Diversified Top-*k* Keyword Query Interpretation on Knowledge Graphs

Ying Wang, Ming Zhong*, Yuanyuan Zhu, Xuhui Li, and Tieyun Qian

State Key Laboratory of Software Engineering
Wuhan University, Wuhan 430072, China
{wysklse,clock,yyzhu,lixuhui,qty}@whu.edu.cn

Abstract. Exploring a knowledge graph through keyword queries to discover meaningful patterns has been studied in many scenarios recently. From the perspective of query understanding, it aims to find a number of specific interpretations for ambiguous keyword queries. With the assistance of interpretation, the users can actively reduce the search space and get more relevant results.

In this paper, we propose a novel diversified top-*k* keyword query interpretation approach on knowledge graphs. Our approach focuses on reducing the redundancy of returned results, namely, enriching the semantics covered by the results. In detail, we 1) formulate a diversified top-*k* search problem on a schema graph of knowledge graph for keyword query interpretation; 2) define an effective similarity measure to evaluate the semantic similarity between search results; 3) present an efficient search algorithm that guarantees to return the exact top-*k* results and minimize the calculation of similarity, and 4) propose effective pruning strategies to optimize the search algorithm. The experimental results show that our approach improves the diversity of top-*k* results significantly from the perspectives of both statistics and human cognition. Furthermore, with very limited loss of result precision, our optimization methods can improve the search efficiency greatly.

Keywords: Diversification; Keyword Query Interpretation; Top-k Search; Knowledge Graph

1 Introduction

1.1 Motivation

Recently, keyword search is well recognized as a popular and effective approach to acquire knowledge from the large-scale knowledge graphs, such as DBPedia [3], Yago [8], Freebase [4], Probase [11], etc. However, keyword search suffers from a trade-off between expressiveness and ease-of-use, which results in the ambiguities of users' information needs. Therefore, keyword query interpretation is proposed for predicating the most relevant query semantics to users' information needs. As a result, the

* Corresponding author.

users can still issue keyword queries initially, and then, they will formulate more expressive and relevant queries from the returned intermediate results, thereby narrowing the search space and improving the quality of final results.

The existing keyword query interpretation approaches [12-15] mainly focus on improving the relevance of results and the efficiency. There is still a lack of discussion of an important property of interpretation results, namely, diversity. Unfortunately, according to our observation on real-world knowledge graphs, only considering the relevance of results often leads to lots of similar results which are redundant to the user and also may not reach the different user's intention. That is because the most relevant results mostly have the same nodes, edges and even structures that are preferred by scoring functions. Let us consider the following example.

Example 1. Given a keyword query "London, Paris" on DBPedia, the top-4 relevant results with and without diversification with respect to some specific scoring function and similarity function are shown in Figure 1. Each tree is an interpreted result, and can be seen as a graph query actually. Their leaf nodes contain the two keywords respectively. Moreover, the nodes in different colors represent different classes in DBPedia. Intuitively, the four trees on the top (without diversification) are very similar to each other. They are all rooted at the same class node, and almost share the same classes. In contrast, the four trees on the bottom (with diversification) are quite different from each other, and demonstrate various relationships between "London" and "Paris", such as biological relationship between two plants, soccer player who served in two clubs, or geographic relationship between two locations.

In order to make the interpretations of a keyword query meet the various information needs of different users, we need to diversify the results.

1.2 Related Work

Keyword Query Interpretation. It is a popular research topic in the communities of semantic web, information retrieval and database. There are generally two kinds of methodologies. The first is to map the keyword query to semantic patterns precomputed from the graph. For example, Pound and etc. [5,6] generates a list of possible elements in the knowledge graph for each keyword phrase in the query, sorts the elements by syntactic similarity, and lastly processes the sorted lists by using a variation of Threshold Algorithm. The second is to search the relationships between keywords in the schema graph and compose patterns. For example, Tran and etc. [9,10] models the interpretation results as subgraphs of the schema graph that connect all keywords in the query. The top-k results ranked with respect to relevance are returned. Overall, the current research works try to find the best individual results but not the best result set.

Diversified Top-k Search. It has been studied in many applications recently, such as [16-18]. The most straightforward solutions (e.g., [1,2]) assume the rankings of all search results are known in advance, and diversified search algorithms are given to output k results with respect to score and similarity. However, graph search usually returns a very large number of results, so that it is not feasible to rank all results in the context of this paper. Then, Qin et al. [7] propose general frameworks to handle the

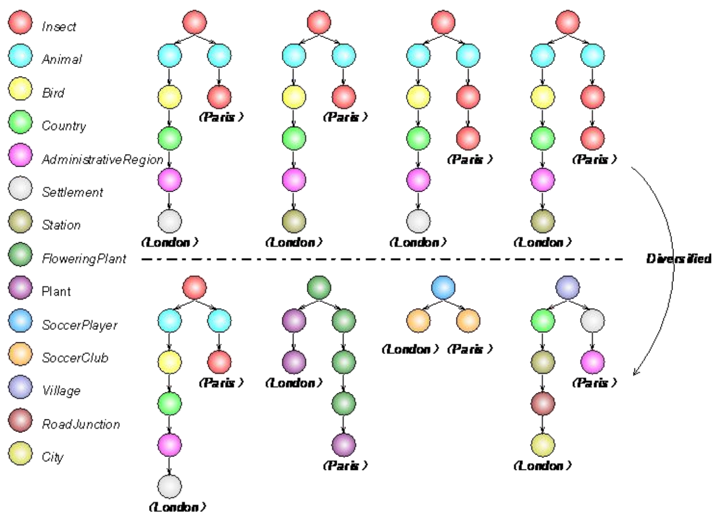


Fig. 1. An example of top-*k* interpretations with and without diversification.

diversified top-k search problem, which can stop early without exploring all search results. But it cannot be directly applied to solve our keyword query interpretation problem due to the different problem definition.

1.3 Our Contributions

In this paper, we propose a novel diversified top-k keyword query interpretation approach to address the redundancy problem of interpreted results, so that the top-k results could cover richer semantics and satisfy different users. Our approach follows the line of [10] to search for semantic patterns on a schema graph of knowledge graph for a given keyword query, and tries to return k patterns that have the most possible high scores and meanwhile are not similar to each other.

Our main contributions are as follows.

- We reasonably formulate a diversified top-k search problem on a schema graph of knowledge graph for keyword query interpretation.
- We define an effective similarity measure to evaluate the semantic similarity between search results.
- We present an efficient top-*k* algorithm to address the problem. The algorithm can guarantee to return the exact top-*k* results, and minimize the calculation of similarity. Moreover, two heuristic pruning strategies are proposed to improve the search efficiency with possible losses of exact top-*k* results.
- We perform experiments on DBPedia. The results show that our approach improves the diversity of top-*k* results effectively from the perspective of both statistics and human cognition. Meanwhile, with a small loss (on average 3%) of result precision as trade-off, the pruning strategies reduce the response time dramatically.

The rest of this paper is organized as follows. Section 2 introduces the background and problem definition. We present our similarity measure and search algorithm in Section 3 and 4 respectively. Section 5 shows the experimental results. Lastly, we conclude in Section 6.

2 Preliminaries

2.1 Data and Query Models

Knowledge Graph. Without loss of generality, we simply model a knowledge graph as $G = (V, E, H, K)$, where V is a set of nodes, $E = V \times V$ is a set of edges between the nodes, H is a set of classes of nodes, and K is a set of keywords contained by the nodes. For each node $v \in V$, we denote by $class(v) \in H$ and $keyword(v) \subset K$ its class and set of keywords, respectively.

Schema Graph. Given a data graph G, let its schema graph $G_s = (V_s, E_s)$, where V_s is a set of class nodes, one for each class in H, and $E_s = V_s \times V_s$ is a set of edges between the class nodes. For a node $v \in V$ and a class node $v_s \in V_s$, $v \in v_s$ if and only if $class(v) \subset class(v_s)$, so that V_s is actually a partition of V with respect to the classes of nodes. Moreover, we denote by $keyword(v_s) \subset K$ the union of keywords contained by all nodes of v_s, namely, $keyword(v_s) = \bigcup_{v \in v_s} keyword(v)$. Lastly, an edge $(v_s, u_s) \in E_s$ if and only if there exist nodes $v \in v_s$ and $u \in u_s$ such that $(v, u) \in E$.

Keyword Query. A keyword query $Q \subseteq K$ is simply a set of keywords. In particular, we call a class node $v_s \in V_s$ keyword node if and only if there exists a keyword $q \in Q$ such that $q \in keyword(v_s)$.

Pattern Tree. Given a schema graph G_s, we interpret a keyword query Q to a set of pattern trees, where each node is an embedding of a class node on the schema graph and the leaf nodes are the keyword nodes of each keyword in Q, the root node is the same class node in the end of each path. Formally, for a pattern tree $T = (V_t, E_t)$, there is a mapping $f: V_t \mapsto V_s$. For each edge $(v_t, u_t) \in E_t$ we have $(f(v_t), f(u_t)) \in E_s$. Moreover, a pattern tree is comprised of $|Q|$ keyword-to-root paths called *search paths*. For a search path $P_q = v_{t1}/ \ldots /v_{tn}$ outgoing from the keyword $q \in Q$, we have $q \in keyword(f(v_{t1}))$.

2.2 Scoring Metrics

Interpreting a keyword query by searching on a schema graph usually returns a huge number of results due to the explosive combinations of nodes and edges, most of which are irrelevant. To address the problem, existing keyword query interpretation works have considered a variety of scoring metrics in order to evaluate how well an interpreted result matches the user's information needs. For example, some widely-used metrics are introduced as follows.

Compactness. In the context of keyword search over graphs, a basic assumption is that more tightly-connected nodes comprise a more meaningful answer. For example, the answer trees with less edges or levels are ranked higher. Thus, the pattern trees comprised of shorter search paths are preferred.

Popularity. For each node in the schema graph, we can compute a popularity score by means like PageRank. Like web page ranking, pattern trees that contain more popular nodes should be ranked higher.

Relevance. As a common measure in IR, TF/IDF can also be used to evaluate the relevance of pattern trees to the given keywords. For a keyword node, we can compute an initial relevance score.

For a search path P, we denote by $score(P)$ its score that incorporates path length, popularity of nodes on the path, relevance of keyword node, and even other metrics. The scoring function is featured by 1) the higher the score, the better the search path, and 2) for a search path $P' = P/\ldots/v$ extended from another search path P, we have $score(P') < score(P)$. The details of scoring function are omitted because it is not the focus of this paper.

Based on the scoring function of search path, we evaluate the score of a pattern tree as follows.

$$score(T) = \sum_{P\in T} score(P) \tag{1}$$

Obviously, the scoring function of pattern tree is monotonic in the context of search algorithms, thereby facilitating efficient top-k search algorithm and effective search path pruning during the search (see Section 4).

2.3 Problem

Consider a set of results $S = \{T_1, T_2, \ldots\}$. For each $T_i \in S$, the score of T_i is denoted as $score(T_i)$. Given two results $T_i, T_j \in S$, the similarity between them is denoted as $sim(T_i, T_j)$ with $0 \leq sim(T_i, T_j) \leq 1$. The parameter τ is defined as the threshold to determine whether two pattern trees are similar. When $\tau \leq sim(T_i, T_j) \leq 1$, T_i is similar to T_j, and vice versa. The definition of the diversified top-*k* results is as follows.

Definition 1 (Diversified Top-*k* Results). Given an integer k with $1 \leq k \leq |S|$, the diversified top-*k* results of S is S_k such that

1. $S_k \subseteq S$ and $|S_k| \leq k$;
2. for any two results $T_i, T_j \in S_k$ with $T_i \neq T_j$, we have $sim\left(T_i, T_j\right) < \tau$, namely, they are not similar to each other;
3. for each result $T_i \in S_k$, if $T_j \in S$ and $score\left(T_j\right) > score(T_i)$, we have either $T_j \in S_k$ or $\exists T_l \in S_k$ such that $score(T_l) > score(T_j)$ and $\tau \leq sim(T_l, T_j) \leq 1$.

Example 2. Figure 2 shows an example result set S= $\{T_1, T_2, \ldots, T_7\}$ and the corresponding S_k with *k* = 4. Each node in the figure represents a pattern tree, and the edges indicate that the two connected pattern trees are similar. Moreover, the labels of nodes are their scores. The diversified top-*k* results S_k includes four nodes T_3, T_1, T_6 and T_7, which are sorted by their scores. For T_2 and T_5, their scores are just too low. For T_4, although its score is higher than T_6 and T_7, it is not qualified for top-*k* because

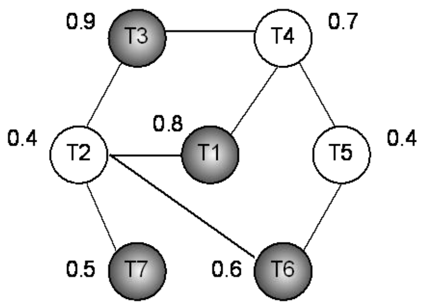


Fig. 2. An example of diversified top-k results.

it is similar to T_3 and T_1. Thus, we diversify the top-k results by abandoning T_4 and importing T_6 and T_7.

With Definition 1, the problem addressed in this paper is defined as follows.

Problem. Given a schema graph G_s and a keyword query Q, let S be the set of pattern trees interpreted from Q on G_s, compute the diversified top-k results S_k of S without generating the whole S.

3 Similarity Measure

In order to diversify the pattern trees, we need to measure the similarity between them.

Given a keyword query Q, each pattern tree returned will have $|Q|$ search paths outgoing from each keyword in Q. We firstly abstract each search path as a feature vector. Each feature represents the frequency of a class in the path. Given a schema graph G_s, the dimensionality of feature vector is the number of classes, namely, $|V_s|$. Let $F(P)$ = (I_1, I_2, …, $I_{|V_s|}$) be the feature vector of a search path P. We calculate I_i as follows.

$$I_i = \frac{n_i}{|P|} \tag{2}$$

where $|P|$ is the total number of nodes on the path P and n_i is the number of nodes that belong to the i-th class node of V_s on the path.

Let P_{q1} and P_{q2} be two search paths outgoing from a keyword q in two different pattern trees. We use the angle cosine formula to calculate the similarity of the two paths.

$$sim(P_{q1},\ P_{q2}) = \frac{F(P_{q1})\cdot F(P_{q2})}{|F(P_{q1})|*|F(P_{q2})|} \tag{3}$$

Lastly, the similarity of two pattern trees is the mean of the similarity of all their corresponding paths.

$$\mathrm{sim}(T_1, T_2) = \frac{\sum_{P_{q1}\in T_1, P_{q2}\in T_2} sim(P_{q1}, P_{q2})}{|Q|} \tag{4}$$

Intuitively, our similarity function measures how redundant the classes in two pattern trees are. We do not consider the structural similarity measures like tree edit distance because of the high computational complexity and low additional profit.

4 Diversified Top-*k* Search

As mentioned above, we interpret a keyword query by searching its pattern trees on the schema graph. In this section, we present the top-*k* search algorithm and optimization techniques.

4.1 Search Algorithm

The main goal of our algorithm is to avoid the unnecessary similarity comparison, which is relatively expensive and could be used very frequently during the search. It is due to the fact that lots of pattern trees with low scores generated during the search are unlikely to become the top-*k* results and thereby are not needed for similarity comparison. Thus, our algorithm calculates the similarity for a pattern tree only when its score meets a specific condition.

The pseudo codes are given in Algorithm 1. Let C be the candidate set, which is a priority queue of generated pattern trees in descending order of score, and S_k be the diversified top-*k* result set, which is also sorted in descending order of score. We denote by $\overline{unseen}$ the upper bound of score for the unseen results. For simplicity, the function $search()$ is used to traverse the schema graph and generate a set of pattern trees in each iteration. We do not discuss how to schedule the graph traversals and how to generate pattern trees here, which have been well studied by existing research works like [10]. Obviously, the value of $\overline{unseen}$ will decrease gradually, and meanwhile, there will emerge pattern trees with higher scores than $\overline{unseen}$. Then we compute the similarity between the first emerged pattern tree and results in S_k. If it is not similar to any result in S_k, it will be put into S_k. Otherwise, it will be abandoned directly. The algorithm terminates as soon as there are k results in S_k.

It is easy to prove that, each candidate popped up from C is a pattern tree with the highest score in all remaining results, so that it is certainly the next top-*k* result if it is not similar to any existing top-*k* result. Thus, the correctness of Algorithm 1 is guaranteed.

Then we prove that we reduce the cost of similarity comparison to the minimum. The pattern trees with a score higher than the last result in S_k have to be compared with the results in S_k for identifying whether it is qualified for becoming one of top-*k*. While, the similarity comparison of the rest pattern trees is totally unnecessary to find the top-*k* results. Our algorithm only calculates the similarity while the score of the pattern is higher than $\overline{unseen}$, which is always no less than the score of the last result in S_k. So our algorithm minimizes the calculation of similarity.

Algorithm 1. DivSA

```
Input: a schema graph $G_s$, a keyword query $Q$
Output: $S_k$
$C \leftarrow \emptyset$, $S_k \leftarrow \emptyset$, $\overline{unseen} \leftarrow 1$;
while $|S_k| < k$ do
  $temp\_set \leftarrow search()$; //see prune details in Section 4.2
  for each $T \in temp\_set$
    put $T$ into the candidate set $C$;
  end for
  update $\overline{unseen}$;
  while $C.peek() > \overline{unseen}$ and $|S_k| < k$ do
    $T \leftarrow$ the first candidate in $C$;
    if $T$ is not similar to any results in $S_k$ do
      put $T$ into $S_k$;
    end if
  end while
end while
```

Execution example. Consider the example in Figure 2. Table1 describes the search procedure of DivSA. Initially, the value of $\overline{unseen}$ is 1, and candidate set C and diversified top-k results set S_k are both empty. The pattern trees are generated by calling $search()$ iteratively. After two iterations, T_1, T_2, T_3, T_4 and T_5 have been generated, and $\overline{unseen}$ decreases to 0.85. Then, the pattern tree T_3 ($score(T_3) = 0.9 > 0.85$) is moved to S_k because it is certainly the best of all possible results. In the next iteration, T_6 is generated and $\overline{unseen}$ decreases to 0.75. Then, T_1 ($score(T_1) = 0.8 > 0.75$) becomes the best of rest results, and meanwhile it is not similar to T_3, thereby being moved from C to S_k. When $\overline{unseen}$ decreases to 0.45, there are five pattern trees in C, i.e., $\{T_4, T_6, T_7, T_2, T_5\}$, where T_4, T_6 and T_7 have higher score than $\overline{unseen}$. So they will be compared with the results in S_k one by one. Lastly, T_6 and T_7 are moved to S_k, and T_4 is removed from C because it is similar to T_3.

Table 1. An example search procedure of DivSA.

iteration #	new results	C	S_k	$\overline{unseen}$
0	$\emptyset$	$\emptyset$	$\emptyset$	1
1	$\{T_1, T_2\}$	$\{T_1, T_2\}$	$\emptyset$	0.9
2	$\{T_3, T_4, T_5\}$	$\{T_3, T_1, T_4, T_2, T_5\}$	$\{T_3\}$	0.85
3	$\{T_6\}$	$\{T_1, T_4, T_6, T_2, T_5\}$	$\{T_3, T_1\}$	0.75
4	$\{T_7\}$	{~~T_4~~, T_6, T_7, T_2, T_5}	$\{T_3, T_1, T_6, T_7\}$	0.45

4.2 Optimization

In order to improve the search efficiency, we propose two rules to prune the search path that is traversed by $search()$ in each iteration.

Rule 1. Let m_q be the maximum score of search paths outgoing from the keyword $q \in Q$, and $score_{max}$ be the highest score of currently generated pattern trees. For a new path $P_{q'}$ outgoing from a keyword $q' \in Q$, we prune $P_{q'}$ if $score(P_{q'}) + \sum_{q \in Q, q \neq q'} m_q < \mu * score_{max}$.

The rationality behind Rule 1 is as follows. Firstly, we prove that the left side of the inequality is the upper bound of score of pattern trees that contain the path $P_{q'}$. As indicated in Section 2.2, the score of a search path composed of another path and a new edge is certainly less than the original path. So the possibly best pattern tree that contains the path $P_{q'}$ is composed of the current best paths outgoing from each keyword. According to Equation (1), $score(P_{q'}) + \sum_{q \in Q, q \neq q'} m_q$ is the upper bound.

The right side of the inequality is the estimated lower bound of score of top-k results. The lower bound is to prevent the similarity computation for results with very low scores. The empirical parameter μ is used to balance the prune effect and result completeness. It is possible to return fewer than k results when the value of μ is relatively large. But in practice the returned results can be guaranteed to be complete by carefully tuning the value of μ.

Rule 2. For two search paths P_q and P_q' outgoing from a same keyword q with $\text{score}(P_q) > \text{score}(P_q')$, if $sim(P_q, P_q') = 1$, we can prune the search path P_q' safely, and if $1 > sim(P_q, P_q') > \tau$, we will prune the search path P_q' with a probability $\delta = 1 - \cos^{-1} sim(P_q, P_q') / \cos^{-1} \tau$.

First, assume that the pattern trees T_i and T_j contain P_q and P_q' respectively. And the other search paths of T_i and T_j are same. If $\text{score}(P_q) > \text{score}(P_q')$ and $sim(P_q, P_q') = 1$, then $\text{score}(T_i) > score(T_j)$ and $sim(T_i, T_j) = 1$. The similarity between two paths is one do not mean that the two paths are exactly the same. Because the feature vector of the path does not describe all details of the path. The similarity between pattern trees is calculated entirely based on the similarity between paths. So we can know $sim(T_i, T_j) = 1$.The score of the pattern tree is the sum of the score of all its paths. So we can know $\text{score}(T_i) > score(T_j)$.

In one case, when T_i is in the S_k, because $\text{score}(T_i) > score(T_j)$ and $sim(T_i, T_j) = 1$, T_j can't be a result in the S_k. In another case, when T_i is not a result in the S_k, there must be a pattern tree T_x in S_k having higher score than T_i and $1 > sim(T_i, T_x) > \tau$. Since the feature vector of all paths of T_i and T_j are the same, we can know that $1 > sim(T_j, T_x) > \tau$. And obviously, $\text{score}(T_x) > \text{score}(T_j)$. So T_j can't be a result in the S_k.

In summary, T_j is not needed in the final results. And that also means that P_q' is not needed during the generation of pattern trees. Because if there is any pattern tree containing P_q', there is always another pattern tree containing P_q making the previous one useless. So we can prune the search path P_q'.

When $1 > sim(P_q, P_q') > \tau$, δ is a good simulation of the possibility of $1 > sim(T_j, T_x) > \tau$(According to the preceding proof, it is easy to understand that it is also the possibility to prune the search path P_q'). In fact, when P_q is more similar to P_q', P_q' is more likely to be pruned. δ simulates this trend with very good effect. So we use it as a probability to prune the path P_q'.

5 Experiments

5.1 Setup

Dataset. We perform the experiments on DBPedia, a popular real-world RDF dataset which contains over two million entities and nearly ten million relationships. From DBPedia, we extract a schema graph with 272 class nodes, such as "Aircraft", "BaseballPlayer", "ChemicalCompound", etc. and nearly 20K edges. Our keyword query interpretation approach is to search for pattern trees on the schema graph and generate diverse patterns covering various classes.

Metrics. To evaluate the effectiveness of our approach, we introduce the following two metrics.

- *Coverage*. Each top-*k* pattern tree could be a representative of many other similar patterns. Thus, we try to count how many patterns are covered by (namely, similar to) the top-*k* pattern trees, which is a reasonable measure of their diversity. Certainly, the more pattern trees covered by the top-*k* results, the more diverse the results. Formally, we define the coverage of diversified top-*k* results S_k as

$$coverage(S_k) = \frac{|\{T \in S, LT \in S_k | score(T) \geq score(LT)\}|}{|S_k|} \tag{5}$$

where S is the result set including all the pattern trees once generated and LT is the last pattern tree in S_k.

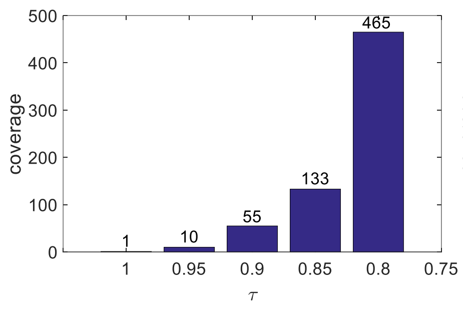
- *Precision*. Since our probabilistic pruning method could result in losses of exact top-*k* results, we evaluate the effectiveness of pruning method by using the precision of top-*k* results, namely, the percentage of "correct" top-*k* results that are also returned by the original algorithm. Formally, we define the precision of top-*k* results S_k as

$$precision(S_k) = \frac{|\{P \in S_k | P \in S_k'\}|}{|S_k|} \tag{6}$$

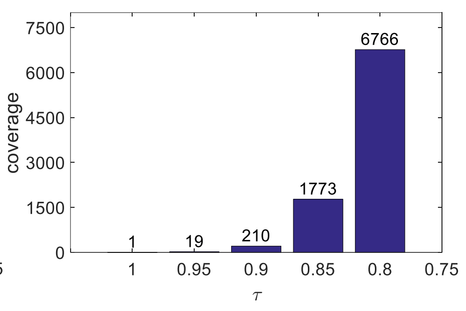
where S_k' is the set of exact top-*k* results corresponding to S_k.

5.2 Effectiveness

Test 1. We firstly evaluate the effectiveness in the sense of coverage. We test 50 random keyword queries that have two or three keywords respectively, with k = 10 and varying values of τ. As shown in Figure 3, the value of coverage increases



(a) Coverage of S_k with |*Q*|=2.



(b) Coverage of S_k with |*Q*|=3.

Fig. 3. Coverage of $\boldsymbol{S_k}$.

significantly with the decrease of τ. For example, when τ = 0.8, there are averagely 4655 pattern trees represented by our diversified top-10 results of two-keyword queries. Moreover, if the queries have more keywords, the coverage of diversified top-*k* results is even higher, because there are more redundant pattern trees combined from similar paths.

Test 2. We also evaluate the effectiveness by conducting a user case study. Given a query "Beckham, Ronaldo", Table 2 shows the top-5 results without diversification and Table 3 shows the top-5 results of DivSA with τ = 0.9. We can see that, the top-1 result without diversification is indeed the most desired semantic relationship between these two famous soccer players, namely, another soccer player who is the teammate of both. However, the other 4 results are all about soccer player, though they are slightly different. In contrast, the results of DivSA reveal rich semantics between the two keywords. Thus, if some users are interested in "Beckham" as an album and "Ronaldo" as a musical artist, they will find out that the album and the artist share some sort of musical genre. Thus, our approach can indeed improve the search results in human sense.

Table 2. A user case study: the top-5 results without diversification.

Root	Paths
Soccer Player	Soccer Club—Soccer Player(Beckham) Soccer Club—Soccer Player(Ronaldo)
Soccer Player	Soccer Club—Soccer Player—Person(Beckham) Soccer Club—Soccer Player(Ronaldo)
Soccer Player	Soccer Club—Soccer Player—Album(Beckham) Soccer Club—Soccer Player(Ronaldo)
Soccer Player	Soccer Club—Soccer Player(Beckham) Soccer Club—Stadium(Ronaldo)
Soccer Player	Soccer Club—Soccer Player—Person(Beckham) Soccer Club--Stadium(Ronaldo)

Table 3. A user case study: the top-5 results of DivSA.

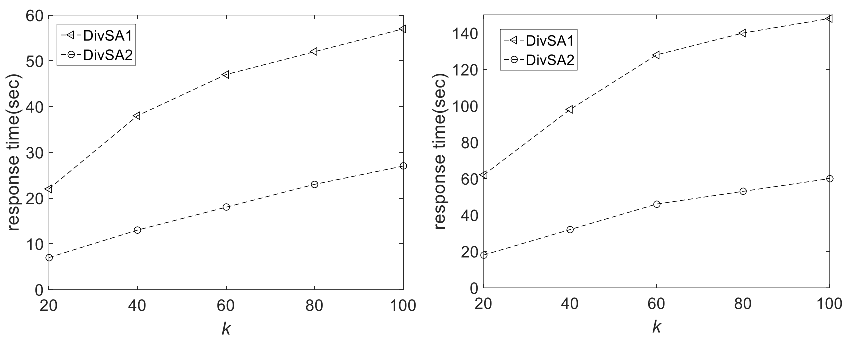
Root	Paths
Soccer Player	Soccer Club—Soccer Player(Beckham) Soccer Club—Soccer Player(Ronaldo)
Broadcast Network	Country—Administrative Region—Settlement(Beckham) Country—Administrative Region—School(Ronaldo)
Music Genre	Album(Beckham) Musical Artist(Ronaldo)
Book	Language—Book(Beckham) Country—Administrative Region—School(Ronaldo)
Record Label	Album(Beckham) Country—Administrative Region—School(Ronaldo)

5.3 Efficiency

We compare the efficiency of two algorithms: DivSA1 and DivSA2. DivSA1 is our diversified top-k search algorithm DivSA using Rule 1 for pruning. DivSA2 is the DivSA using both Rule 1 and 2 for pruning.

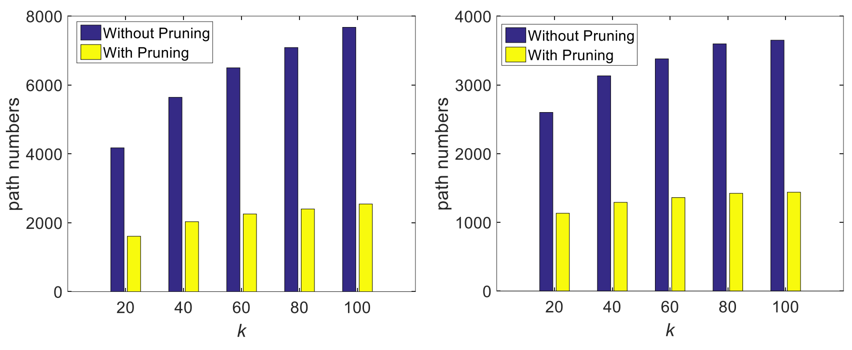
We test 50 random keyword queries with two or three keywords respectively by using each algorithm with varying values of parameters. The followings are our observations.

1. Figure 4 depict the average response time of all algorithms with τ=0.9, $\mu = 0.5$ and $k = 20, 40, \ldots, 100$ for queries with two and three keywords respectively. In most of the time, DivSA1 can find the top-k results within tens of seconds, since it reduces the overheads of calculating similarity significantly. Moreover, the optimized DivSA2 is more efficient than DivSA1. Specifically, DivSA2 is averagely 2.43 and 2.92 times faster than DivSA1 when the keyword number is 2 and 3 respectively. It verifies the effectiveness of our pruning strategy.
   With the increase of the value of k, the response time of both DivSA1 and DivSA2 increases sub-linearly.
   With the increase of the keyword number in query, the response time of both DivSA1 and DivSA2 increases rapidly, due to the explosive combinations of paths. Thus, keyword query cleaning is useful when there are many keywords.
2. Figure 5 demonstrate the effectiveness of pruning. Generally, we can see that the number of search paths is reduced significantly by pruning. For the queries with two keywords, 66% of search paths are pruned. For the queries with three keywords, 59% of search paths are pruned. Meanwhile, although the pruning is heuristic, the precision is testified to be quite high.
3. Since our heuristic pruning is not guaranteed to be safe, we need to testify the precision of returned results. Table 4 shows the average precision of top-k results of DivSA2. We can see that the average precision is generally higher than 95%. Thus, although DivSA2 improves the efficiency dramatically by using unsafe pruning, it is still reliable with respect to the precision of top-k results.



(a) Average response time |Q|=2 **(b)** Average response time |Q|=3

Fig. 4. Average response time.



(a) Pruning |Q|=2 **(b)** Pruning |Q|=3

Fig. 5. Pruning effectiveness of DivSA2.

Table 4. Precision of the top-*k* results returned by DivSA2.

	k=20	*k*=40	*k*=60	*k*=80	*k*=100
$\lvert Q\rvert = 2$	0.975	0.98295	0.96842	0.97045	0.97545
$\lvert Q\rvert = 3$	0.98667	0.965	0.96667	0.96667	0.95286

6 Conclusion

In this paper, we study the problem of diversified top-*k* keyword query interpretation on knowledge graphs. Firstly, we define the problem as diversified top-*k* search on a schema graph. An effective similarity measure is proposed to evaluate the semantic similarity between search results. Then, we present an efficient search algorithm that guarantees to return the exact top-*k* results and minimize the calculation of similarity. In order to further optimize the algorithm, we propose two heuristic pruning strategies. Lastly, we perform experiments on a real-world knowledge graph to verify the effectiveness and efficiency of our approach.

Acknowledgments

This work was supported by National Natural Science Foundation of China under contracts 61202036, 61572376, 61502349, and 61272110, and by Wuhan Morning Light Plan of Youth Science and Technology under contract 2014072704011250.

References

1. Agrawal R., Gollapudi S., Halverson A., Ieong S.: Diversifying search results. In: WSDM, pp. 5-14. (2009).
2. Angel A., Koudas N.: Efficient diversity-aware search. In: SIGMOD, pp. 781-792. (2011).
3. Auer S., Bizer C., Kobilarov G., Lehmann J., Cyganiak R., Ives Z.: Dbpedia: A nucleus for a web of open data. In: ISWC/ASWC, pp. 722-735. (2007).
4. Bollacker K., Evans C., Paritosh P., Sturge, T., Taylor J.: Freebase: A collaboratively created graph database for structuring human knowledge. In: SIGMOD, pp. 1247–1250. (2008).
5. Pound J., IIyas I.F., Weddell G.: Expressive and flexible access to web-extracted data: A keyword-based structured query language. In: SIGMOD, pp. 423-434. (2010).
6. Pound J., Hudek A.K., IIyas I.F., Weddell G.: Interpreting keyword queries over web knowledge bases. In: CIKM, pp. 305–314. (2012).
7. Qin L., Yu J.X., Chang L.: Diversifying top-k results. In: VLDB, pp. 1124-1135. (2012).
8. Suchanek F.M., Kasneci G., Weikum G.: Yago: A core of semantic knowledge unifying wordnet and wikipedia. In: WWW, pp. 697–706. (2007).
9. Tran T., Cimiano P., Rudolph S., Studer R.: Ontology-based interpretation of keywords for semantic search. In: ISWC, pp. 523–536. (2007).
10. Tran T., Wang H., Rudolph S., Cimiano P.: Top-k Exploration of Query Candidates for Efficient Keyword Search on Graph-Shaped (RDF) Data. In: ICDE, pp. 405-419. (2009).
11. Wu W., Li H., Wang H., Zhu K.: Probase: A probabilistic taxonomy for text understanding. In: SIGMOD, pp. 481–492. (2012).
12. Wu Y., Yang S., Srivatsa M., Iyengar A., Yan X.: Summarizing answer graphs induced by keyword queries. In: VLDB, pp. 1774–1785. (2013).
13. Zeng Z., Bao Z., Le T.N., Lee M.L., Ling W.T.: ExpressQ: Identifying keyword context and search target in relational keyword queries. In: CIKM, pp. 31–40. (2014).
14. Zhao F., Zhang X., Tung A.K.H., Chen G.: BROAD: Diversified keyword search in databases. In: VLDB, pp. 1355-1358. (2011).
15. Zhou Q., Wang C., Xiong M., Wang H., Yu Y.: Spark: Adapting keyword query to semantic search. In: ISWC, pp. 694–707. (2007).
16. Garbonell J.G. and Goldstein J.: The use of MMR, diversity-based reranking for reordering documents and producing summaries. In: SIGIR, pp. 335-336. (1998).
17. Demidova E., Fankhauser P., Zhou X. and Nejdl W.: DivQ: diversification for keyword search over structured databases. In: SIGIR, pp. 331-338. (2010).
18. Golenberg K., Kimelfeld B. and Sagiv Y.: Keyword proximity search in complex data graphs. In: SIGMOD, pp. 927-940. (2008).