, 10, 30$ in Fig. 2. Figs. 2 (a) and (b) depict the precision of exact queries and approximate queries performed by EAQM on the DBLP dataset. This precision represents the rate of results that exactly match the query graph and approximately match the query graph in the returned results. Additionally, Figs. 2 (c) and (d) showcase the precision of exact queries and approximate queries by EAQM on the FUA-S dataset. The approximate query results ($k = 10$) containing precision (π) and recall (ρ) are shown in Table 3.

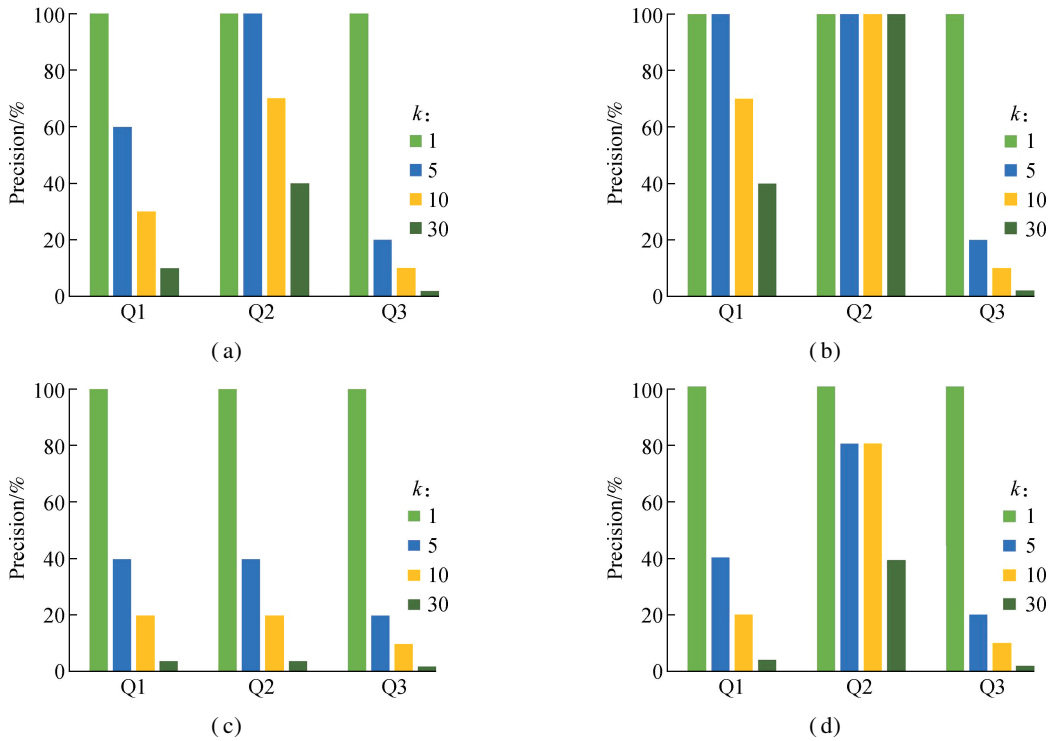

Fig. 2 Query effectiveness. (a) DBLP exact query; (b) DBLP approximate query; (c) FUA-S exact query; (d) FUA-S approximate query

Table 3 Query effectiveness on DBLP and FUA-S

Method	Q1		Q2		Q3		Q4		Q5		Q6	
	π	ρ	π	ρ	π	ρ	π	ρ	π	ρ	π	ρ
PathSim			0.80	0.03					0.10	0.91		
Nema	1.00	0.29	1.00	0.03	0.10	1.00	0.00	0.00	0.10	0.91	0.00	0.00
VF2	0.30	0.09	≤1.00	≤0.03	≤0.10	≤1.00	0.10	0.33	0.10	0.91	0.10	1.00
SGQ	0.30	0.09	≤0.20	≤0.01	≤0.10	≤1.00	0.10	0.33	0.20	0.18	0.10	1.00
VEQ	0.30	0.09	1.00	0.03	0.10	1.00	0.10	0.33	0.10	0.91	0.10	1.00
EAQM	0.70	0.20	1.00	0.03	0.10	1.00	0.20	0.67	0.80	0.73	0.10	1.00

In Fig. 2, it is evident that precision declines as k increases. This occurs because the number of correct answers does not exceed k. This is a common phenomenon given that there are often only one or two correct answers in k results. The precision for chain-shaped queries is higher than that of star-shaped queries because the number of confirmed nodes in a star query strengthens the conditions. For example, only one correct answer is returned on $k = 1$ for queries Q3 and Q6, reflecting a similar pattern in Fig. 2. The FUA-S dataset reveals a lower precision owing its sparsity, which causes a lower number of correct answers. As constraints are relaxed, the precision of approximate queries tends to be higher than that of exact queries. Table 3 shows that EAQM outperforms other baselines on DBLP and FUA-S in most cases, demonstrating its advantages on KGs with different densities. While Nema has a slight advantage over EAQM in terms of precision on the DBLP dataset, it exhibits response times that are significantly longer, by tens of times, than those of EAQM during efficiency experiments. This makes EAQM a more suitable method overall when both effectiveness and efficiency are considered for approximate KG

queries. It is important to note that PathSim and VF2 have precision values less than 1, indicating lower accuracy. By contrast, the remaining baselines support approximate queries, resulting in higher precision values.

3.3 Evaluating efficiency

This experiment evaluates the efficiency of EAQM, as shown in Fig. 3, where query graphs are randomly selected to match the shapes of the query graphs listed in Table 2. For instance, Q1-shaped query graphs have the same structure as Q1. It is evident that response time increases with the value of k. The response time of EAQM ranges from 0.04 to 0.12 s on DBLP and from 0.004 to 0.08 s on FUA-S. Queries on DBLP consistently exhibit longer response times owing to the larger dataset size and density. Star-shaped queries, such as Q3-shaped queries and Q6-shaped queries, take more time to compute distances because they involve more confirmed nodes. However, dataset size and density significantly impact efficiency.

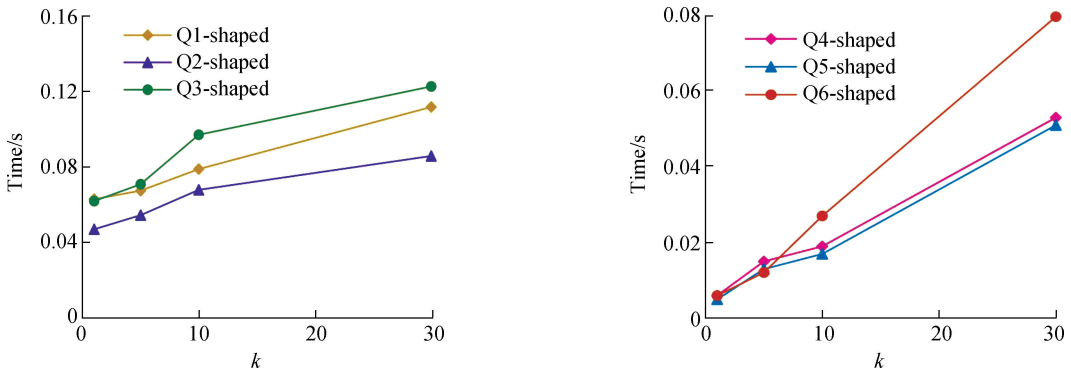

Fig. 3 Query time of approximate queries. (a) Query time on DBLP; (b) Query time on FUA-S

Tables 4 and 5 present the response time of different methods. The response time of VF2 is not included because VF2 is unable to respond within a reasonable time frame, often 10 d. These tables show that EAQM consistently has the shortest response time among the tested methods. For example, the response time of EAQM for Q2-shaped queries at $k = 10$ is 32.35% longer than at $k = 1$, indicating that the value of k is influential on EAQM performance. Compared to Q5-shaped queries, the response time for Q2-shaped queries increases by 75% when $k = 10$, highlighting that the response time of EAQM will increase on larger KGs. Regarding query graph size, EAQM presents longer response times for star-shaped queries. For instance, the response time for Q1-shaped queries at $k = 10$ is 18.56% longer than that for Q3-shaped queries. This indicates that although EAQM is affected by graph density, dataset size, and query graph size, it outperforms other baselines in terms of efficiency. EAQM can return query results in less than 0.10 s on DBLP and 0.03 s on FUA-S.

Table 4 Response time of DBLP approximate queries s

Method	Q1-shaped			Q2-shaped			Q3-shaped		
	$k=1$	$k=5$	$k=10$	$k=1$	$k=5$	$k=10$	$k=1$	$k=5$	$k=10$
PathSim				0.669	0.712	0.741			
Nema	136.3	136.3	136.2	133.4	133.3	133.4	127.4	127.5	127.5
SGQ	10.85	12.47	11.47	16.11	22.89	21.22	6.02	6.60	5.98
VEQ	860.0	881.6	862.6	730.0	786.0	744.5	988.1	983.8	988.8
EAQM	0.062	0.067	0.079	0.046	0.054	0.068	0.062	0.071	0.097

Table 5 Response time of FUA-S approximate queries s

Method	Q4-shaped			Q5-shaped			Q6-shaped		
	$k=1$	$k=5$	$k=10$	$k=1$	$k=5$	$k=10$	$k=1$	$k=5$	$k=10$
PathSim				0.004	0.004	0.004			
Nema	0.550	0.548	0.557	0.261	0.263	0.262	0.305	0.316	0.326
VF2	0.021	0.018	0.020	0.037	0.043	0.040	1.647	1.627	1.624
SGQ	0.213	0.211	0.215	0.047	0.049	0.049	0.143	0.130	0.131
VEQ	0.058	0.058	0.058	0.105	0.105	0.107	0.236	0.242	0.242
EAQM	0.006	0.015	0.019	0.005	0.013	0.017	0.006	0.012	0.027

4 Conclusions

1) In this paper, we propose EAQM, an embedding-based framework designed to implement approximate queries on KGs. We first decompose the query graph into three query constraints to quantify the approximation of different nodes. EAQM then utilizes these constraints within an embedding-based framework to implement the approximate query. This model analyzes the embedding distances in high-dimensional space and employs a loss function in a DNN for training and scoring results.

2) Experimental results show that EAQM improves the effectiveness of both dense and sparse graphs. Additionally, EAQM returns query results in less than 0.10 s for DBLP and 0.03 s for FUA-S.

3) In the future we aim to enhance the compatibility of embedding-based KG queries for more complex query graphs.

References

[1] Su Y, Yang S Q, Sun H, et al. Exploiting relevance feedback in knowledge graph search[C]// *Proceedings of the* 21*st ACM Knowledge Discovery and Data Mining*. Sydney, Australia, 2015: 1135 − 1144. DOI: 10.1145/2783258.2783320.

[2] Suchanek F M, Kasneci G, Weikum G. YAGO: A large ontology from Wikipedia and WordNet[J]. *Journal of Web Semantics*, 2008, **6**(3): 203 − 217. DOI: 10.1016/j.websem.2008.06.001.

[3] Liu L H, Chen Y Z, Das M, et al. Knowledge graph question answering with ambiguous query[C]// *Proceedings of the* 32*nd International World Wide Web Conferences*. Austin, TX, USA, 2023: 2477 − 2486. DOI: 10.1145/3543507.3583316.

[4] Li H Y, Zhao M, Yu W Q. A multi-attention RNN-based relation linking approach for question answering over knowledge base [J]. *Journal of Southeast University* (*English Version*), 2020, **36**(4): 385 − 392. DOI: 10.3969/j.issn.1003-7985.2020.04.003.

[5] Zhou K, Zhao W X, Bian S Q, et al. Improving conversational recommender systems via knowledge graph based semantic fusion [C]// *Proceedings of the* 26*th ACM Knowledge Discovery and Data Mining*. Virtual Event, USA, 2020: 1006 − 1014. DOI: 10.1145/3394486.3403143.

[6] Wei J Q, Han S, Zou L. VISION-KG: Topic-centric visualization system for summarizing knowledge graph [C]// *Proceedings of the* 13*th International Conference on Web Search and Data Mining*. Houston, TX, USA, 2020: 857 − 860. DOI: 10.1145/3336191.3371863.

[7] Arenas M, Cuenca Grau B, Kharlamov E, et al. Faceted search over RDF-based knowledge graphs[J]. *Journal of Web Semantics*, 2016, **37**/**38**: 55 − 74. DOI: 10.1016/j.websem.2015.12.002.

[8] Mailis T, Kotidis Y, Nikolopoulos V, et al. An efficient index for RDF query containment[C]//*Proceedings of the* 2019 *International Conference on Management of Data*. Amsterdam, the Netherlands, 2019: 1499 − 1516. DOI: 10.1145/3299869.3319864.

[9] Abdelaziz I, Harbi R, Khayyat Z, et al. A survey and experimental comparison of distributed SPARQL engines for very large RDF data[J]. *Proceedings of the VLDB Endowment*, 2017, **10**(13): 2049 − 2060. DOI: 10.14778/3151106.3151109.

[10] Zeng J, U L H, Yan X, et al. Fast core-based top-k frequent pattern discovery in knowledge graphs[C]// *Proceedings of the* 37*th IEEE International Conference on Data Engineering*. Chania, Greece, 2021: 936 − 947. DOI: 10.1109/ICDE51399.2021.00086.

[11] Sun Y Z, Han J W, Yan X F, et al. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks[C]// *Proceedings of the VLDB Endowment*. Seattle, WA, USA, 2011: 992 − 1003. DOI: 10.14778/3402707.3402736.

[12] Yang S Q, Han F Q, Wu Y H, et al. Fast top-k search in knowledge graphs[C]// *Proceedings of the* 32*nd IEEE International Conference on Data Engineering*. Helsinki, Finland, 2016: 990 − 1001. DOI: 10.1109/ICDE.2016.7498307.

[13] Wang Y X, Khan A, Wu T X, et al. Semantic guided and response times bounded top-k similarity search over knowledge graphs[C]// *Proceedings of the* 36*th IEEE International Conference on Data Engineering*. Dallas, TX, USA, 2020: 445 − 456. DOI: 10.1109/ICDE48307.2020.00045.

[14] Qin Z Y, Bai Y S, Sun Y Z. GHashing: Semantic graph hashing for approximate similarity search in graph databases[C]// *Proceedings of the* 26*th ACM Knowledge Discovery and Data Mining*. Virtual Event, USA, 2020: 2062 − 2072. DOI: 10.1145/3394486.3403257.

[15] Biao Y, Lin G Y, Zhang W G. Adaptive topology learning of camera network across non-overlapping views[J]. *Journal of Southeast University* (*English Version*), 2015, **31**(1): 61 − 66. DOI: 10.3969/j.issn.1003-7985.2015.01.011.

[16] Zhao N N, Jiang R. Poisoning attack detection scheme based on data integrity sampling audit algorithm in neural network[J]. *Journal of Southeast University* (*English Version*), 2023, **39**(3): 314 − 322. DOI: 10.3969/j.issn.1003-7985.2023.03.012.

[17] Wang Y H, He J Z, Zhang M Z, et al. Concrete crack identification in complex environments based on SSD and pruning neural network[J]. *Journal of Southeast University* (*English Version*), 2023, **39**(4): 393 − 399. DOI: 10.3969/j.issn.1003-7985.2023.04.008.

[18] Han M, Kim H, Gu G, et al. Efficient subgraph matching: Harmonizing dynamic programming, adaptive matching order, and failing set together[C]// *Proceedings of the* 2019 *ACM Conference on Management of Data*. Amsterdam, the Netherlands, 2019: 1429 − 1446. DOI: 10.1145/3299869.3319880.

[19] Chen S W. Graph embedding [EB/OL]. (2022)[2023-12-08]. https://github.com/shenweichen/GraphEmbedding.

[20] Khan A, Wu Y H, Aggarwal C C, et al. Nema: Fast graph search with label similarity[C]// *Proceedings of the VLDB Endowment*. Riva del Garda, Italy, 2013: 181 − 192. DOI: 10.14778/2535569.2448952.

[21] Cordella L P, Foggia P, Sansone C, et al. A (sub) graph isomorphism algorithm for matching large graphs [J]. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2004, **26**(10): 1367 – 1372. DOI: 10.1109/TPAMI.2004.75.

[22] Kim H J, Choi Y Y, Park K S, et al. Versatile equivalences: Speeding up subgraph query processing and subgraph matching[C]// *Proceedings of the* 2021 *ACM Conference on Management of Data*. Virtual Event, China, 2021: 925 – 937. DOI: 10.1145/3448016.3457265.

基于嵌入的知识图谱近似查询

邱敬怡[1] 章杜锡[2] 宋爱波[1] 王红林[3] 张添博[1] 金嘉晖[1] 方效林[1] 李雅琦[1]

([1]东南大学计算机科学与工程学院,南京 211189)
([2]国网浙江省电力有限公司宁波供电公司,宁波 315000)
([3]南京信息工程大学人工智能学院,南京 210044)

摘要:为解决现实生活中知识图谱规模庞大而导致近似查询效率低下的问题,提出了一种基于嵌入的知识图谱近似查询方法. 首先对查询图中的节点进行分类,根据不同类型节点所需的近似程度,将查询问题转化为 3 个约束条件,提取近似信息. 然后,通过计算嵌入之间的相似度,生成候选集. 最后,设计了一个深度神经网络模型和基于高维椭球形扩散距离的损失函数,根据嵌入判断节点间距离,并构建打分函数,返回 k 个节点作为查询结果. 结果表明,所提方法可以同时返回精确匹配结果和近似匹配结果. 该方法在 DBLP 和 FUA-S 两个数据集上均获得了最高的准确率和召回率,且可分别在 0.10 和 0.03 s 内返回结果,效率高于 PathSim 等对比方法.

关键词:近似查询;知识图谱;嵌入;深度神经网络
中图分类号:TP392