Keyword-Based Knowledge Graph Exploration Based on Quadratic Group Steiner Trees

Yuxuan Shi[1,2] , **Gong Cheng**[1*] , **Trung-Kien Tran**[2] , **Jie Tang**[3] and **Evgeny Kharlamov**[2,4]
[1]State Key Laboratory for Novel Software Technology, Nanjing University, China
[2]Bosch Center for Artificial Intelligence, Renningen, Germany
[3]Department of Computer Science and Technology, Tsinghua University, China
[4]Department of Informatics, University of Oslo, Norway
yxshi@smail.nju.edu.cn, gcheng@nju.edu.cn, {trungkien.tran, evgeny.kharlamov}@de.bosch.com, jietang@tsinghua.edu.cn

Abstract

Exploring complex structured knowledge graphs (KGs) is challenging for non-experts as it requires knowledge of query languages and the underlying structure of the KGs. Keyword-based exploration is a convenient paradigm, and computing a group Steiner tree (GST) as an answer is a popular implementation. Recent studies suggested improving the cohesiveness of an answer where entities have small semantic distances from each other. However, how to efficiently compute such an answer is open. In this paper, to model cohesiveness in a generalized way, the quadratic group Steiner tree problem (QGSTP) is formulated where the cost function extends GST with quadratic terms representing semantic distances. For QGSTP we design a branch-and-bound best-first (B^{3}F) algorithm where we exploit combinatorial methods to estimate lower bounds for costs. This exact algorithm shows practical performance on medium-sized KGs.

1 Introduction

A knowledge graph (KG) is a collection of annotated and interconnected entities. By offering a flexible way to structure and integrate information, KGs have been widely used in AI applications. Consider a sample KG in Fig. 1 which describes various relations between physicists, institutions, and cities.

Task. When a user such as a data journalist looks into a machine-readable KG, a fundamental task is *KG exploration*. Users often explore a complex structured KG with difficulty. For those who lack expertise in formulating formal queries or whose vague information needs can hardly be formalized, machine-assisted KG exploration is crucial [Lissandrini *et al.*, 2020]. While various AI-empowered methods have been devised to support this task, such as KG summarization [Cheng *et al.*, 2016; Cheng *et al.*, 2017a] and relevance-based entity recommendation [Gu *et al.*, 2019; Zhou *et al.*, 2020], the most commonly adopted user interface is based on keyword queries: a user explores a KG by easily

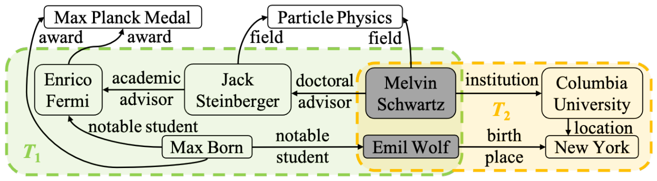


Figure 1: An example KG and two extracted subgraphs: a semantically cohesive subgraph T_1 and a semantically incohesive subgraph T_2 for answering the query "melvin schwartz, emil wolf".

submitting a set of keywords expressing an information need. State-of-the-art methods [Shi *et al.*, 2020] efficiently find and present an optimum subgraph extracted from the KG that contains all the keywords in the query. For example, Fig. 1 shows two extracted subgraphs T_1 and T_2 for answering the query "melvin schwartz, emil wolf". An optimum subgraph will be computed and returned as an answer.

Motivation. In the literature it was standard to assign weights to vertices or edges in a KG and then compute a minimum-weight connected subgraph that contains all the keywords in a query as an optimum answer [Ding *et al.*, 2007; Li *et al.*, 2016; Shi *et al.*, 2020], aka a group Steiner tree (GST) [Ihler, 1991]. The underlying assumption is: an aggregation of salient graph elements (i.e., with small weights) constitute a good answer. However, the assumption has recently been challenged: by analyzing industrial KGs, it was suggested to improve the *cohesiveness* of an answer where entities (i.e., vertices) have small *semantic distances* from each other [Cheng and Kharlamov, 2017], e.g., having similar types, textual annotations, or topics. The suggestion has been supported by empirical studies focusing on different concrete measures of semantic distance [Cheng *et al.*, 2017b; Bryson *et al.*, 2020]. As an example, T_2 in Fig. 1 is an uninteresting mixture of disparate entities despite their salience, while T_1 cohesively and meaningfully describes physicists and their advisors. Although the effectiveness of such cohesive answers in KG exploration has been accepted, one problem remains: *how to efficiently compute a cohesive answer*. This is our research target in this paper.

Quadratic GST problem. The standard GST problem minimizes the total weight of the vertices or edges in a sub-

[*]Contact Author

graph. This cost function cannot model implicit relationships between graph elements such as semantic distances. Recently we have extended it to calculate the total weight of graph elements and their semantic distances [Shi *et al.*, 2021]. This extended optimization problem is referred to as the *quadratic group Steiner tree problem* (QGSTP) since semantic distance is a quadratic term involving two graph elements. QGSTP is fundamentally more difficult than the GST problem. Indeed, quadratic terms are harder to optimize. Moreover, weights and semantic distances require joint optimization while their values could be independent from each other.

Algorithm B^{3}F. To solve QGSTP, we design an exact algorithm named B^{3}F that guarantees to find an optimum solution. B^{3}F is an iterative **b**ranch-and-**b**ound algorithm and it performs **b**est-**f**irst search in each iteration. We define a branch of the search space as a set of subgraphs that subsume a common path. We leverage the path to develop a lower bound estimation for the costs of the subgraphs in the branch, where we manipulate and integrate combinatorial methods including set covering and maximum matching.

Contributions. Below we summarize our contributions.

- We design the first exact algorithm B^{3}F for QGSTP. We significantly prune the search space of B^{3}F by devising an effective lower bound estimation for costs.
- We conduct extensive experiments on public KGs and queries. B^{3}F computes more cohesive answers than the classical GST and it runs in comparable time.

2 Related Work

2.1 GST-Based Exploration

Keyword-based KG exploration has been commonly formulated as a GST problem to extract a minimum-weight tree that contains all the keywords in a query [Ding *et al.*, 2007; Li *et al.*, 2016; Shi *et al.*, 2020], or as a variant of this problem [Kargar and An, 2011; Le *et al.*, 2014; Yang *et al.*, 2019]. They assume that aggregating salient graph elements with small weights produces a good answer, but they ignore the semantic relationships between elements. Indeed, a set of salient elements may not form a semantically cohesive subgraph and hence not a meaningful answer.

Our work extends this line of research by incorporating semantic distances between elements and formulating QGSTP to compute cohesive answers. Our cost function contains quadratic terms that are harder to optimize.

2.2 Cohesiveness Computation

The above shortcoming of GST-based exploration has been noticed when researchers processed industrial KGs and they suggested computing cohesive answers [Cheng and Kharlamov, 2017]. The suggestion is supported by a user study where a concrete measure of semantic distance was implemented based on entity types [Cheng *et al.*, 2017b]. By comparing pairs of answers, users preferred more cohesive answers where entities have similar types. Another implemented measure of semantic distance relies on random walk with restart [Bryson *et al.*, 2020], to which a specific heuristic algorithm was presented to compute such cohesive answers.

Our work addresses efficient computation of cohesive answers, which is not considered in [Cheng and Kharlamov, 2017; Cheng *et al.*, 2017b]. Our algorithm computes an optimum solution to QGSTP while the algorithms in [Bryson *et al.*, 2020; Shi *et al.*, 2021] cannot guarantee to find an optimum solution. Regarding other studies of "cohesiveness" in graph search and exploration [Dass *et al.*, 2015; Zhu *et al.*, 2018], their definitions are orthogonal to ours.

2.3 Query Interpretation

To interpret a keyword query (or a natural language question) for KG exploration, one popular approach is to transform the keyword query into a formal query and then execute the formal query over a standard query engine [Tran *et al.*, 2009; Pound *et al.*, 2012; Shekarpour *et al.*, 2013; Sun *et al.*, 2020]. The formal query is typically a graph pattern that formally represents the meaning of the keyword query and is matched with the KG to retrieve answers.

Our work is suitable for vague information needs that can hardly be interpreted as a precise graph pattern, e.g., exploring relationships between a set of entities [Cheng, 2020; Li *et al.*, 2020; Cheng *et al.*, 2020]. The above transformation-based methods are targeted at information lookup where there is a specific query intent that can be represented by a graph pattern. The two lines of research are complementary.

3 Problem

3.1 Preliminaries

A KG is a directed graph denoted by $G = \langle V, E \rangle$, where V is a set of vertices representing entities, and $E \subseteq V \times V$ is a set of directed edges representing relations between entities. We allow the edges in a path/tree to be in different directions.

Let $\mathbb{K}$ be the set of all keywords. A matching function $\texttt{hits} : \mathbb{K} \mapsto 2^V$ maps each keyword to a subset of vertices. The concrete implementation of hits, e.g., either syntactically or semantically matching keywords with vertex annotations, is not our focus. For simplicity our formulation omits edge mapping but it can be transformed into vertex mapping: for each edge (u, v) we create a vertex w with the annotation of (u, v); then replace (u, v) by two edges (u, w) and (w, v). A query $Q \subseteq \mathbb{K}$ is a set of g keywords $Q = \{k_1, \ldots, k_g\}$. To ease the notation we write $\texttt{hits}(k_i)$ as K_i for $1 \leq i \leq g$. We call K_i keyword vertices.

Similar to the GST problem [Ihler, 1991], given $G = \langle V, E \rangle$ we define a *feasible answer* to Q as a subgraph of G denoted by $T = \langle V_T, E_T \rangle$ such that: (1) T is connected, (2) T contains at least one keyword vertex from each K_i for $1 \leq i \leq g$, and (3) T is structurally minimal for (1) and (2), i.e., none of its proper subgraphs satisfy both (1) and (2), so T is a tree where leaf vertices must be keyword vertices. For example, T_1 and T_2 in Fig. 1 are two feasible answers to the query "melvin schwartz, emil wolf".

3.2 Problem Formulation

Let $\texttt{wt}(v)$ be the non-negative *weight* of vertex v; small wt represents salience. Let $\texttt{sd}(v_i, v_j)$ be the non-negative *semantic distance* between vertices v_i and v_j; small sd represents cohesiveness. The concrete computation of wt and sd is independent from our approach. The *cost* of a feasible answer $T = \langle V_T, E_T \rangle$ is the total weight of its vertices and their semantic distances:

$$\texttt{cost}(T) = \alpha \sum_{v \in V_T} \texttt{wt}(v) + (1-\alpha) \sum_{\substack{v_i, v_j \in V_T \\ i<j}} \texttt{sd}(v_i, v_j)\,, \tag{1}$$

where $\alpha \in [0, 1]$ is a parameter. An *optimum answer* is a feasible answer that minimizes $\texttt{cost}$. This optimization problem is referred to as the *quadratic group Steiner tree problem* (QGSTP) [Shi *et al.*, 2021]. It extends the cost function of the vertex-weighted GST problem [Ihler, 1991; Klein and Ravi, 1995] by introducing a quadratic term $\texttt{sd}(v_i, v_j)$ representing an extra cost that will be paid if two vertices v_i and v_j are both included in T.

3.3 Problem Complexity

Theorem 1. *QGSTP is an NP-hard optimization problem.*

Proof. We can reduce two NP-hard problems to QGSTP.

It is straightforward to reduce the vertex-weighted Steiner tree problem to an instance of QGSTP where $\alpha = 1$ and $|K_i| = 1$ for $1 \leq i \leq g$.

Alternatively, we reduce the quadratic set cover problem to an instance of QGSTP where $\alpha = 1 - \alpha$ and vertices in G are pairwise adjacent. Thus, sets containing an element correspond to vertices mapped to from a keyword. $\square$

Compared with the vertex-weighted GST problem, QGSTP is more difficult due to the extended optimization of the quadratic $\texttt{sd}$ which is independent from $\texttt{wt}$. For example, vertices that are adjacent in KGs can be semantically distant.

4 Approach

QGSTP is an NP-hard optimization problem, for which a brute-force algorithm would be computationally prohibitive even on small KGs. We present a **B**ranch-and-**B**ound **B**est-**F**irst algorithm, abbreviated to B^{3}F. It is the first exact algorithm for QGSTP. As we will see in the experiments it shows promising performance on medium-sized KGs.

Before presenting B^{3}F, we define some terms. For path p, its length $\texttt{len}(p)$ is the number of edges in p. The distance between two vertices is the length of a shortest path connecting them. The eccentricity of vertex v is the greatest distance between v and other vertices. The radius and diameter of graph T, denoted by $\texttt{rad}(T)$ and $\texttt{diam}(T)$, are the minimum and maximum eccentricity of the vertices in T, respectively. A central vertex is a vertex of minimum eccentricity. For two graphs T and T', we write $T \preceq T'$ if T is a subgraph of T'.

4.1 Basic Idea

In general, B^{3}F iteratively explores the search space in a top-down manner (i.e., branching). Branches where feasible answers cannot be better than the optimal answer found so far are discarded (i.e., bounding). In each iteration, the most promising branch is explored (i.e., best-first).

Algorithm 1 B^{3}F

```
Input: $G = \langle V, E \rangle$, $Q = \{k_1, \ldots, k_g\}$, integer $d$
Output: $T_{\text{opt}}$
 1: $T_{\text{opt}} \leftarrow$ null;  /* We define $\texttt{cost}(\text{null}) = \infty$. */
 2: $PQ \leftarrow$ an empty min-priority queue of paths;
 3: for all $v \in \bigcup_{i=1}^{g} K_i$ do
 4:     $PQ$.insert($v$);
 5: for $i = 1$ to $g$ do
 6:     $P_i \leftarrow$ an empty set of paths;
 7: while $PQ$ is not empty do
 8:     $p_{\text{top}} \leftarrow PQ$.pull();
 9:     if $\overline{\texttt{cost}}(p_{\text{top}}) \geq \texttt{cost}(T_{\text{opt}})$ then
10:         break the while loop;
11:     for all $k_i \in \texttt{QK}(V_{p_{\text{top}}})$ do
12:         $P_i \leftarrow P_i \cup \{p_{\text{top}}\}$;
13:     for all $\langle p_1, \ldots, p_g \rangle \in P_1 \times \cdots \times P_g$ containing $p_{\text{top}}$
        and having a common end vertex do
14:         $T \leftarrow$ merge $p_1, \ldots, p_g$;
15:         if $T$ is structurally minimal then
16:             if $\texttt{cost}(T) < \texttt{cost}(T_{\text{opt}})$ then
17:                 $T_{\text{opt}} \leftarrow T$;
18:     if $\texttt{len}(p_{\text{top}}) < d$ then
19:         for all $v \in V$ that is adjacent from/to the end vertex
            of $p_{\text{top}}$ and is outside $p_{\text{top}}$ do
20:             $p \leftarrow$ grow $p_{\text{top}}$ to $v$ with one edge;
21:             $PQ$.insert($p$);
22: return $T_{\text{opt}}$
```

Answer construction. B^{3}F constructs a feasible answer by merging g paths $p_1, \ldots, p_g$ where each p_i starts from a keyword vertex in K_i and they end at a common vertex. For example, T_1 in Fig. 1 is constructed by merging two paths:

$$\text{M. Schwartz} \xrightarrow{\text{doctoral advisor}} \text{J. Steinberger} \xrightarrow{\text{academic advisor}} \text{E. Fermi}$$

$$\text{E. Wolf} \xleftarrow{\text{notable student}} \text{M. Born} \xrightarrow{\text{notable student}} \text{E. Fermi}$$

Branching. Each path p defines a branch of the search space where each feasible answer T satisfies $p \preceq T$. A longer path defines a smaller search space; thus, path growing corresponds to branching. B^{3}F iteratively enumerates paths that grow from keyword vertices. For each enumerated path p, we estimate $\overline{\texttt{cost}}(p)$. It is a lower bound for the costs of the feasible answers in the branch defined by p:

$$\overline{\texttt{cost}}(p) \leq \min_{T:\, p \preceq T} \texttt{cost}(T)\,. \tag{2}$$

Best-first search and bounding. B^{3}F performs best-first search. In each iteration, it processes an enumerated path p_{top} that minimizes $\overline{\texttt{cost}}$. If $\overline{\texttt{cost}}(p_{\text{top}}) \geq \texttt{cost}(T_{\text{opt}})$ where T_{opt} denotes the optimal answer found so far, B^{3}F will be terminated and return T_{opt}; such bounding discards all feasible answers in the unprocessed branches. If $\overline{\texttt{cost}}(p_{\text{top}}) < \texttt{cost}(T_{\text{opt}})$, new feasible answers will be constructed by merging p_{top} with other processed paths that end at the same vertex as p_{top}; then T_{opt} may be updated. B^{3}F completes the current iteration by growing p_{top} to enumerate longer paths.

4.2 Algorithm B^{3}F

B^{3}F is detailed in Algorithm 1. Following common practice [Kacholia *et al.*, 2005; Cheng *et al.*, 2017b], we introduce an extra input d representing the largest allowed depth of search to prevent large subgraphs (i.e., $\mathtt{diam} > 2d$) which are not favored by users [Cheng *et al.*, 2017b].

T_{opt} denotes the optimal answer found so far (line 1). Paths to be processed are kept in priority queue PU and sorted by their $\overline{\mathtt{cost}}$ values in ascending order (line 2); we will elaborate the computation of $\overline{\mathtt{cost}}$ in Section 4.4. Paths that grow from keyword vertices are iteratively enumerated. Initially, each keyword vertex is treated as a path of length zero and inserted into PQ to be processed (lines 3–4). For each keyword $k_i \in Q$, we maintain the set P_i of processed paths that contain a keyword vertex in K_i (lines 5–6).

In each iteration (line 7), p_{top} which minimizes $\overline{\mathtt{cost}}$ in PQ is pulled to process (line 8). If $\overline{\mathtt{cost}}(p_{\text{top}}) \geq \mathtt{cost}(T_{\text{opt}})$, the algorithm will be terminated (lines 9–10). Otherwise, p_{top} is added to each proper P_i (lines 11–12). Specifically, $V_{p_{\text{top}}}$ is the set of vertices in p_{top}, and we define

$$\mathtt{QK}(V_{p_{\text{top}}}) = \{k_i \in Q : \ V_{p_{\text{top}}} \cap K_i \neq \emptyset\} . \tag{3}$$

New feasible answers are constructed with p_{top}. We consider each combination of paths in $P_1 \times \cdots \times P_g$ such that p_{top} is in this combination where all the paths have a common end vertex (line 13). They are merged into a subgraph T which clearly is connected and covers all the keywords in Q (line 14). If T is also structurally minimal (line 15), i.e., it is a feasible answer, and its cost is smaller than $\mathtt{cost}(T_{\text{opt}})$ (line 16), it will replace T_{opt} (line 17).

Longer paths are enumerated by growing p_{top} with one edge (lines 19–20), and are inserted into PQ to be processed (line 21). Growing is conditioned on the length of p_{top} (line 18), subject to the largest allowed depth of search.

Finally, T_{opt} is an optimum answer and is returned (line 22).

4.3 Algorithm Analysis

Correctness. Consider Lemma 2 which decomposes a feasible answer into a set of paths.

Lemma 2. *Every feasible answer T s.t. $\mathtt{diam}(T) \leq 2d$ can be constructed by merging g paths $p_1, \dots, p_g$ such that for $1 \leq i \leq g$: (1) $\mathtt{len}(p_i) \leq d$, (2) the start vertex of p_i is in K_i, and (3) all p_i have a common end vertex.*

Proof. Since $T = \langle V_T, E_T \rangle$ is a tree, it is known in graph theory that: $\mathtt{rad}(T) = \lfloor \frac{\mathtt{diam}(T)+1}{2} \rfloor \leq \lfloor \frac{2d+1}{2} \rfloor = d$. Therefore, T has a central vertex c that is at most d hops away from every vertex in T. For $1 \leq i \leq g$, since $V_T \cap K_i \neq \emptyset$, we choose $v_i \in V_T \cap K_i$ and let p_i be the unique path between v_i and c in T. All such p_i satisfy (1)–(3) of the lemma, and by merging them we can construct T. □

The correctness theorem follows directly from Lemma 2.

Theorem 3. *B^{3}F returns an optimum answer s.t. $\mathtt{diam} \leq 2d$.*

Proof. Consider a variant of B^{3}F where we fix $\overline{\mathtt{cost}} = -\infty$; thus branches are never discarded. When this variant is finished, $P_1, \dots, P_k$ contain all possible paths that start from keyword vertices and are at most d long. All their combinations have been used to construct feasible answers, including the combination that produces an optimum answer according to Lemma 2. Therefore, T_{opt} is an optimum answer.

Now consider the standard version of B^{3}F where $\overline{\mathtt{cost}}$ is properly estimated. When it is terminated early, according to the definition of $\overline{\mathtt{cost}}$, it is impossible to discard a feasible answer better than T_{opt}, so T_{opt} is an optimum answer. □

Time complexity. For $G = \langle V, E \rangle$, let $|V| = n$ and $|E| = m$. There are $O(n)$ keyword vertices, starting from which $O(n^{d+1})$ paths are enumerated and processed. Fibonacci heap allows $O(\log n^{d+1})$ time for pull and $O(1)$ time for insert. All the queue operations take $O(n^{d+1}(d+1)\log n)$ time. To construct feasible answers, $O(n^{g(d+1)})$ combinations of paths are merged. Each combination uses $O(gd+1)$ time for merging and $O((gd+1)^2)$ time for calculating $\mathtt{cost}$, assuming $O(1)$ for computing $\mathtt{sd}$. Therefore, finding an optimum answer takes $O((gd+1)^2 n^{g(d+1)})$ time. The total time complexity of B^{3}F is $O(n^{d+1}(d+1)\log n + (gd+1)^2 n^{g(d+1)})$. It increases exponentially with g and d, which are very small in practice. By early termination based on our lower bound estimation of $\overline{\mathtt{cost}}$ described in Section 4.4, B^{3}F can achieve promising performance on medium-sized KGs.

4.4 Lower Bound Estimation

For path $p = \langle V_p, E_p \rangle$, properly estimating $\overline{\mathtt{cost}}(p)$ is the key to the performance of B^{3}F and is one of our main technical contributions. To ensure Eq. (2), we analyze the cost of an arbitrary feasible answer $T = \langle V_T, E_T \rangle$ that satisfies $p \preceq T$. For convenience, we decompose the cost function in Eq. (1):

$$\mathtt{cost}(T) = \alpha \cdot \mathtt{cost}_{\mathtt{wt}}(T) + (1-\alpha) \cdot \mathtt{cost}_{\mathtt{sd}}(T) \text{ , where}$$
$$\mathtt{cost}_{\mathtt{wt}}(T) = \sum_{v \in V_T} \mathtt{wt}(v) \, , \quad \mathtt{cost}_{\mathtt{sd}}(T) = \sum_{\substack{v_i, v_j \in V_T \\ \text{s.t. } i<j}} \mathtt{sd}(v_i, v_j) \, . \tag{4}$$

We will separately estimate a lower bound for each part. Note that the estimated bounds must be independent from T. Moreover, the estimation per se should be computationally inexpensive to improve the overall performance of B^{3}F.

Lower Bound Estimation for $\mathtt{cost}_{\mathtt{wt}}(T)$

We expand $\mathtt{cost}_{\mathtt{wt}}(T)$ in Eq. (4):

$$\mathtt{cost}_{\mathtt{wt}}(T) = \sum_{v \in V_p} \mathtt{wt}(v) + \sum_{v \in (V_T \setminus V_p)} \mathtt{wt}(v) \, . \tag{5}$$

The first sum is independent from T and is directly calculated. For the second sum, we estimate its lower bound using the smallest total weight of other vertices besides V_p that are needed for T to cover all the keywords in Q. The computation of this total weight is formulated as the following instance of the weighted set cover problem (WSCP):

$$\begin{aligned} \text{universe of elements: } & Q \setminus \mathtt{QK}(V_p) \, , \\ \text{sets of elements: } & \text{each } \emptyset \subset Q' \subseteq (Q \setminus \mathtt{QK}(V_p)) \\ & \text{s.t. } \exists v \in V, \ \mathtt{QK}(\{v\}) \setminus \mathtt{QK}(V_p) = Q' \, , \\ \text{weights of sets: } & \min_{v \in V: \ \mathtt{QK}(\{v\}) \setminus \mathtt{QK}(V_p) = Q'} \mathtt{wt}(v) \text{ for } Q' \, . \end{aligned} \tag{6}$$

Let $\mathbf{Q}'_{\text{opt}}$ be an optimum solution to Eq. (6) computed by dynamic programming. The total weight of the sets in $\mathbf{Q}'_{\text{opt}}$ forms a lower bound for the second sum in Eq. (5). To conclude, we estimate the following lower bound for $\texttt{cost}_{\texttt{wt}}(T)$:

$$\texttt{cost}_{\texttt{wt}}(T) \geq \sum_{v \in V_p} \texttt{wt}(v) + \sum_{Q' \in \mathbf{Q}'_{\text{opt}}} \min_{v \in V:\, \texttt{QK}(\{v\}) \setminus \texttt{QK}(V_p) = Q'} \texttt{wt}(v)\,, \tag{7}$$

which is independent from T.

Although WSCP is NP-hard, there are at most $(2^g - 1)$ unique instances of WSCP to solve, where g is the number of keywords in Q; each instance takes a unique proper subset of Q as the universe. Since g is very small in practice, both the number of unique instances of WSCP and the size of each instance are small. Therefore, in practice the time for the above estimation is relatively neglectable in B^{3}F.

Lower Bound Estimation for $\texttt{cost}_{\texttt{sd}}(T)$

We assume sd satisfies symmetry and triangle inequality. We expand $\texttt{cost}_{\texttt{sd}}(T)$ in Eq. (4):

$$\begin{aligned}\texttt{cost}_{\texttt{sd}}(T) &\geq \sum_{\substack{v_i, v_j \in V_p \\ \text{s.t. } i<j}} \texttt{sd}(v_i, v_j) + \sum_{v_i \in (V_T \setminus V_p),\, v_j \in V_p} \texttt{sd}(v_i, v_j) \\ &\geq \sum_{\substack{v_i, v_j \in V_p \\ \text{s.t. } i<j}} \texttt{sd}(v_i, v_j) + r \cdot \min_{v_i \in V} \sum_{v_j \in V_p} \texttt{sd}(v_i, v_j)\,.\end{aligned} \tag{8}$$

The first sum is independent from T and is directly calculated. In the second sum, r represents the smallest number of other vertices besides V_p that are needed for T to cover all the keywords in Q. The computation of r is formulated as the following instance of the set cover problem (SCP):

$$\begin{aligned}&\text{universe of elements: } Q \setminus \texttt{QK}(V_p)\,, \\ &\text{sets of elements: each } \emptyset \subset Q'' \subseteq (Q \setminus \texttt{QK}(V_p)) \\ &\qquad \text{s.t. } \exists v \in V,\ \texttt{QK}(\{v\}) \setminus \texttt{QK}(V_p) = Q''\,.\end{aligned} \tag{9}$$

Let $\mathbf{Q}''_{\text{opt}}$ be an optimum solution to Eq. (9) computed by dynamic programming. The number of sets in $\mathbf{Q}''_{\text{opt}}$ forms r in Eq. (8), i.e., $r = |\mathbf{Q}''_{\text{opt}}|$.

To estimate a lower bound for the minimum in Eq. (8), our idea is: for $v_i \in V$, its semantic distance from *some* vertices in V_p may be very small, but its total semantic distance from *all* the vertices in V_p cannot be arbitrarily small due to triangle inequality. Therefore, we construct a complete undirected graph H where vertices are V_p and the edge between $v_i, v_j \in V_p$ has weight $\texttt{sd}(v_i, v_j)$. Let M be a maximum weighted matching in H, which puts $2\lfloor \frac{|V_p|}{2} \rfloor$ vertices in pairs. Let $\widetilde{M}$ be the sum of edge weights in M. Since sd satisfies triangle inequality, we have

$$\widetilde{M} \leq \min_{v_i \in V} \sum_{v_j \in V_p} \texttt{sd}(v_i, v_j)\,. \tag{10}$$

Now we can turn to estimate a lower bound for $\widetilde{M}$. When $|V_p|$ is even, the edges in H can be partitioned into $|V_p| - 1$ perfect matchings according to graph theory, so

$$\widetilde{M} \geq \frac{\sum_{\substack{v_i, v_j \in V_p \\ \text{s.t. } i<j}} \texttt{sd}(v_i, v_j)}{|V_p| - 1}\,. \tag{11}$$

When $|V_p|$ is odd, let H_k be the subgraph of H from which $v_k \in V_p$ and its incident edges are removed. Let M_k be a maximum weighted matching in H_k, and let $\widetilde{M_k}$ be the sum of edge weights in M_k. For each $v_k \in V_p$, we have

$$\begin{aligned}\sum_{\substack{v_i, v_j \in V_p \\ \text{s.t. } i<j}} \texttt{sd}(v_i, v_j) = &\sum_{\substack{v_i, v_j \in (V_p \setminus \{v_k\}) \\ \text{s.t. } i<j}} \texttt{sd}(v_i, v_j) \\ &+ \sum_{v \in (V_p \setminus \{v_k\})} \texttt{sd}(v, v_k)\,.\end{aligned} \tag{12}$$

Summing this equation over all $v_k \in V_p$, we have

$$\begin{aligned}&|V_p| \sum_{\substack{v_i, v_j \in V_p \\ \text{s.t. } i<j}} \texttt{sd}(v_i, v_j) \\ &= \sum_{v_k \in V_p} \Big(\sum_{\substack{v_i, v_j \in (V_p \setminus \{v_k\}) \\ \text{s.t. } i<j}} \texttt{sd}(v_i, v_j) + \sum_{v \in (V_p \setminus \{v_k\})} \texttt{sd}(v, v_k) \Big) \\ &= \sum_{v_k \in V_p} \sum_{\substack{v_i, v_j \in (V_p \setminus \{v_k\}) \\ \text{s.t. } i<j}} \texttt{sd}(v_i, v_j) + 2 \sum_{\substack{v_i, v_j \in V_p \\ \text{s.t. } i<j}} \texttt{sd}(v_i, v_j)\,,\end{aligned} \tag{13}$$

from which we obtain

$$\begin{aligned}\sum_{\substack{v_i, v_j \in V_p \\ \text{s.t. } i<j}} \texttt{sd}(v_i, v_j) &= \sum_{v_k \in V_p} \frac{\sum_{\substack{v_i, v_j \in (V_p \setminus \{v_k\}) \\ \text{s.t. } i<j}} \texttt{sd}(v_i, v_j)}{|V_p| - 2} \\ &\leq \sum_{v_k \in V_p} \widetilde{M_k} \leq |V_p| \cdot \widetilde{M}\,,\end{aligned} \tag{14}$$

that is,

$$\widetilde{M} \geq \frac{\sum_{\substack{v_i, v_j \in V_p \\ \text{s.t. } i<j}} \texttt{sd}(v_i, v_j)}{|V_p|}\,. \tag{15}$$

Combining Eqs. (8)(10)(11)(15), we estimate the following lower bound for $\texttt{cost}_{\texttt{sd}}(T)$:

$$\texttt{cost}_{\texttt{sd}}(T) \geq \Big(1 + \frac{|\mathbf{Q}''_{\text{opt}}|}{1 + 2\lfloor \frac{|V_p| - 1}{2} \rfloor}\Big) \sum_{\substack{v_i, v_j \in V_p \\ \text{s.t. } i<j}} \texttt{sd}(v_i, v_j)\,, \tag{16}$$

which is independent from T.

In practice, the time for the above estimation is relatively neglectable in B^{3}F because: first, following our analysis about WSCP, the time for computing $|\mathbf{Q}''_{\text{opt}}|$ is relatively neglectable; second, the sum in Eq. (16) can be incrementally calculated when p is enumerated by growing its sub-path.

5 Experiments

We experimented with a 3.5GHz CPU and 24GB memory.
Code: https://github.com/nju-websoft/B3F .

5.1 Datasets

We used five versions of three public KGs. They represent small to medium-sized KGs as shown in Table 1.

	KG			Query	
	Source	$\lvert V\rvert$	$\lvert E\rvert$	Qty	g
MND	MONDIAL	40,890	120,690	34	2–4
L1	LUBM	20,797	82,590	200	2–6
L4	LUBM	91,045	370,654	200	2–6
D20K	DBpedia	20,000	37,017	128	2–6
D100K	DBpedia	100,000	205,743	241	2–6

Table 1: KGs and Queries

MONDIAL[1] (**MND**) is a geographical KG that has been popularly used for evaluating keyword querying. We reused 50 queries provided by [Coffman and Weaver, 2014] but removed those containing unmatchable keywords. Our matching function returns vertices whose textual annotations (i.e., literals) contain the given keyword.

LUBM[2] is a benchmark for generating synthetic KGs in the university domain. We generated two KGs describing one university (**L1**) and four universities (**L4**). We generated 200 synthetic queries following the procedure in [Li *et al.*, 2016]. Specifically, we varied two parameters: the number of keywords in a query (i.e., g) and the average number of vertices matched with each keyword in a query (denoted by f). For each $g \in \{2, 3, 4, 5, 6\}$ and each $f \in \{5, 10, 50, 100\}$ we generated 10 queries. Given g and f, to generate a query we created g pseudo keywords and we let each pseudo keyword match with an average of f random vertices.

From the well-known encyclopedic KG of DBpedia[3] we extracted two subgraphs **D20K** and **D100K** consisting of different numbers of vertices with the largest degrees and all the edges between them. We reused 429 queries provided by [Hasibi *et al.*, 2017] but removed those containing unmatchable keywords, i.e., the given keyword is not contained in any label annotations (`rdfs:label`) of vertices.

5.2 Cost Function and Parameters

Concrete implementations of `wt` and `sd` are not our research focus. We reused existing normalized PageRank for `wt`:

$$\mathtt{wt}(v) = 1 - \mathtt{sigmoid}\left(\log \frac{\text{PageRank}(v)}{\min\limits_{v' \in V} \text{PageRank}(v')}\right). \tag{17}$$

For semantic distance `sd`, we followed [Cheng *et al.*, 2017b] to compute the Jaccard distance between sets of types of entities in DBpedia to capture ontological semantics. This is not suitable for MONDIAL and LUBM where each entity has a single type. On these datasets, we computed the angular distance between 10-dimensional embedding vectors of entities generated by pyRDF2Vec[4] to capture structural semantics.

For the parameter α in the cost function we compared two values: $\alpha = 0.3$ and $\alpha = 0.7$. For the largest allowed depth of search in B^{3}F we set $d = 3$ since subgraphs of $\mathtt{diam} > 6$ would be too large to present in real applications.

[1]https://www.dbis.informatik.uni-goettingen.de/Mondial

[2]http://swat.cse.lehigh.edu/projects/lubm/

[3]https://wiki.dbpedia.org/ (version 2016-10)

[4]https://github.com/IBCNServices/pyRDF2Vec

		MND	L1	L4	D20K	D100K
Time	B^{3}F ($\alpha = 0.3$)	1.68	1.84	30.96	4.09	261.14
	B^{3}F ($\alpha = 0.7$)	1.31	2.42	32.50	6.52	101.12
	DPBF	0.19	16.15	113.00	2.29	25.41
Ratio	B^{3}F ($\alpha = 0.3$)	0.89	0.47	0.53	0.79	0.64
	B^{3}F ($\alpha = 0.7$)	0.92	0.56	0.63	0.80	0.65

Table 2: Mean Run Time (seconds) and Mean Cohesiveness Ratio

5.3 Baseline

We are the first to propose an exact algorithm for QGSTP. We compared our algorithm with **DPBF** [Ding *et al.*, 2007], a state-of-the-art exact algorithm for the standard vertex-weighted GST problem which is a simple special case of QGSTP. DPBF minimizes the total weight of the vertices in an answer but is unaware of semantic distance. Recent faster algorithms for the GST problem [Li *et al.*, 2016; Shi *et al.*, 2020] were not compared because they were designed for edge weights but could not handle vertex weights.

5.4 Efficiency Results

Recall that our research target in this paper is to efficiently solve QGSTP. Thus, run time is our main evaluation metric.

Metric. We measured the mean **run time** of each algorithm for answering a query. The results are shown in Table 2.

Comparison with baseline. The run time of B^{3}F and DPBF was generally at the same level of magnitude. B^{3}F was 3.48–8.78 times as fast as DPBF on L1 and L4, but was 1.79–10.28 times as slow as DBPF on the other KGs.

Scalability. B^{3}F was fast on small KGs of 20k–40k vertices. It only used 1.31–6.52 seconds to answer a query on MND, L1, and D20K. However, it took 30.96–261.14 seconds on medium-sized KGs of about 100k vertices like L4 and D100K, i.e., the run time grew superlinearly with the number of vertices. In Fig. 2 we break down the results under $\alpha = 0.3$ by g, the number of keywords in a query. The run time grew noticeably when g was increased. Similar results under $\alpha = 0.7$ are shown in Fig. 3.

Discussion. On small and medium-sized KGs, B^{3}F showed promising performance comparable to DPBF but note that B^{3}F solved a harder problem where cohesiveness was modeled by quadratic terms to produce better answers. The performance is acceptable for many enterprise-level applications such as analytics-oriented exploration which is more focused on answer quality rather than run time. Moreover, one could easily improve the performance by parallelizing the execution of B^{3}F, e.g., by processing multiple combinations of paths in parallel. However, B^{3}F and DPBF are exact algorithms for NP-hard problems and they have exponential run time in the worst case. They may not be the best choice for applications where KGs are extremely large and run time rather than answer quality is the main concern. For such applications, one may consider more scalable approximation algorithms [Shi *et al.*, 2021; Shi *et al.*, 2020].

5.5 Effectiveness Results

Our research is focused on efficient algorithms for the generalized QGSTP, rather than the effectiveness of its concrete im-

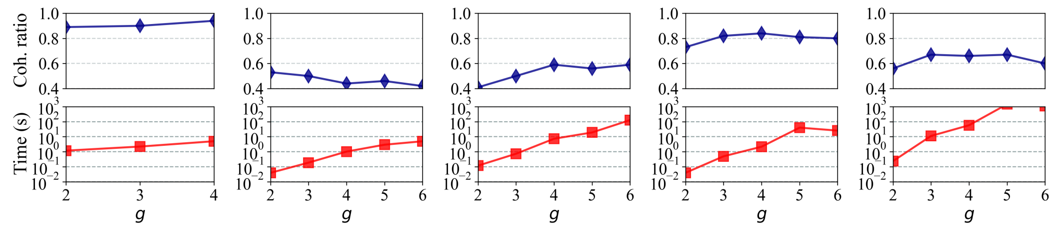

(a) MND (b) L1 (c) L4 (d) D20K (e) D100K

Figure 2: Mean cohesiveness ratio and run time of B^{3}F ($\alpha = 0.3$) under different values of g.

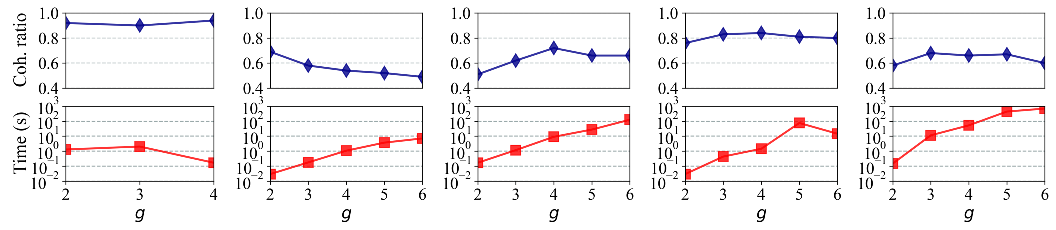

(a) MND (b) L1 (c) L4 (d) D20K (e) D100K

Figure 3: Mean cohesiveness ratio and run time of B^{3}F ($\alpha = 0.7$) under different values of g.

plementation in KG exploration which has been empirically demonstrated by case or user studies [Cheng *et al.*, 2017b; Bryson *et al.*, 2020]. Here we mainly aimed to extend such qualitative studies by quantitatively comparing the cohesiveness of our quadratic GSTs with standard GSTs.

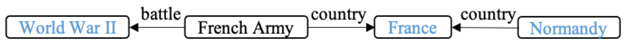
Metric. For each query we measured **cohesiveness ratio**:

$$\text{cohesiveness ratio} = \frac{\texttt{cost}_{\texttt{sd}}(T_{\mathrm{B^3F}})}{\texttt{cost}_{\texttt{sd}}(T_{\mathrm{DPBF}})}, \quad (18)$$

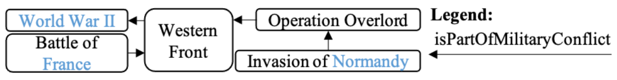
where $\texttt{cost}_{\texttt{sd}}$ is defined in Eq. (4); $T_{\mathrm{B^3F}}$ and T_{DPBF} represent the answers computed by B^{3}F and DPBF, respectively. The mean cohesiveness ratios for a query are shown in Table 2.

Comparison with baseline. The ratios were in the range of 0.47–0.92, i.e., quadratic GSTs were 8%–53% more cohesive than standard GSTs. It partially justified the advantage of cohesive answers reported in [Cheng *et al.*, 2017b]. Similar to our efficiency experiment, in Fig. 2 and Fig. 3 we break down the results by g, the number of keywords in a query. We did not observe explicit correlation between them.

Case study. In Fig. 4 we show two answers computed by different methods for a query on DBpedia in our experiments. The GST computed by DPBF is more compact and contains more salient entities such as `France`, but it is not a cohesive subgraph. The four entities have different types and are semantically distant from each other. They do not constitute a meaningful answer. By comparison, the quadratic GST computed by B^{3}F is a cohesive subgraph where the five entities have the same type and are semantically close to each other. It meaningfully describes relations between military conflicts.



(a) A GST computed by DPBF.



(b) A quadratic GST computed by B^{3}F.

Figure 4: Two answers computed by different methods for the query "France, World War II, Normandy" on DBpedia.

6 Conclusion

QGSTP has been formulated for improving the cohesiveness of answers for KG exploration. This generalized model offers the flexibility to measure weights, semantic distances, and tune their combination. To solve QGSTP we proposed B^{3}F. Compared with exact algorithms for computing standard GSTs, our exact algorithm for QGSTP computed more cohesive answers in comparable run time. It showed promising performance on medium-sized KGs which have supported many real-life applications. It can also be used as a baseline for evaluating approximation algorithms for QGSTP.

Acknowledgments

This work was supported in part by the National Key R&D Program of China (2018YFB1005100), in part by the NSFC (62072224), and in part by the Fundamental Research Funds for the Central Universities (0202-14380084).

References

[Bryson *et al.*, 2020] Spencer Bryson, Heidar Davoudi, Lukasz Golab, Mehdi Kargar, Yuliya Lytvyn, Piotr Mierzejewski, Jaroslaw Szlichta, and Morteza Zihayat. Robust keyword search in large attributed graphs. *Inf. Retr. J.*, 23(5):502–524, 2020.

[Cheng and Kharlamov, 2017] Gong Cheng and Evgeny Kharlamov. Towards a semantic keyword search over industrial knowledge graphs (extended abstract). In *IEEE BigData 2017*, pages 1698–1700, 2017.

[Cheng *et al.*, 2016] Gong Cheng, Cheng Jin, and Yuzhong Qu. HIEDS: A generic and efficient approach to hierarchical dataset summarization. In *IJCAI 2016*, pages 3705–3711, 2016.

[Cheng *et al.*, 2017a] Gong Cheng, Cheng Jin, Wentao Ding, Danyun Xu, and Yuzhong Qu. Generating illustrative snippets for open data on the web. In *WSDM 2017*, pages 151–159, 2017.

[Cheng *et al.*, 2017b] Gong Cheng, Fei Shao, and Yuzhong Qu. An empirical evaluation of techniques for ranking semantic associations. *IEEE Trans. Knowl. Data Eng.*, 29(11):2388–2401, 2017.

[Cheng *et al.*, 2020] Gong Cheng, Shuxin Li, Ke Zhang, and Chengkai Li. Generating compact and relaxable answers to keyword queries over knowledge graphs. In *ISWC 2020, Part I*, pages 110–127, 2020.

[Cheng, 2020] Gong Cheng. Relationship search over knowledge graphs. *ACM SIGWEB Newsl.*, 2020(Summer):3, 2020.

[Coffman and Weaver, 2014] Joel Coffman and Alfred C. Weaver. An empirical performance evaluation of relational keyword search techniques. *IEEE Trans. Knowl. Data Eng.*, 26(1):30–42, 2014.

[Dass *et al.*, 2015] Ananya Dass, Aggeliki Dimitriou, Cem Aksoy, and Dimitri Theodoratos. Incorporating cohesiveness into keyword search on linked data. In *WISE 2015, Part II*, pages 47–62, 2015.

[Ding *et al.*, 2007] Bolin Ding, Jeffrey Xu Yu, Shan Wang, Lu Qin, Xiao Zhang, and Xuemin Lin. Finding top-k min-cost connected trees in databases. In *ICDE 2007*, pages 836–845, 2007.

[Gu *et al.*, 2019] Yu Gu, Tianshuo Zhou, Gong Cheng, Ziyang Li, Jeff Z. Pan, and Yuzhong Qu. Relevance search over schema-rich knowledge graphs. In *WSDM 2019*, pages 114–122, 2019.

[Hasibi *et al.*, 2017] Faegheh Hasibi, Fedor Nikolaev, Chenyan Xiong, Krisztian Balog, Svein Erik Bratsberg, Alexander Kotov, and Jamie Callan. DBpedia-Entity v2: A test collection for entity search. In *SIGIR 2017*, pages 1265–1268, 2017.

[Ihler, 1991] Edmund Ihler. The complexity of approximating the class Steiner tree problem. In *WG 1991*, pages 85–96, 1991.

[Kacholia *et al.*, 2005] Varun Kacholia, Shashank Pandit, Soumen Chakrabarti, S. Sudarshan, Rushi Desai, and Hrishikesh Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB 2005*, pages 505–516, 2005.

[Kargar and An, 2011] Mehdi Kargar and Aijun An. Keyword search in graphs: Finding r-cliques. *PVLDB*, 4(10):681–692, 2011.

[Klein and Ravi, 1995] Philip N. Klein and R. Ravi. A nearly best-possible approximation algorithm for node-weighted Steiner trees. *J. Algorithms*, 19(1):104–115, 1995.

[Le *et al.*, 2014] Wangchao Le, Feifei Li, Anastasios Kementsietsidis, and Songyun Duan. Scalable keyword search on large RDF data. *IEEE Trans. Knowl. Data Eng.*, 26(11):2774–2788, 2014.

[Li *et al.*, 2016] Rong-Hua Li, Lu Qin, Jeffrey Xu Yu, and Rui Mao. Efficient and progressive group Steiner tree search. In *SIGMOD 2016*, pages 91–106, 2016.

[Li *et al.*, 2020] Shuxin Li, Gong Cheng, and Chengkai Li. Relaxing relationship queries on graph data. *J. Web Semant.*, 61-62:100557, 2020.

[Lissandrini *et al.*, 2020] Matteo Lissandrini, Torben Bach Pedersen, Katja Hose, and Davide Mottin. Knowledge graph exploration: where are we and where are we going? *ACM SIGWEB Newsl.*, 2020(Summer):4, 2020.

[Pound *et al.*, 2012] Jeffrey Pound, Alexander K. Hudek, Ihab F. Ilyas, and Grant E. Weddell. Interpreting keyword queries over web knowledge bases. In *CIKM 2012*, pages 305–314, 2012.

[Shekarpour *et al.*, 2013] Saeedeh Shekarpour, Axel-Cyrille Ngonga Ngomo, and Sören Auer. Question answering on interlinked data. In *WWW 2013*, pages 1145–1156, 2013.

[Shi *et al.*, 2020] Yuxuan Shi, Gong Cheng, and Evgeny Kharlamov. Keyword search over knowledge graphs via static and dynamic hub labelings. In *WWW 2020*, pages 235–245, 2020.

[Shi *et al.*, 2021] Yuxuan Shi, Gong Cheng, Trung-Kien Tran, Evgeny Kharlamov, and Yulin Shen. Efficient computation of semantically cohesive subgraphs for keyword-based knowledge graph exploration. In *WWW 2021*, 2021.

[Sun *et al.*, 2020] Yawei Sun, Lingling Zhang, Gong Cheng, and Yuzhong Qu. SPARQA: skeleton-based semantic parsing for complex questions over knowledge bases. In *AAAI-IAAI-EAAI 2020*, pages 8952–8959, 2020.

[Tran *et al.*, 2009] Thanh Tran, Haofen Wang, Sebastian Rudolph, and Philipp Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (RDF) data. In *ICDE 2009*, pages 405–416, 2009.

[Yang *et al.*, 2019] Yueji Yang, Divyakant Agrawal, H. V. Jagadish, Anthony K. H. Tung, and Shuang Wu. An efficient parallel keyword search engine on knowledge graphs. In *ICDE 2019*, pages 338–349, 2019.

[Zhou *et al.*, 2020] Tianshuo Zhou, Ziyang Li, Gong Cheng, Jun Wang, and Yuang Wei. GREASE: A generative model for relevance search over knowledge graphs. In *WSDM 2020*, pages 780–788, 2020.

[Zhu *et al.*, 2018] Yuanyuan Zhu, Qian Zhang, Lu Qin, Lijun Chang, and Jeffrey Xu Yu. Querying cohesive subgraphs by keywords. In *ICDE 2018*, pages 1324–1327, 2018.