Generating Compact and Relaxable Answers to Keyword Queries over Knowledge Graphs

Gong Cheng[1(✉)], Shuxin Li[1], Ke Zhang[1], and Chengkai Li[2]

[1] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
gcheng@nju.edu.cn, {sxli,161120169}@smail.nju.edu.cn
[2] Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, USA
cli@uta.edu

Abstract. Keyword search has been a prominent approach to querying knowledge graphs. For exploratory search tasks, existing methods commonly extract subgraphs that are group Steiner trees (GSTs) as answers. However, a GST that connects all the query keywords may not exist, or may inevitably have a large and unfocused graph structure in contrast to users' favor to a compact answer. Therefore, in this paper, we aim at generating compact but relaxable subgraphs as answers, i.e., we require a computed subgraph to have a bounded diameter but allow it to only connect an incomplete subset of query keywords. We formulate it as a new combinatorial optimization problem of computing a minimally relaxed answer with a compactness guarantee, and we present a novel best-first search algorithm. Extensive experiments showed that our approach efficiently computed compact answers of high completeness.

Keywords: Keyword search · Knowledge graph · Query relaxation

1 Introduction

Non-expert users have difficulty in querying a knowledge graph (KG) without a prior knowledge of the specialized query language. Keyword search provides users with convenient access to KGs, by automatically matching keyword queries with KGs. For a lookup task where there is a specific search target that can be represented as a precise formal query such as a SPARQL query over RDF-based KGs, existing methods commonly transform the keyword query into a formal query to execute [10,22,25]. For exploratory search [19] where the keyword query and the underlying search target are vague and cannot be precisely interpreted as a formal query, e.g., searching for relationships between a set of entities [2], one promising solution is to directly extract subgraphs from KGs that contain query keywords as answers [7,16,18,23]. The extracted subgraphs should satisfy certain constraints and have high quality. For example, it has been standard to

J. Z. Pan et al. (Eds.): ISWC 2020, LNCS 12506, pp. 110–127, 2020.
https://doi.org/10.1007/978-3-030-62419-4_7

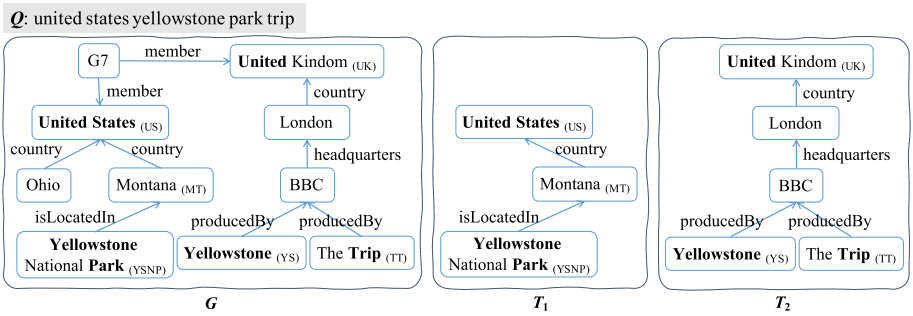


Fig. 1. A keyword query Q over a KG G with two answers T_1 and T_2.

extract a *group Steiner tree* (GST) [13]. By assigning weights to vertices and/or edges of KGs to represent salience, a GST is a tree of minimum total weight that connects as least one matching vertex for each query keyword.

Motivation. The effectiveness of GST-based answers to human users in exploratory search was recently challenged by the results of an extensive user study [5]. Several popular weighting schemes were evaluated in [5] but, surprisingly, none of them were assessed to be useful for shaping a favourable answer. Instead, the study showed that users strongly favor answers that are structurally *compact* subgraphs having a small diameter, thereby suggesting extracting such subgraphs as answers. However, when some query keywords are absent, disconnected, or only connected by long paths in the KG, one cannot find any structurally compact subgraph that connects all the query keywords. To address this problem, we proposed to extract compact but *relaxable* subgraphs that have a small diameter to ensure compactness but are not required to connect all the query keywords, i.e., allowing query relaxation [17]. The preliminary search algorithm we designed in [17], called CertQR+, has several limitations that will be addressed in this paper.

Our Approach. We aim at computing *co*mpact and *re*laxable subgraphs as answers to exploratory keyword search over KGs. Our algorithm is abbreviated to **CORE** and is open source.[1] To help understanding our approach, consider the example in Fig. 1. For keyword query "united states yellowstone park trip", a traditional GST-based answer has to contain the long path between `Yellowstone National Park` and `The Trip` in the KG. This unfocused path is not interesting to the user but prevents structurally compact subgraphs as answers. In comparison, CORE can compute answers like T_1 that is more compact and meaningful though drops a keyword "trip" in the query. Answer completeness (i.e., covering all the query keywords) and compactness (i.e., having a small diameter) are conflicting objectives. CORE achieves the following trade-off: computing a minimally relaxed answer (i.e., covering the largest number of query keywords) with a compactness guarantee (i.e., having a bounded diameter). To model this

[1] https://github.com/nju-websoft/CORE.

trade-off, we formulate a combinatorial optimization problem, and we design an efficient algorithm for computing an optimum answer.

Contributions. We summarize our contributions in the paper as follows.

- To generate compact and relaxable subgraphs as answers, we formulate a new combinatorial optimization problem of computing a minimally relaxed answer with a compactness guarantee (MRA).
- To solve MRA, we design a best-first search algorithm called CORE. The efficiency of the algorithm benefits from a theoretical result proved in the paper which exploits distance calculation.
- With public KGs and keyword queries, we demonstrate the necessity of trading off answer completeness for compactness, and we show that CORE performs significantly faster than CertQR+.

The remainder of the paper is organized as follows. Section 2 formulates MRA. Section 3 discusses related work. Section 4 describes CORE. Section 5 presents experiments. Finally we conclude the paper in Sect. 6.

2 Problem Formulation

2.1 Preliminaries

Below we define necessary terms used in the paper.

Knowledge Graph. A *knowledge graph* (KG) represents a set of linked and annotated entities. We formulate it as a directed graph $G = \langle V, E \rangle$, where V is a set of n annotated vertices representing entities, and $E \subseteq V \times V$ is a set of m annotated edges representing relations between entities. We keep edge directions, but the edges in a path/tree can be oriented in different directions.

Keyword Query. Let $\mathbb{K}$ be the set of all keywords. A retrieval function $\texttt{hits} : \mathbb{K} \mapsto 2^V$ maps keywords to subsets of vertices from V. The concrete implementation of `hits`, i.e., the exact way of matching keywords with entity annotations, is not our research focus. For simplicity, edge matching is omitted from our formulation but is supported by our approach. Indeed, we can subdivide an edge (u, v) by yielding a new vertex w with the annotations of the edge (u, v) and then replacing (u, v) by two new edges (u, w) and (w, v). A *keyword query* $Q \subseteq \mathbb{K}$ is a non-empty set of g keywords $Q = \{k_1, \ldots, k_g\}$. For the ease of notation we write $\texttt{hits}(k_i)$ as K_i for $1 \leq i \leq g$ and call them *keyword vertices*.

Query Answer. A *complete answer* to Q is a subgraph $T = \langle V_T, E_T \rangle$ such that: (1) T is connected, (2) T covers all the query keywords, i.e., $V_T \cap K_i \neq \emptyset$ for $1 \leq i \leq g$, and (3) T is minimal, i.e., none of its proper subgraphs satisfy both (1) and (2). Minimality indicates T's tree structure where leaves are keyword vertices. For example, in Fig. 1, T_1 is a complete answer to "united states yellowstone park", and T_2 is a complete answer to "united yellowstone trip".

Graph Terminology. $\mathtt{N}(u)$ is the set of vertex u's *neighbors*, i.e., vertices adjacent from/to u. The *degree* of a vertex is the number of incident edges. We define *path* in a standard way, except that the edges can be oriented differently. The *length* (`len`) of a path is the number of edges. The *distance* (`d`) between two vertices is the length of a shortest path connecting them. The *eccentricity* of vertex u is the greatest distance between u and other vertices. The *radius* (`rad`) and *diameter* (`diam`) of a graph are the minimum and maximum eccentricity of the vertices, respectively. A *central vertex* is a vertex of minimum eccentricity.

2.2 Problem Statement

We assess the quality of an answer by its degrees of relaxation and compactness.

Degree of Relaxation. We allow an answer to cover an incomplete subset of Q. For $\emptyset \subset Q' \subset Q$, a complete answer to Q' is called a *partial answer* to Q. Complete and partial answers are collectively called *answers*. We measure the *degree of relaxation* (dor) of answer T to Q as follows:

$$\mathtt{dor}(T) = |Q \setminus \mathtt{CQ}(T)|\,, \quad \mathtt{CQ}(T) = \{k_i \in Q : K_i \cap V_T \neq \emptyset\}\,, \tag{1}$$

where $\mathtt{CQ}(T)$ represents the subset of keywords in Q that are covered by T. For example, T_1 and T_2 in Fig. 1 are partial answers to "united states yellowstone park trip" with $\mathtt{dor}(T_1) = 1$ and $\mathtt{dor}(T_2) = 2$.

Degree of Compactness. Following [3–5], we measure the *degree of compactness* (doc) of an answer by its diameter:

$$\mathtt{doc}(T) = \mathtt{diam}(T)\,. \tag{2}$$

For example, in Fig. 1 we have $\mathtt{doc}(T_1) = 2$ and $\mathtt{doc}(T_2) = 3$.

MRA Problem. To achieve a trade-off between answer completeness and compactness, we aim at computing a *m*inimally *r*elaxed *a*nswer with a compactness guarantee, abbreviated to the **MRA** problem. Such an answer has a bounded degree of compactness and the smallest degree of relaxation:

$$\underset{T\,:\,\mathtt{doc}(T) \leq D}{\arg\min}\ \mathtt{dor}(T)\,, \tag{3}$$

where $D \geq 0$ is a predetermined integer bound.

3 Related Work

Our work is related to but different from the following research in the literature.

Semantic Parsing. One way to interpret a keyword query over a KG is to turn it into a formal query such as a SPARQL query to be executed using a standard back end [10,22,25]. Such methods are suitable for lookup tasks where there is a specific search target that can be formally and precisely represented.

Some of these methods support representing as a basic graph pattern while other methods support expressive constructs. However, they are not very suitable for exploratory search with a vague search target.

Answer Extraction. Methods in the style of subgraph extraction are flexible, thus suitable for exploratory search. Their effectiveness relies on the scoring function used to rank subgraphs. In theory, all the functions for ranking candidate formal queries in semantic parsing can be adapted here to take account of the semantics of a keyword query. However, existing methods commonly extract GSTs [13] using exact [7,16,18] or approximation algorithms [23]. The weighting schemes used are simple and seem not very effective [5]. The computed min-weight subgraphs are not necessarily structurally compact as favored by human users [5]. Therefore, we are motivated to compute structurally compact subgraphs as answers, leaving the incorporation of weights for future work.

Answer Compactness. We follow [3–5,12] to bound the diameter of an answer. As we will see in Sect. 4, our MRA problem based on such a bound admits a polynomial-time solution. Alternatively, if we bound the number of vertices as in [1,14,24], the problem will be intractable because one can easily reduce to it from the GST problem with unit weights. In [8,15], all possible radius-bounded answers are offline extracted and indexed. They allow fast online retrieval of answers but their bounds cannot be online tuned as in our approach.

Answer Completeness. Our CORE extends CertQR+ [17] by overcoming several limitations. First, CORE is a more general approach. It supports keyword search where a keyword is mapped to a set of vertices, whereas CertQR+ only supports entity relationship search [2] where a keyword is mapped to a single vertex. Second, CORE has a better overall performance. It directly computes a minimally relaxed answer, whereas CertQR+ only computes a minimally relaxed query which needs further execution. Third, as we will see in Sect. 4, the depth of search $\lfloor \frac{D}{2} \rfloor$ in CORE is smaller than $\lceil \frac{D}{2} \rceil$ in CertQR+ when D is odd. There are other studies of query relaxation in the Semantic Web community such as [9,21,26]. They relax SPARQL queries, e.g., by making some triple patterns optional. These methods could not directly apply to our MRA problem.

4 Approach

In this section, we firstly give some theoretical foundations, and then we design and analyze our algorithm CORE.

4.1 Theoretical Foundations

In the following proposition, we establish a necessary and sufficient condition for the existence of a compactness-bounded complete answer to a keyword query. It describes the existence of a vertex v that is fairly close to all the query keywords. Later in the paper we will employ this proposition to design CORE.

Proposition 1. *A complete answer $T = \langle V_T, E_T \rangle$ to Q with $\texttt{diam}(T) \leq D$ exists if and only if a vertex $v \in V$ exists such that*

1. $\forall 1 \leq i \leq g,\ \exists u \in K_i,\ \texttt{d}(u, v) \leq \lceil \frac{D}{2} \rceil$;
2. *if D is odd and $\widehat{Q} = \{k_i \in Q : \min_{u \in K_i} \texttt{d}(u, v) = \lceil \frac{D}{2} \rceil\} \neq \emptyset$, then $\exists v' \in \texttt{N}(v),\ \forall k_i \in \widehat{Q},\ \exists u \in K_i,\ \texttt{d}(u, v') = \lfloor \frac{D}{2} \rfloor$;*
3. $\exists 1 \leq i \leq g,\ \exists u \in K_i,\ \texttt{d}(u, v) \leq \lfloor \frac{D}{2} \rfloor$.

Proof. We present a constructive proof.

Necessity. Let v be an arbitrary central vertex of T. Below we will show that v satisfies all the three conditions in the proposition.

For Condition 1, since T is a tree, graph theory tells us that

$$\texttt{rad}(T) = \left\lceil \frac{\texttt{diam}(T)}{2} \right\rceil \leq \left\lceil \frac{D}{2} \right\rceil . \tag{4}$$

Let $\texttt{d}$ and $\texttt{d}_T$ be the distances between two vertices in G and in T, respectively. Since T is a complete answer to Q, we have $\texttt{CQ}(T) = Q$, i.e., $\forall 1 \leq i \leq g,\ \exists u \in (K_i \cap V_T)$. Because T is a subgraph of G, and v is a central vertex of T, we have

$$\texttt{d}(u, v) \leq \texttt{d}_T(u, v) \leq \texttt{rad}(T) \leq \left\lceil \frac{D}{2} \right\rceil . \tag{5}$$

For Condition 2, if D is odd and $|\widehat{Q}| \neq \emptyset$, we choose an arbitrary $k_i \in \widehat{Q}$ and $u_i = \arg\min_{u' \in (K_i \cap V_T)} \texttt{d}(u', v)$, i.e., $\texttt{d}(u_i, v) \geq \lceil \frac{D}{2} \rceil$. Since T is a tree, consider the unique path P_i between u_i and v in T. We have $\texttt{len}(P_i) = \texttt{d}_T(u_i, v)$, and

$$\texttt{d}_T(u_i, v) \leq \texttt{rad}(T) \leq \left\lceil \frac{D}{2} \right\rceil \quad \text{and} \quad \texttt{d}_T(u_i, v) \geq \texttt{d}(u_i, v) \geq \left\lceil \frac{D}{2} \right\rceil . \tag{6}$$

Therefore, $\texttt{len}(P_i) = \texttt{d}_T(u_i, v) = \texttt{d}(u_i, v) = \lceil \frac{D}{2} \rceil$ and hence P_i is a shortest path between u_i and v in G. Let $v' \in \texttt{N}(v)$ be the unique neighbor of v in P_i. Below we show that v and v' satisfy Condition 2. We trivially have $\texttt{d}(u_i, v') = \lfloor \frac{D}{2} \rfloor$. Then $\forall k_j \in (\widehat{Q} \setminus \{k_i\})$, let $u_j = \arg\min_{u' \in (K_j \cap V_T)} \texttt{d}(u', v)$. Similar to the above proof, we know $\texttt{d}(u_j, v) = \lceil \frac{D}{2} \rceil$ and the unique path P_j between u_j and v in T is their shortest path in G with $\texttt{len}(P_j) = \lceil \frac{D}{2} \rceil$. This path has to pass through v' and hence $\texttt{d}(u_j, v') = \lfloor \frac{D}{2} \rfloor$ because otherwise

$$\texttt{d}_T(u_i, u_j) = \texttt{len}(P_i) + \texttt{len}(P_j) = \left\lceil \frac{D}{2} \right\rceil + \left\lceil \frac{D}{2} \right\rceil = D + 1 , \tag{7}$$

which contradicts $\texttt{diam}(T) \leq D$.

For Condition 3, when D is even, it is easily derived from Condition 1. When D is odd, we prove by contradiction and assume: $\forall 1 \leq i \leq g,\ \forall u \in K_i,\ \texttt{d}(u, v) > \lfloor \frac{D}{2} \rfloor$. Combined with Condition 1 we obtain $\forall 1 \leq i \leq g,\ \min_{u \in K_i} \texttt{d}(u, v) = \lceil \frac{D}{2} \rceil$ and hence $Q = \widehat{Q}$. Now we discuss the cardinality of $|Q|$. When $|Q| = 1$, T is a

trivial graph, and Condition 3 is satisfied. When $|Q| = |\widehat{Q}| \geq 2$, if we consider v' in Condition 2 as a new v, the above assumption will be contradicted as one can easily verify that this new v satisfies all the conditions.

Sufficiency. If D is even, we call GenAnsEven(G, Q, D, v) in Algorithm 3 to construct T. If D is odd, we call GenAnsOdd$(G, Q, D, v, v', \widehat{Q})$ in Algorithm 4. We will detail these algorithms later in the paper.

□

Certificate Vertex. In Proposition 1, vertex v is a certificate of the existence of a compactness-bounded complete answer to Q. We refer to v as a *certificate vertex* for Q. For example, in Fig. 1, under $D = 2$, $v =$ `Montana` is a certificate vertex for "united states yellowstone park". Under $D = 3$, $v =$ `London` is a certificate vertex with $v' =$ `BBC` for "united yellowstone trip".

Proposition 1 helps to solve the MRA problem in two aspects. First, if we want to decide the existence of a compactness-bounded complete answer to Q or to some $Q' \subset Q$, we can avoid actually searching for an answer which would be an expensive process. Instead, we only need to calculate a few distances. Distance calculation can be efficiently implemented based on the proper graph indexes. Second, once the above existence of an answer is confirmed, i.e., a certificate vertex v is found, we will be able to easily construct such an answer using v. These two benefits form the foundation of our algorithms.

4.2 Algorithm Design

Below we overview and then detail CORE, and finally show a running example.

Overview. Under a compactness bound D, our main algorithm CORE adopts a best-first search strategy. Iteratively, the most promising search direction is explored, which may update the current best (i.e., minimally relaxed) answer. The search process will be terminated if it is guaranteed that unexplored answers cannot exceed the current best answer. In the search process, rather than directly searching for a better (i.e., less relaxed) answer, our subroutine FindAns firstly searches for a certificate vertex for a larger subset of Q, and then constructs a complete answer to this sub-query using the subroutine GenAnsEven or GenAnsOdd, depending on the parity of D. These two steps in FindAns are both supported by Proposition 1.

Algorithm 1: CORE

```
   Input: G = ⟨V, E⟩,
          Q = {k_1, ..., k_g}, D.
   Output: An optimum answer T_opt.
1  T_opt ← null;
   /* CQ(null) = ∅, dor(null) = |Q| */
2  foreach u ∈ ⋃_{i=1}^{g} K_i do
3  |  visited[u][u] ← true;
4  |  foreach w ∈ (V \ {u}) do
5  |  |  visited[u][w] ← false;
6  PQ ← an empty min-priority queue
   of ordered pairs of vertices;
7  foreach u ∈ ⋃_{i=1}^{g} K_i do
8  |  PQ.Insert(⟨u, u⟩);
9  foreach u ∈ V do
10 |  checked[u] ← false;
11 while PQ is not empty do
12 |  ⟨u, v⟩ ← PQ.Pull();
13 |  if pri(u, v) ≥ dor(T_opt) then
14 |  |  break the while loop;
15 |  if checked[v] is false then
16 |  |  T ←
   |  |  FindAns(G, Q, D, v, T_opt);
17 |  |  checked[v] ← true;
18 |  |  if T is not null then
19 |  |  |  T_opt ← T;
20 |  if d(u, v) < ⌊D/2⌋ then
21 |  |  foreach w ∈ N(v) do
22 |  |  |  if visited[u][w] is false
   |  |  |  then
23 |  |  |  |  visited[u][w] ← true;
24 |  |  |  |  PQ.Insert(⟨u, w⟩);
25 return T_opt;
```

Algorithm CORE. Our main algorithm is presented in Algorithm 1. T_{opt} represents the current best answer (line 1). We run one independent search starting from each keyword vertex in $\bigcup_{i=1}^{g} K_i$. Each search maintains a separate set of visited vertices; $visited[u][w]$ represents whether w has been visited in the search starting from u (lines 2–5). The frontiers of all these searches are kept in a shared priority queue PQ where each element is an ordered pair of vertices $\langle u, v\rangle$ representing a vertex v to be explored in the search starting from u (line 6). Initially, PQ is fed with all the keyword vertices (lines 7–8). A vertex can be visited multiple times, at most one time in each search, but is checked using the subroutine FindAns at most once; $checked[u]$ represents whether u has been checked (lines 9–10). We will describe the implementation of FindAns later. Briefly, for vertex v, FindAns either returns an answer better than T_{opt}, which is a compactness-bounded complete answer to the largest subset of Q which v is a certificate vertex for, or returns null if such a better answer does not exist.

Iteratively, vertices that are at most $\lfloor \frac{D}{2} \rfloor$ hops away from each keyword vertex are searched and checked in a best-first manner (lines 11–24). In each iteration, the pair $\langle u, v\rangle$ in PQ with the minimum priority $\texttt{pri}(u, v)$ is pulled out of PQ (line 12). We will compute $\texttt{pri}(u, v)$ later in the paper; it represents a lower bound on the number of keywords dropped by subsets of Q which v or its descendant in the search starting from u is a certificate vertex for. If this lower bound is not better than the `dor` of T_{opt}, the algorithm will be terminated because T_{opt} is guaranteed to be optimum (lines 13–14). Otherwise, v or some of its descendants may be a certificate vertex for a larger subset of Q and hence a better answer may exist. So if v has not been checked in other searches, it will be checked using FindAns which either returns a better answer T and updates T_{opt}, or returns null (lines 15–19). The search starting from u continues, and the unvisited neighbors of v are expanded (lines 20–24). By requiring $\texttt{d}(u, v) < \lfloor \frac{D}{2} \rfloor$, the search is restricted to vertices that are at most $\lfloor \frac{D}{2} \rfloor$ hops away from u.

Algorithm 2: FindAns

Input: G, $Q = \{k_1, \ldots, k_g\}$, D, v, T_{opt}.
Output: An answer T better than T_{opt}, or null if not found.

1 **for** $i \leftarrow 1$ **to** g **do**
2 $\quad U[1][i] \leftarrow \{u \in K_i : \mathtt{d}(u, v) \leq \lceil \frac{D}{2} \rceil\}$;
3 $\quad U[2][i] \leftarrow \{u \in K_i : \mathtt{d}(u, v) = \lceil \frac{D}{2} \rceil\}$;
4 $Q_1 \leftarrow \{k_i \in Q : U[1][i] \neq \emptyset\}$;
5 $Q_2 \leftarrow \{k_i \in Q : U[1][i] = U[2][i] \neq \emptyset\}$;
6 **if** D *is even* **then**
7 $\quad$ **if** $|Q_1| > |\mathtt{CQ}(T_{opt})|$ **then**
8 $\quad\quad$ **return** $\mathtt{GenAnsEven}(G, Q_1, D, v)$;
9 **if** D *is odd* **then**
10 $\quad \widehat{Q}_{\max} \leftarrow \emptyset$, $v'_{\max} \leftarrow$ null;
11 $\quad$ **foreach** $v' \in \mathtt{N}(v)$ **do**
12 $\quad\quad \widehat{Q} \leftarrow \emptyset$;
13 $\quad\quad$ **foreach** $k_i \in Q_2$ **do**
14 $\quad\quad\quad$ **if** $\exists u \in U[2][i]$, $\mathtt{d}(u, v') = \lfloor \frac{D}{2} \rfloor$ **then**
15 $\quad\quad\quad\quad \widehat{Q} \leftarrow \widehat{Q} \cup \{k_i\}$;
16 $\quad\quad$ **if** $|\widehat{Q}| > |\widehat{Q}_{max}|$ **then**
17 $\quad\quad\quad \widehat{Q}_{\max} \leftarrow \widehat{Q}$, $v'_{\max} \leftarrow v'$;
18 $\quad Q_{\max} \leftarrow (Q_1 \setminus Q_2) \cup \widehat{Q}_{\max}$;
19 $\quad$ **if** $|Q_{max}| > |\mathtt{CQ}(T_{opt})|$ **then**
20 $\quad\quad$ **return** $\mathtt{GenAnsOdd}(G, Q_{max}, D, v, v'_{max}, \widehat{Q}_{max})$;
21 **return** *null*;

Subroutine FindAns. This subroutine is presented in Algorithm 2. It employs Proposition 1 to find the largest subset of Q which v is a certificate vertex for, and then construct a compactness-bounded complete answer to this sub-query. The answer will be returned only if it is better than T_{opt}. According to Condition 1 in the proposition, Q_1 (line 4) represents the largest possible subset of Q which v can be a certificate vertex for. Q_2 (line 5) represents the largest possible $\widehat{Q}$ in Condition 2.

When D is even (line 6), Q_1 is indeed the largest subset of Q which v is a certificate vertex for. If Q_1 is larger than the set of query keywords covered by T_{opt}, we will return T constructed by the subroutine GenAnsEven (lines 7–8).

When D is odd (line 9), the vertices in Q_2 may not all satisfy Condition 2 for the same $v' \in \mathtt{N}(v)$. We look for $\widehat{Q}_{\max}$, the largest subset of Q_2 that can satisfy Condition 2 for the same neighbor of v, denoted by $v'_{\max}$ (line 10). We compute $\widehat{Q}_{\max}$ and $v'_{\max}$ by going over all the neighbors of v (lines 11–17). Finally, $Q_{\max}$ (line 18) is the largest subset of Q which v is a certificate vertex for. If $Q_{\max}$ is larger than the set of query keywords covered by T_{opt}, we will return T constructed by the subroutine GenAnsOdd (lines 19–20).

Subroutine GenAnsEven. This subroutine is presented in Algorithm 3. It employs Proposition 1 to construct a complete answer to Q_1 using its certificate vertex v under an even D. Following Condition 1 in the proposition, for each $k_i \in Q_1$, we find $u_i \in K_i$ and a shortest path P_i between u_i and v in G such that $\mathtt{len}(P_i) \leq \lceil \frac{D}{2} \rceil$ (lines 2–4). All such $|Q_1|$ shortest paths are merged into a connected subgraph T (line 5), which covers all the keywords in Q_1 and satisfies $\mathtt{diam}(T) \leq D$. Here, to ensure that T has a tree structure, when there are multiple shortest paths between two vertices, we consistently choose one throughout our algorithm, e.g., the alphabetically smallest shortest path in terms of vertex identifiers. Still, T may not be minimal when some u_i covers not only keyword k_i but also some $k_j \neq k_i$ so that u_j should be removed from T for the minimality of T. To handle this, we repeatedly remove an unnecessary leaf from T until T is minimal and hence is a complete answer to Q_1 (lines 6–7).

Subroutine GenAnsOdd. This subroutine is presented in Algorithm 4. It employs Proposition 1 to construct a complete answer to $Q_{\max}$ using its certificate vertex v and a neighbor $v'_{\max}$ thereof under an odd D. The difference between GenAnsEven and GenAnsOdd is that, following Condition 2 in the proposition, for each $k_i \in \widehat{Q}_{\max}$, we find $u_i \in K_i$ and a shortest path P'_i between u_i and $v'_{\max}$ such that $\texttt{len}(P'_i) = \lfloor \frac{D}{2} \rfloor$ (lines 3–5), and then we extend P'_i from $v'_{\max}$ to v by adding one edge to form P_i (line 6); furthermore, for each $k_i \in (Q_{\max} \setminus \widehat{Q}_{\max})$, we find $u_i \in K_i$ such that P_i connecting u_i and v satisfies $\texttt{len}(P_i) \leq \lfloor \frac{D}{2} \rfloor$ (lines 7–9). These modifications ensure that $\texttt{diam}(T) \leq D$.

Algorithm 3: GenAnsEven

```
   Input: G, Q_1, D, v.
   Output: A complete answer T to Q_1.
1  T ← a null graph;
2  foreach k_i ∈ Q_1 do
3  |  ∃u_i ∈ K_i, d(u_i, v) ≤ ⌈D/2⌉;
4  |  P_i ← a shortest path between u_i
   |     and v in G;
5  |  T ← T merged with P_i;
6  while T is not minimal do
7  |  Remove an unnecessary leaf from T;
8  return T;
```

Algorithm 4: GenAnsOdd

```
   Input: G, Q_max, D, v, v'_max, Q̂_max.
   Output: A complete answer T to Q_max.
1  T ← a null graph;
2  foreach k_i ∈ Q_max do
3  |  if k_i ∈ Q̂_max then
4  |  |  ∃u_i ∈ K_i, d(u_i, v'_max) = ⌊D/2⌋;
5  |  |  P'_i ← a shortest path between u_i
   |  |     and v'_max in G;
6  |  |  P_i ← P'_i extended by adding the
   |  |     edge between v'_max and v;
7  |  else
8  |  |  ∃u_i ∈ K_i, d(u_i, v) ≤ ⌊D/2⌋;
9  |  |  P_i ← a shortest path between u_i
   |  |     and v in G;
10 |  T ← T merged with P_i;
11 while T is not minimal do
12 |  Remove an unnecessary leaf from T;
13 return T;
```

Priority Computation. Priority $\texttt{pri}(u, v)$ represents a lower bound on the number of keywords dropped by subsets of Q which v or its descendant in the search starting from u is a certificate vertex for. A complete answer to such a sub-query satisfies $\texttt{diam} \leq D$ and should contain u. It inspires us to define the following heuristic lower bound:

$$\texttt{pri}(u, v) = |\texttt{UQ}(u, v)|, \quad \texttt{UQ}(u, v) = \{k_i \in Q : \forall u_i \in K_i,\ \texttt{d}(u, v) + \texttt{d}(v, u_i) > D\}\,. \tag{8}$$

We will later prove that this heuristic guarantees the optimality of T_{opt} returned at the end of Algorithm 1.

Further, observe that $\texttt{pri}(u, v)$ is an integer. There can be ties in the priority queue. We break ties based on the degree of v. If multiple vertices have the same value of `pri`, we will give priority to one that has the smallest degree because the cost of expanding its neighbors in Algorithm 1 (lines 21–24) is minimal.

Running Example. In Fig. 2 we illustrate the process of CORE using the example in Fig. 1 under $D = 2$. Initially, PQ is fed with five keyword vertices. In the first iteration, $\langle$`YSNP`, `YSNP`$\rangle$ has the smallest priority; in particular, `YSNP` has a smaller degree than `US`. For `YSNP`, it is checked using FindAns which computes an answer that consists of a single vertex `YSNP` and assigns this answer to T_{opt}. The only neighbor of `YSNP`, namely `MT`, is then expanded. In the second iteration,

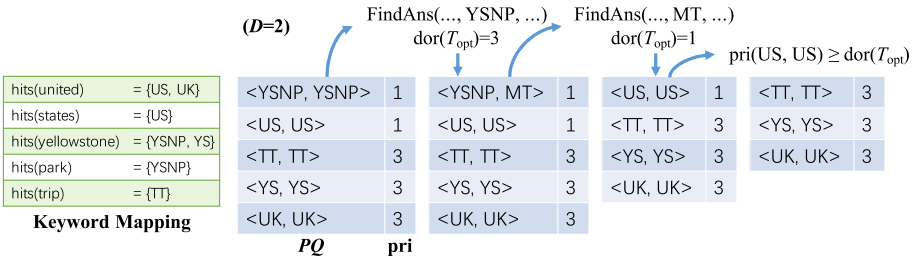


Fig. 2. A running example where vertices are written in abbreviations.

MT is checked using FindAns which computes T_1 in Fig. 1 and updates T_{opt}. In the third iteration, we terminate the search process because the priority of the head of PQ is not better than $\texttt{dor}(T_{\text{opt}})$. Therefore, T_1 is returned.

4.3 Algorithm Analysis

We analyze the correctness and time complexity of CORE.

Correctness. The following proposition gives the correctness of CORE.

Proposition 2. *CORE returns an optimum solution to the MRA problem.*

Proof. We prove by contradiction.

Assume CORE returns a suboptimal answer T_{opt} whose dor is larger than that of an optimum answer T^*. Below we will show that when CORE is terminated, T^* should have been found, leading to a contradiction.

Let v be a certificate vertex for $\texttt{CQ}(T^*)$. Without loss of generality, assume that T^* is composed of shortest paths between keyword vertices and v in G, i.e., T^* can be constructed by calling subroutine GenAnsEven or GenAnsOdd. Following Condition 3 in Proposition 1, let u be a keyword vertex in T^* that satisfies $\texttt{d}(u, v) \leq \lfloor \frac{D}{2} \rfloor$, and let P_{uv} be the unique path between u and v in T^*, which is a shortest path in G and hence satisfies $\texttt{len}(P_{uv}) \leq \lfloor \frac{D}{2} \rfloor$.

Following Condition 1 in Proposition 1, $\forall k_i \in \texttt{CQ}(T^*)$ that is not covered by u, $\exists u_i \in K_i$ such that u_i is in T^* and $\texttt{d}(u_i, v) \leq \lceil \frac{D}{2} \rceil$. Let P_{vu_i} be the unique path between v and u_i in T^*, which is a shortest path in G and hence satisfies $\texttt{len}(P_{vu_i}) \leq \lceil \frac{D}{2} \rceil$. For every vertex w in P_{uv}, we have

$$\texttt{d}(u, w) + \texttt{d}(w, u_i) \leq \texttt{len}(P_{uv}) + \texttt{len}(P_{vu_i}) \leq \left\lfloor \frac{D}{2} \right\rfloor + \left\lceil \frac{D}{2} \right\rceil = D\,. \tag{9}$$

Therefore, $k_i \notin \texttt{UQ}(u, w)$ and hence $\texttt{CQ}(T^*) \cap \texttt{UQ}(u, w) = \emptyset$. It gives us

$$\texttt{pri}(u, w) = |\texttt{UQ}(u, w)| \leq |Q \setminus \texttt{CQ}(T^*)| = \texttt{dor}(T^*) < \texttt{dor}(T_{\text{opt}})\,. \tag{10}$$

Due to the best-first search strategy, it is impossible that CORE returns T_{opt} before checking each w in P_{uv} from u to v using subroutine FindAns, which would have found T^* when checking v, leading to a contradiction.

□

Time Complexity. Recall that $|V| = n$, $|E| = m$, $|Q| = g$. We precompute a hub labeling [23]; this index structure supports computing distance in $O(n)$ and computing a shortest path in $O(n \log n)$. The run time of CORE consists of

- $O(ng)$ for collecting query vertices,
- $O(n(n+m))$ for $O(n)$ searches starting from different keyword vertices,
- $O(n^4 g)$ for computing priority for $O(n)$ vertices in $O(n)$ searches,
- $O(n^2 \log n)$ for $O(n^2)$ insert/pull operations over a Fibonacci heap for priority queue, and
- $O(n^2(n+m)g)$ for $O(n)$ calls of FindAns, including $O(n^3 g)$ for computing U, $O(n^2 mg)$ for computing distances to v', and $O(n^2 g \log n)$ for calling GenAnsEven or GenAnsOdd.

Therefore, the total run time of CORE is in $O(n^4 g)$, i.e., the MRA problem admits a polynomial-time solution. As we will see in the next section, the run time in practice is much faster than $O(n^4 g)$ because: g is commonly much smaller than n, distance and shortest path computation based on a hub labeling is nearly in constant time, and the search process is usually terminated early.

5 Experiments

We carried out experiments to verify three research hypotheses (RH). First, compared with traditional GST-based answers, our approach trades off answer completeness for compactness. We showed the necessity of this trade-off (**RH1**). We did not evaluate the effectiveness of compact answers to human users in exploratory search because it has been demonstrated in [5]. Still, we showed that the completeness of our computed answers is very high (**RH2**), thereby having little influence on effectiveness. Last but not least, we demonstrated the efficiency of CORE and showed that it significantly outperformed CertQR+ [17] (**RH3**).

Table 1. KGs and keyword queries.

	KG		Keyword query		
	Vertices (n)	Edges (m)	Quantity	Size (g): avg;	Max
MONDIAL (about geography)	8,478	34,868	40	2.03;	4
LinkedMDB (about movies)	1,326,784	2,132,796	200	5.38;	10
DBpedia (an encyclopedia)	5,356,286	17,494,749	438	3.92;	10

5.1 Experiment Setup

We used a 3.10 GHz CPU with 80 GB memory for Java programs.

KGs. We used three popular KGs in RDF format: MONDIAL,[2] LinkedMDB,[3] and DBpedia.[4] Their sizes (n and m) are shown in Table 1.

Keyword Queries. We used public queries. For MONDIAL we reused 40 keyword queries in [6]. For LinkedMDB we randomly sampled 200 natural language questions from [20] and transformed each question into a keyword query by removing stop words and punctuation marks. For DBpedia we reused 438 keyword queries in [11]. Their sizes (g) are shown in Table 1.

Keyword Mapping. Our retrieval function `hits` maps each keyword k to vertices K whose human-readable names (`rdfs:label`) contain k. To test scalability, we set the maximum allowable number of keyword vertices retrieved by `hits` to different values: $|K| \leq 1$, $|K| \leq 10$, and $|K| \leq 100$.

Compactness Bound. Following [3,4], we used two settings: $D \in \{3, 4\}$.

Other experiment settings, e.g., using more advanced `hits` functions, would be left for the future work.

5.2 Baselines

We implemented two baseline methods for comparison. We did not compare with semantic parsing such as [10,22,25] since we focused on exploratory search tasks.

CertQR+. CORE is the first algorithm for the MRA problem. However, we could adapt CertQR+ [17] to MRA. CertQR+ originally addresses a special case of MRA where each keyword is mapped to a single vertex, and it only computes a minimally relaxed query rather than an answer. In our adaptation, for $Q = \{k_1, \ldots, k_g\}$ in MRA, we: (1) for each combination of keyword vertices in $K_1 \times \cdots \times K_g$, feed the combined set of keyword vertices as input into CertQR+, which outputs a minimally relaxed set of keyword vertices, (2) choose the maximum output set over all the combinations, and (3) find a complete answer to connect the chosen set of keyword vertices using a state-of-the-art search algorithm [3,4]. Therefore, we remark that the performance of this baseline relies on the search algorithm used, which may affect the fairness of its comparison with CORE.

GST-Based Answers. We implemented an algorithm [16] to compute traditional GST-based answers. We assigned unit weights to the edges of KGs. We also tried several other weighting schemes [5] but obtained similar findings.

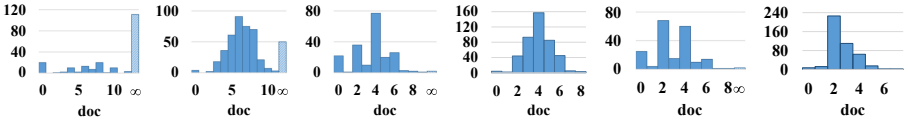
5.3 Experiment 1: Compactness of GST-Based Answers (RH1)

To show the necessity of trading off answer completeness for compactness (RH1), we computed the degree of compactness (`doc`) of traditional GST-based answers

[2] http://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.rdf.

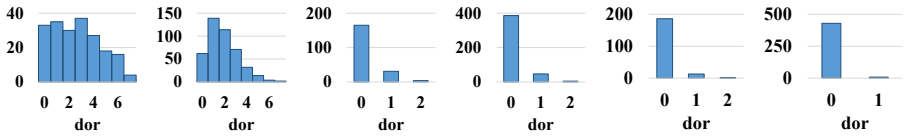
[3] http://www.cs.toronto.edu/~oktie/linkedmdb/linkedmdb-latest-dump.zip.

[4] http://downloads.dbpedia.org/2016-10/core-i18n/en/mappingbased_objects_en.tql.bz2.



(a) $|K| \leq 1$, LinkedMDB (b) $|K| \leq 1$, DBpedia (c) $|K| \leq 10$, LinkedMDB (d) $|K| \leq 10$, DBpedia (e) $|K| \leq 100$, LinkedMDB (f) $|K| \leq 100$, DBpedia

Fig. 3. Distribution of `doc` of GST-based answers for all the queries.



(a) $|K| \leq 1$, LinkedMDB (b) $|K| \leq 1$, DBpedia (c) $|K| \leq 10$, LinkedMDB (d) $|K| \leq 10$, DBpedia (e) $|K| \leq 100$, LinkedMDB (f) $|K| \leq 100$, DBpedia

Fig. 4. Distribution of `dor` of relaxable answers for all the queries under $D = 4$.

according to Eq. (2). The distributions of `doc` (i.e., diameter) of answers for all the queries over two large KGs are presented in Fig. 3. When $|K| \leq 1$, GST-based answers cannot be found (i.e., $\texttt{doc} = \infty$) for 56% of queries over LinkedMDB and for 11% on DBpedia because keyword vertices are not pairwise connected. Even for $|K| \leq 100$, it happens for a few queries over LinkedMDB. Apart from that, `doc` is as large as up to 8 on LinkedMDB and to 7 on DBpedia. When $|K| \leq 10$, we observe $\texttt{doc} > 4$ for 27% of queries over LinkedMDB and for 32% on DBpedia. These many failing queries and structurally large answers are not favored by human users [5], thereby demonstrating the necessity of trading off answer completeness for compactness (RH1).

5.4 Experiment 2: Completeness of Relaxable Answers (RH2)

To show the completeness of our relaxable answers (RH2), we computed their degree of relaxation (`dor`) according to Eq. (1). The distributions of `dor` (i.e., number of dropped query keywords) of answers for all the queries over two large KGs under $D = 4$ are presented in Fig. 4. Only in the extreme case of $|K| \leq 1$, the `dor` of our answers is not satisfying due to the disconnectivity between keyword vertices. At least 3 keywords have to be dropped (i.e., $\texttt{dor} \geq 3$) in more than half of the queries over LinkedMDB, and at least 2 (i.e., $\texttt{dor} \geq 2$) on DBpedia. With $|K|$ increased to 10 and 100, relaxation is not needed (i.e., $\texttt{dor} = 0$) for over 80% and 90% of the queries, respectively. When relaxation takes place (i.e., $\texttt{dor} > 0$), typically only 1 and very rarely 2 keywords are dropped. These results show that the completeness of our relaxable answers is very high in normal cases (RH2).

Table 2. Number of completed queries (CP), timeout exceptions (TO), and mean run time (ms) for a query.

			MONDIAL			LinkedMDB			DBpedia		
			CP	TO	Time	CP	TO	Time	CP	TO	Time
$\|K\| \leq 1$	$D = 3$	CertQR+	40	0	55.60	200	0	30.91	438	0	62.65
		CORE	40	0	1.88	200	0	5.38	438	0	27.87
	$D = 4$	CertQR+	40	0	15.13	200	0	127.36	438	0	221.68
		CORE	40	0	2.80	200	0	96.33	438	0	193.47
$\|K\| \leq 10$	$D = 3$	CertQR+	40	0	85.10	125	75	436,566.95	321	117	327,641.86
		CORE	40	0	3.95	200	0	23.45	438	0	1,017.26
	$D = 4$	CertQR+	40	0	6.80	113	87	475,926.77	319	119	319,777.02
		CORE	40	0	2.23	200	0	179.17	438	0	4,181.92
$\|K\| \leq 100$	$D = 3$	CertQR+	40	0	39.23	82	118	611,297.51	211	227	547,598.42
		CORE	40	0	5.15	200	0	506.62	437	1	6,107.36
	$D = 4$	CertQR+	40	0	5.70	90	110	561,819.84	253	185	453,752.70
		CORE	40	0	4.13	200	0	324.22	437	1	4,307.86

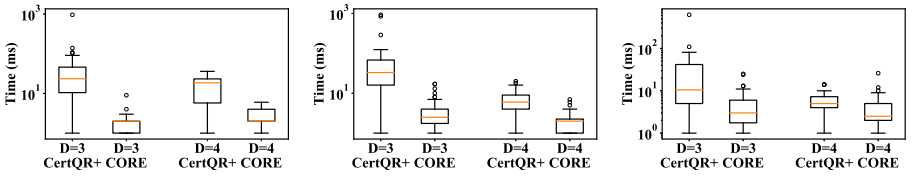
5.5 Experiment 3: Efficiency of CORE (RH3)

To show the efficiency of CORE (RH3), we compared the run time of CORE and CertQR+. We set a timeout of 1,000 s. Each run of an algorithm for a query that reached timeout was terminated, and its run time was defined to be the timeout value. Therefore, the longest run time reported below is at most 1,000 s. The overall results are presented in Table 2. When $|K| \leq 1$, CertQR+ and CORE complete all the queries before timeout. CORE is faster because it fuses query relaxation and answer generation into a single search process. When $|K| \leq 10$, CertQR+ is called up to 10^g times for each query. On LinkedMDB and DBpedia, it reaches timeout for many queries and uses several hundred seconds. CORE never reaches timeout and performs at least two orders of magnitude faster because it fuses at most $10g$ searches into a single best-first search process. For $|K| \leq 100$, the comparison is similar.

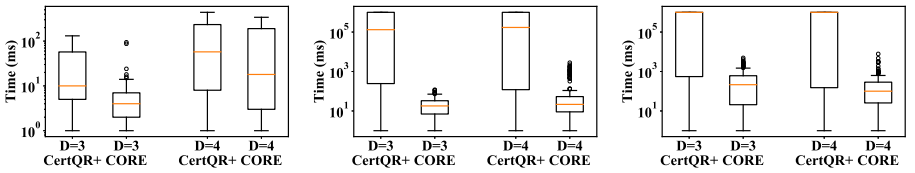
Furthermore, we visualize the distributions of run time for all the queries as box plots in Fig. 5. CORE uses less than 1 s for most queries, except for a small number of outliers. Comparing the median run time of CORE and CertQR+ represented by vertical lines going through boxes, on large KGs (LinkedMDB and DBpedia), for non-trivial settings of the retrieval function ($|K| \leq 10$ and $|K| \leq 100$), CORE is about 2–4 orders of magnitude faster than CertQR+. All the above results demonstrate the efficiency of CORE, which significantly outperforms CertQR+ (RH3).

5.6 Experiment 4: Effectiveness in Lookup Tasks

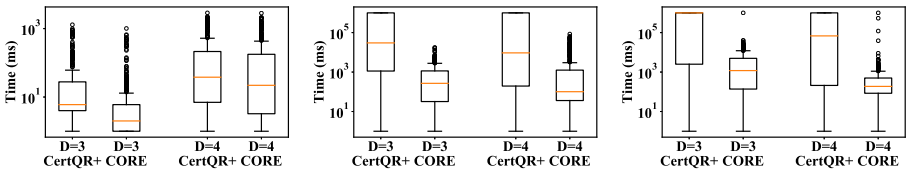
Although our approach is mainly developed for supporting exploratory search, one may be interested in its effectiveness in lookup tasks. We conducted the following experiment to accommodate such interests. Since CORE only returns an optimum answer, we computed the precision at 1 (P@1) of the answers returned



(a) MONDIAL, $|K| \leq 1$ (b) MONDIAL, $|K| \leq 10$ (c) MONDIAL, $|K| \leq 100$



(d) LinkedMDB, $|K| \leq 1$ (e) LinkedMDB, $|K| \leq 10$ (f) LinkedMDB, $|K| \leq 100$



(g) DBpedia, $|K| \leq 1$ (h) DBpedia, $|K| \leq 10$ (i) DBpedia, $|K| \leq 100$

Fig. 5. Distribution of run time for all the queries.

Table 3. Mean P@1 for a lookup query on DBpedia.

		CertQR+	CORE	GST-Based Answers	FSDM
$\|K\| \leq 1$	$D = 3$	0.17	0.17	0.27	0.98
	$D = 4$	0.23	0.23		
$\|K\| \leq 10$	$D = 3$	0.21	0.23	0.31	
	$D = 4$	0.24	0.28		
$\|K\| \leq 100$	$D = 3$	0.17	0.29	0.30	
	$D = 4$	0.21	0.23		

by each algorithm. We compared CORE with CertQR+ and traditional GST-based answers. We conducted this experiment on DBpedia since the keyword queries we used for this dataset were originally extracted from entity lookup tasks with gold-standard answers [11] so that we could compute precision. Specifically, we defined P@1 = 1 if a computed answer contained a gold-standard answer entity, otherwise P@1 = 0. The mean P@1 for a lookup query over DBpedia is presented in Table 3. CORE and CertQR+ show similar results since they essentially aim at the same goal. For compactness, they trade off answer completeness

and hence we observe a very small loss of P@1 compared with traditional GST-based answers (up to 0.29 versus up to 0.31). However, none of these methods are comparable with those specifically designed for lookup tasks, and such comparisons would be unfair. For example, FSDM [27] as a state-of-the-art method for entity lookup reaches P@1 = 0.98.

6 Conclusion and Future Work

We have presented the first study of generating compact and relaxable answers for exploratory keyword search over KGs. With real KGs and queries, we showed the necessity of trading off answer completeness for compactness, and showed the high completeness of answers we maintained in practice. Our polynomial-time best-first search algorithm CORE for solving the new MRA problem builds on a theoretical result proved in the paper. Experiments demonstrated that CORE was fast and significantly outperformed its previous version CertQR+.

In the future, we plan to incorporate vertex and edge weights into our approach to exploit the various semantics of different types of entities and relations. By computing weights to represent the semantic relevance of an answer to a keyword query, our approach can be extended to better support lookup tasks.

Acknowledgments. This work was supported in part by the National Key R&D Program of China (2018YFB1005100), in part by the NSFC (61772264), and in part by the Six Talent Peaks Program of Jiangsu Province (RJFW-011).

References

1. Chen, C., Wang, G., Liu, H., Xin, J., Yuan, Y.: SISP: a new framework for searching the informative subgraph based on PSO. In: CIKM, pp. 453–462 (2011)
2. Cheng, G.: Relationship search over knowledge graphs. ACM SIGWEB Newsl. **2020**(Summer), 3 (2020)
3. Cheng, G., Liu, D., Qu, Y.: Efficient algorithms for association finding and frequent association pattern mining. In: Groth, P., et al. (eds.) ISWC 2016. LNCS, vol. 9981, pp. 119–134. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46523-4_8
4. Cheng, G., Liu, D., Qu, Y.: Fast algorithms for semantic association search and pattern mining. IEEE Trans. Knowl. Data Eng. Early Access, 1–13 (2019). https://doi.org/10.1109/TKDE.2019.2942031
5. Cheng, G., Shao, F., Qu, Y.: An empirical evaluation of techniques for ranking semantic associations. IEEE Trans. Knowl. Data Eng. **29**(11), 2388–2401 (2017)
6. Coffman, J., Weaver, A.C.: An empirical performance evaluation of relational keyword search techniques. IEEE Trans. Knowl. Data Eng. **26**(1), 30–42 (2014)
7. Ding, B., Yu, J.X., Wang, S., Qin, L., Zhang, X., Lin, X.: Finding top-k min-cost connected trees in databases. In: ICDE, pp. 836–845 (2007)
8. Dosso, D., Silvello, G.: A scalable virtual document-based keyword search system for RDF datasets. In: SIGIR, pp. 965–968 (2019)
9. Elbassuoni, S., Ramanath, M., Weikum, G.: Query relaxation for entity-relationship search. In: Antoniou, G., et al. (eds.) ESWC 2011. LNCS, vol. 6644, pp. 62–76. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21064-8_5

10. Fu, H., Anyanwu, K.: Effectively interpreting keyword queries on RDF databases with a rear view. In: ISWC, pp. 193–208 (2011)
11. Hasibi, F., et al.: DBpedia-entity v2: a test collection for entity search. In: SIGIR, pp. 1265–1268 (2017)
12. Huang, Z., Li, S., Cheng, G., Kharlamov, E., Qu, Y.: MiCRon: making sense of news via relationship subgraphs. In: CIKM, pp. 2901–2904 (2019)
13. Ihler, E.: The complexity of approximating the class Steiner tree problem. In: Schmidt, G., Berghammer, R. (eds.) WG 1991. LNCS, vol. 570, pp. 85–96. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-55121-2_8
14. Kasneci, G., Elbassuoni, S., Weikum, G.: MING: mining informative entity relationship subgraphs. In: CIKM, pp. 1653–1656 (2009)
15. Li, G., Ooi, B.C., Feng, J., Wang, J., Zhou, L.: EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In: SIGMOD, pp. 903–914 (2008)
16. Li, R., Qin, L., Yu, J.X., Mao, R.: Efficient and progressive group steiner tree search. In: SIGMOD, pp. 91–106 (2016)
17. Li, S., Cheng, G., Li, C.: Relaxing relationship queries on graph data. J. Web Semant. **61–62**, 100557 (2020)
18. Lu, X., Pramanik, S., Roy, R.S., Abujabal, A., Wang, Y., Weikum, G.: Answering complex questions by joining multi-document evidence with quasi knowledge graphs. In: SIGIR, pp. 105–114 (2019)
19. Marchionini, G.: Exploratory search: from finding to understanding. Commun. ACM **49**(4), 41–46 (2006)
20. Miller, A.H., Fisch, A., Dodge, J., Karimi, A., Bordes, A., Weston, J.: Key-value memory networks for directly reading documents. In: EMNLP, pp. 1400–1409 (2016)
21. Poulovassilis, A., Selmer, P., Wood, P.T.: Approximation and relaxation of semantic web path queries. J. Web Semant. **40**, 1–21 (2016)
22. Shekarpour, S., Ngomo, A.N., Auer, S.: Question answering on interlinked data. In: WWW, pp. 1145–1156 (2013)
23. Shi, Y., Cheng, G., Kharlamov, E.: Keyword search over knowledge graphs via static and dynamic hub labelings. In: WWW, pp. 235–245 (2020)
24. Tong, H., Faloutsos, C.: Center-piece subgraphs: problem definition and fast solutions. In: KDD, pp. 404–413 (2006)
25. Tran, T., Herzig, D.M., Ladwig, G.: SemSearchPro - using semantics throughout the search process. J. Web Semant. **9**(4), 349–364 (2011)
26. Wang, M., Wang, R., Liu, J., Chen, Y., Zhang, L., Qi, G.: Towards empty answers in SPARQL: approximating querying with RDF embedding. In: Vrandečić, D., et al. (eds.) ISWC 2018. LNCS, vol. 11136, pp. 513–529. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00671-6_30
27. Zhiltsov, N., Kotov, A., Nikolaev, F.: Fielded sequential dependence model for ad-hoc entity retrieval in the web of data. In: SIGIR, pp. 253–262 (2015)