GCTKG: Group Center for Keyword Search over Knowledge Graphs

Xiaoxiao Xie[‡], Shengfei Shi[‡*], Kaiqi Zhang[‡], Chao Yi[§]
[‡] Faculty of Computing, Harbin Institute of Technology, 150001 Harbin, Heilongjiang Province, China
shengfei@hit.edu.cn, xxxie@stu.hit.edu.cn, zhangkaiqi@hit.edu.cn
[§] Chinese Scholartree Ridge State Key Laboratory, China North Vehicle Reserch Institute, 100072 Beijing, China
yichao1667@163.com

***Abstract*—With the development of GraphRAG, leveraging knowledge graphs for information retrieval and question answering has re-emerged as a prominent research direction. For graph-structured data, the widely adopted keyword search approach is based on the Group Steiner Tree problem. This problem involves finding a tree in an undirected graph that satisfies two conditions: (i) the tree contains at least one vertex from each predefined group of vertices, and (ii) the total weight of the edges in the tree is minimized.**

As an NP-hard problem, the best existing polynomial-time algorithm achieves an approximation ratio of $g-1$, leaving a notable gap between the approximate and optimal solutions. Furthermore, algorithms with sublinear approximation ratios exhibit practical runtimes comparable to those of exact solution methods, making them prohibitively expensive in practice.

To address these challenges, this paper introduces an approximation algorithm named GCTKG, which maintains the time complexity of the current best polynomial-time approximation algorithm while improving the approximation ratio to $\frac{g}{2}$. GCTKG leverages group center as root and demonstrates superior performance on high-memory servers, producing more accurate approximate solutions with a runtime comparable to the best existing algorithms.

Additionally, we propose an enhanced version, GCTKG+, which further improves the empirical approximation ratio without increasing the computational complexity. Extensive experiment on three public datasets showcases the exceptional performance of both GCTKG and GCTKG+.

***Index Terms*—Graph Algorithms, Keyword Search, Group Steiner Tree.**

I. Introduction

The role of knowledge graph keyword search techniques in knowledge graph information retrieval lies primarily in enhancing semantic understanding, improving query accuracy and efficiency, and broadening the depth and scope of search results. By matching and associating users' query keywords with entities and relationships in the knowledge graph, this technique enables retrieval systems to better interpret user intent and deliver more precise and relevant information.

In the context of retrieval-enhanced generation technique [1], the keywords needed for knowledge graph retrieval, serving as anchors, must be extracted from natural language. This process is typically carried out by large language models [2]. Organizing these extracted keywords within the knowledge graph and extracting the corresponding knowledge is critical [3]–[6]. Generally, knowledge in a knowledge graph is most commonly represented as paths consisting of entity vertices and relationship edges, with the extracted keywords associated through these paths.

Since the paths between two keywords in a knowledge graph are not unique, it is important to ensure the knowledge obtained is concise, compact, and highly relevant. This implies that the number of hops between keywords should be minimized, meaning the paths connecting them should be as short as possible [23]–[26]. This requirement aligns closely with the definition of the Minimum Group Steiner Tree problem [7]. Moreover, the minimum Steiner tree problem is a classic challenge in relational databases and social networks [28]–[32].

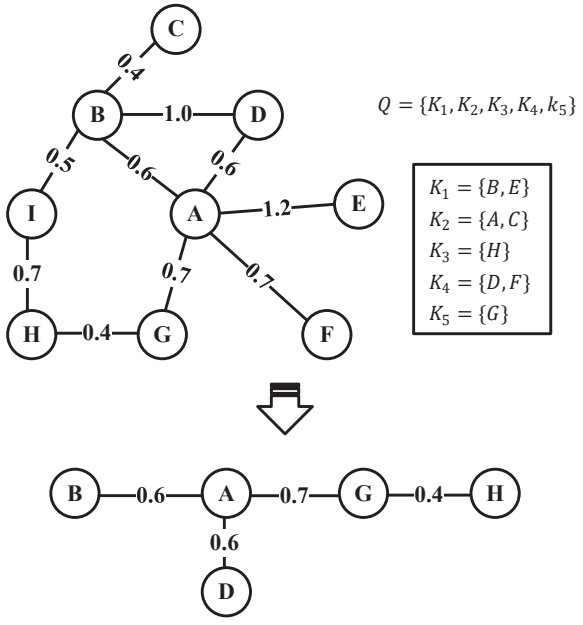


Fig. 1. An example of keyword search on konwledge graph.

The Minimum Group Steiner Tree problem is a NP-hard problem [19], [20], which is defined as follows: Given a set of vertices on a graph as a group, and finite groups constituting a collection, provide a tree on the graph such

The author with * is the corresponding author

that at least one vertex from each group of a collection's subset is included in the tree [8], [18], and the tree is the one with the minimum weight among all trees that satisfy the above condition. The mathematical definition is given below: Let there be a graph $G = \langle V, E \rangle$, and a set of groups $\mathbb{K} = \{K_1, K_2, ..., K_{|\mathbb{K}|}\}$ where each $K_i \subseteq V$. For a query set $Q = \{K_{Q_1}, K_{Q_2}, ..., K_{Q_g}\}$ with each $K_{Q_i} \in \mathbb{K}$, the Minimum Group Steiner Tree $T^* = \langle V_{T^*}, E_{T^*} \rangle$ is defined as:

$$T^* = \arg\min_{|T|}\{T = \langle V_T, E_T \rangle | V_T \subseteq V, E_T \subseteq E, \\ \forall K_i \in Q, \exists k_j \in K_i \ s.t. \ k_j \in V_T\} \tag{1}$$

where $|T|$ denotes the weight of the tree T. In this paper, we only consider the weight of the edges [9].

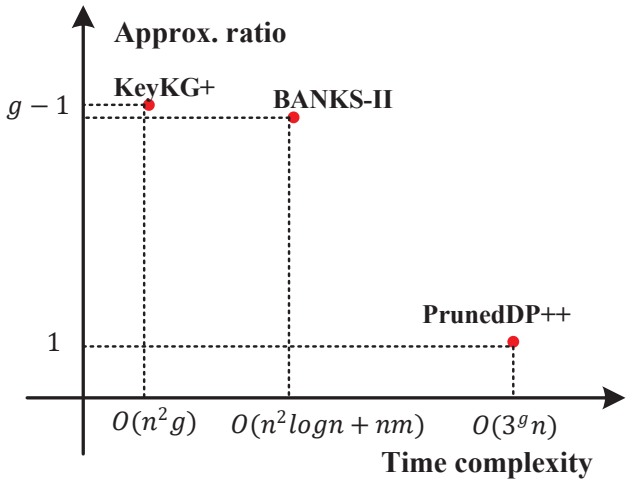


Fig. 2. Pareto optimal curve of existing algorithms.

As illustrated in Fig. 1, when a query Q is issued, keyword vertices in the graph are selected based on the keyword groups within Q, resulting in a minimum Group Steiner Tree. In the example provided, Q consists of five groups, and the optimal solution is the minimum Steiner tree containing the vertices $\{A, B, D, G, H\}$ with a total cost of 2.3.

In existing approximation algorithms [10]–[13], the time complexity is largely dictated by the computation of the shortest paths between vertices in the graph. Some approaches employ bidirectional search strategies, where methods such as bidirectional A* [27] are used to compute the shortest path from both vertices towards a meeting vertex after identifying two vertices in the graph. The computation of these shortest paths is critical in maintaining the current approximation ratio of $g - 1$. Additionally, some methods include preprocessing steps, such as building graph indexes [21], [22], [41], to facilitate subsequent shortest-distance calculations. While these preprocessing steps can incur substantial costs, they are often excluded from the final calculation of the algorithm's time complexity.

To address the time constraints on solution precision imposed by shortest path computations, the method proposed in this paper adheres to the preprocessing cost of the best existing approximation algorithms, which involve performing $|V|$ Dijkstra computations. This preprocessing step generates a shortest-path table for all pairs of vertices in the graph, streamlining the subsequent calculations and processing of group center.

One approach to reducing the approximation ratio of the solution obtained by an algorithm is to minimize the repeated computation of paths. The **2-star** algorithm offers a way to prove the approximation ratio by calculating the maximum number of times a path is computed repeatedly. The **PrunedDP++** algorithm introduces the following formula:

$$f_T^*(v, X) = \min\{\min_{(v,u)\in E}\{f_T^*(u, X) + w(v, u)\}, \\ \min_{\substack{(X=X_1\cup X_2)\wedge \\ (X_1\cap X_2=\emptyset)}}\{f_T^*(v, X_1) + f_T^*(v, X_2)\}\} \tag{2}$$

where $f_T^*(v, X)$ represents the weight of the minimum tree with root v covering the set of groups X. This formula illustrates that one source of path repetition arises from the computation of $w(v, u)$, which indicates that there are common ancestor vertices between the root v and the tree covering the set of groups X. As a result, the approximation algorithm repeatedly computes the cost of $w(v, u)$ $|X|$ times. To minimize path repetition, the method proposed in this paper, **GCTKG**, selects the group center as the root of the final Steiner tree. It is shown that this choice of root leads to an approximation ratio of $\frac{g}{2}$. Given the inapproximability of the minimum Steiner tree problem with an $O(\log g)$ factor [15], reducing the approximation ratio to a fraction of its original value within polynomial time complexity is a significant achievement. By analyzing the characteristics of the exact solution for the Minimum Group Steiner Tree, we propose an improved version of the **GCTKG** algorithm, named **GCTKG+**. This enhanced algorithm improves the solution's precision without exceeding the time complexity and approximation ratio of the original **GCTKG**.

To summarize, our key contributions are as follows:

(i) We propose a novel method for selecting the root that deviates from traditional keyword-based approaches. By utilizing comprehensive preprocessed data, we strategically position the root at the center of the groups. This not only minimizes the cost associated with each keyword but also rigorously establishes the algorithm's $\frac{g}{2}$ approximation ratio.

(ii) Building on the foundation above, we adapte our approach to support multiple subtree root vertices, leading to a further refinement in the reduction of the algorithm's approximation ratio.

(iii) Benefiting from the preprocessing work, our algorithm's time complexity is comparable to that of existing optimal polynomial time complexity approximation algorithms at the same preprocessing cost.

The organization of this paper is as follows: Section 2 provides a comprehensive review of the literature relevant to our approach. In Section 3, we present our proposed method, **GCTKG**, along with a detailed examination of its time com-

plexity and a formal proof of its approximation ratio. Section 4 begins with an in-depth analysis of the exact solution's characteristics, which informs the development of an enhanced version of **GCTKG**, referred to as **GCTKG+**. Section 5 is dedicated to empirical studies using three well-known public datasets (FB15k [33], FB15k-237 [34], and LinkedMDB [35]) to demonstrate the algorithm's efficacy and the accuracy of its results. Future research directions are explored in Section 6, and the paper concludes with a summary in Section 7.

II. RELATED WORK

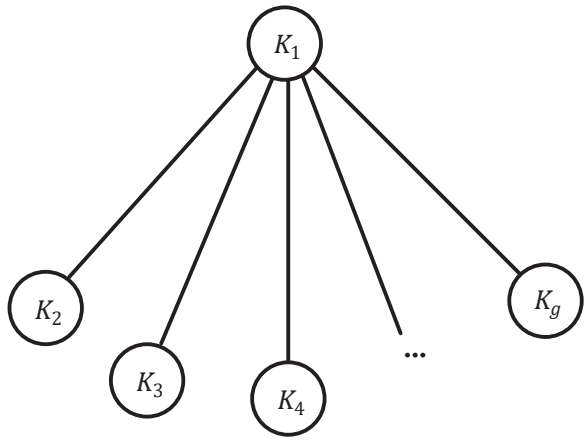


Fig. 3. Abridged general view of Keyword as Root.

There are various approaches to solving the Minimum Group Steiner Tree problem, with the best algorithms in each direction illustrated in Fig. 2. **BANKS** [17] is an integrated database keyword search and browsing system whose underlying algorithm approximates the solution to the minimum-weight GST by consolidating the paths of group vertices into a single common root vertex. **BANKS-II** is an enhancement of **BANKS**, improving performance by expanding the answer ranking model and incorporating a bidirectional search algorithm. **BLINKS** uses precomputed distances and graph partitioning techniques to make the bidirectional search more efficient. The **2-star** algorithm is a sublinear approximation ratio algorithm that improves the solution's precision by selecting secondary subtree roots.

Exact solution algorithms for the Minimum Group Steiner Tree are also subject to continuous iteration and improvement. DPBF is a dynamic programming-based solution capable of finding the minimum-weight GST, while **PrunedDP++** refines DPBF by integrating A* search, offering the most advanced performance for exact solutions to date.

To address the research needs, we categorize the current approaches to solving the Minimum Group Steiner Tree problem into two main types. The first type uses keywords as root vertices, connecting the remaining keyword vertices through the shortest paths to form the minimum Steiner tree. The second type involves algorithms that select an appropriate root (not necessarily a keyword vertex) and construct the corresponding minimum Group Steiner tree.

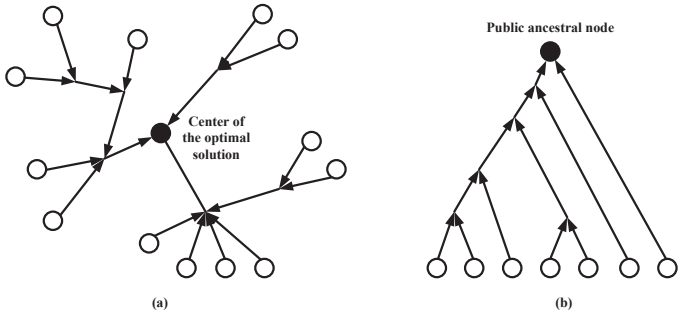


Fig. 4. Abridged general view of Algorithmically Identifying the Optimal Root.

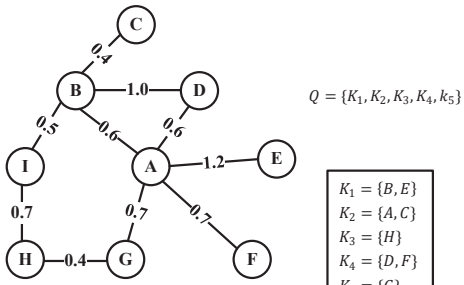
A. Keyword as Root

The schematic diagram of the approximate solution with a keyword as the root is shown in Fig. 3. Methods such as **KeyKG+** fall into this category. The algorithm first selects a group to serve as the keyword root, based on certain evaluation metrics (typically the first query keyword or the smallest group size, etc.), and constructs the minimum spanning tree with this group as the root. When connecting the root to other groups, a virtual group vertex is introduced for each group in the knowledge graph's group set $\mathbb{K}$. This virtual vertex is only connected to vertices within its respective group, with edge weights set to 0. By constructing a minimum spanning tree that connects the virtual vertex of the root group to the virtual vertices of other groups, the algorithm achieves its goal. However, if the knowledge graph is too large to generate a virtual vertex for every group in $\mathbb{K}$, the algorithm must traverse the graph to find the shortest distance between the two closest vertices within a group when constructing the approximate minimum Group Steiner tree. This traversal process increases the overall time complexity of the algorithm.

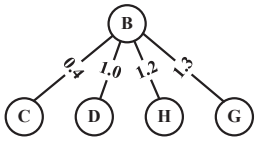
B. Algorithmically Identifying the Optimal Root

The schematic diagram of the approximate solution with other vertices as the root is shown in Fig. 4. Methods such as **2-star** and **BANKS** fall into this category. The algorithm establishes rules (e.g., common root, center of the tree) to conveniently identify the optimal root, and the construction of the minimum Steiner tree is often completed during this process. Compared to using keywords as root, this approach expands the selection range for the root, increasing the likelihood of obtaining a better solution. Moreover, these algorithms typically take into account the tree structure properties of the final result or the characteristics of the optimal solution's tree structure, using this information to guide the design of the approximation algorithm. However, compared to directly selecting a vertex within a group as the root, the expanded search range for identifying the optimal root requires more

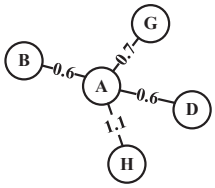
traversals, which leads to an increase in the algorithm's time complexity.



(a) Example of Fig.1



(b) Result of KeyKG+



		A	B	C	D	E	F	G	H	I
K_1	B	0.6	0	0.4	1	1.8	1.3	1.3	1.2	0.5
	E	1.2	1.8	2.2	1.8	0	1.9	1.9	2.3	2.3
K_2	A	0	0.6	1	0.6	1.2	0.7	0.7	1.1	1.1
	C	1	0.4	0	1.4	2.2	1.4	1.7	1.6	0.9
K_3	H	1.1	1.2	1.6	1.7	2.3	1.8	0.4	0	0.7
K_4	D	0.6	1	1.4	0	1.8	1.3	1.3	1.7	1.5
	F	0.7	1.3	1.7	1.3	1.9	0	1.4	1.8	1.8
K_5	G	0.7	1.3	1.7	1.3	1.9	1.4	0	0.4	1.1
		3	3.9	5.1	4.6	7.2	5.2	3.7	4.4	4.7

(c) Result of GCTKG

Fig. 5. Comparition between KeyKG+ and GCTKG.

To achieve a reduction in time complexity while maintaining the high-quality solutions provided by using other vertices as root, we will explore algorithm design strategies that offer both low time complexity and high solution accuracy in high-memory environments. Additionally, we will present the relationship between our algorithm, existing approaches, and the exact solution within the context of this design approach.

III. GCTKG

The core idea of the **GCTKG** algorithm is to identify the central vertex of keyword groups on the knowledge graph G such that the sum of the shortest distances from this central vertex to each group is minimized. The overall functionality of the algorithm is presented in Algorithm 1. In lines 5-12 of the algorithm, for each vertex v on the graph, the $CTChoose$ algorithm is used to compute the sum of the shortest distances to all groups, denoted as $cost_v$, and the set $\mathcal{K}$ consisting of the unique keywords corresponding to these shortest distances. The algorithm then identifies the minimum $cost_v$ as d_{CT} and the corresponding set of keywords $\mathcal{K}_{CT}$. The time cost for this loop is $|V| \times t_{CTChoose}$. After obtaining the central vertex v_{CT}

Algorithm 1 GCTKG

Input: knowledge graph $\mathbb{G} \leftarrow \langle V, E \rangle$,keyword groups $\mathbb{K} \leftarrow \{K_1, K_2, \ldots, K_g\}$,vertex shortest distance table T_D,shortest path predecessor table T_{SP}

Output: A GST of $\mathbb{G}$ covering $K_1, K_2, \ldots, K_g$

1: $v_{CT} \leftarrow -1$
2: $d_{CT} \leftarrow \infty$
3: $\mathcal{K}_{CT} \leftarrow \phi$
4: $\mathbb{T}_{GST} \leftarrow \langle V_{GST} \leftarrow \phi, E_{GST} \leftarrow \phi \rangle$
5: **for all** $v \in V$ **do**
6: $\quad cost_v, \mathcal{K} \leftarrow CTChoose(v, \mathbb{K}, T_D)$
7: $\quad$ **if** $d_{CT} > cost_v$ **then**
8: $\quad\quad d_{CT} \leftarrow cost_v$
9: $\quad\quad v_{CT} \leftarrow v$
10: $\quad\quad \mathcal{K}_{CT} \leftarrow \mathcal{K}$
11: $\quad$ **end if**
12: **end for**
13: **for all** $k \in \mathcal{K}_{CT}$ **do**
14: $\quad p \leftarrow getSP(v_{CT}, k, T_{SP}[v])$
15: $\quad$ Add vertice and edges of p to $\mathbb{T}_{GST}$
16: **end for**
17: **return** $\mathbb{T}_{GST}$

and the corresponding set $\mathcal{K}_{CT}$, the $getSP$ function is used to retrieve the paths from the central vertex to the keywords, with the time cost for this loop being $g \times t_{getSP}$.

Algorithm 2 CTChoose

Input: vertex of knowledge graph v,keyword groups $\mathbb{K} \leftarrow \{K_1, K_2, \ldots, K_g\}$, vertex shortest distance table T_D

Output: A set of vertices $\mathbb{K}$with the closest distance to $v$covering $K_1, K_2, \ldots, K_g$, and the sum of the distances $cost_v$

1: $\mathcal{K} \leftarrow \phi$
2: $cost_v \leftarrow 0$
3: **for all** $K \in \mathbb{K}$ **do**
4: $\quad cost_v \leftarrow \min_{k \in K} getD(v, k, T_D)$
5: $\quad$ Add $\arg\min_{k \in K} getD(v, k, T_D)$ into $\mathcal{K}$
6: **end for**
7: **return** $cost_v, \mathcal{K}$

The core process of the $CTChoose$ algorithm involves $g \times |V|$ invocations of the $getD$ function. Under the experimental conditions of high-memory hardware, we can store the distance table, which consists of the shortest distances between any two vertices in the graph, as an array in memory. This enables us to achieve a constant $O(1)$ time complexity for distance calculations, significantly improving efficiency during the execution of the algorithm.

The $getSP$ function leverages the shortest path preceding vertex table T_{SP} to quickly find the shortest path between two vertices. Each row v in T_{SP} represents the preceding vertex for each vertex in the graph that vertex v can reach. By tracing back through the preceding vertices on the path until the preceding vertex coincides with vertex v, the shortest

Algorithm 3 getSP

Input: For two mutually reachable vertices v and k, the v-th row of the shortest path predecessor table T_{SP}, denoted as $T_{SP}[v]$, stores the information of the predecessors on the shortest path.

Output: The shortest path between vertices v and k.

p is a empty list
while $k! = v$ **do**
    Add k into p
    $k \leftarrow T_{SP}[v][k]$
end while
Add v into p
return p

path connecting the two vertices is obtained. In the worst-case scenario, the time complexity of this algorithm is $O(n)$, where n is the number of vertices in the graph, as the algorithm may need to traverse the entire path from the destination vertex back to the starting vertex.

Algorithm 4 createT

Input:knowledge graph $\mathbb{G} \leftarrow \langle V, E \rangle$

Output: vertex shortest distance table T_D and shortest path predecessor table T_{SP}

$T_D \leftarrow \phi$
$T_{SP} \leftarrow \phi$
for all $node \in V$ **do**
    for all $n \in V$ **do**
        $dist[n] \leftarrow \infty$
        $pred[n] \leftarrow null$
    end for
    $processed \leftarrow \phi$
    $dist[node] \leftarrow 0$
    Priority queue $PQ \leftarrow (0, node)$
    while PQ not null **do**
        $m \leftarrow PQ.pop()$
        if $m \notin processed$ **then**
            $processed \leftarrow processed \bigcup \{m\}$
            for all $nei \in \mathbb{G}.neighbor(m)$ **do**
                if $dist[m] + wt(m, nei) < dist[nei]$ **then**//$wt(u, v)$ is the weight of edge (u, v)
                    $dist[nei] \leftarrow dist[m] + wt(m, nei)$
                    $pred[nei] \leftarrow m$
                    $PQ.push(dist[nei], nei)$
                end if
            end for
        end if
    end while
end for
return T_D, T_{SP}

Constructing the shortest path predecessor table T_{SP} and the vertex shortest distance table T_D requires $|V|$ iterations of Dijkstra's algorithm, as shown in Algorithm 4, which is approximately equivalent to the preprocessing cost of **KeyKG+**.

Using Fig. 1 as a reference to compare the differences between **KeyKG+** and **GCTKG**, as illustrated in Fig. 5, we observe the following: For **KeyKG+**, the chosen root is B, and the keyword group is $\{B, C, D, G, H\}$, resulting in an approximate minimum group Steiner tree with a size of 3.9. On the other hand, **GCTKG** calculates the sum of the shortest distances from each vertex in the graph to each group, as shown in the table in Fig. 5(c), and selects the combination with the lowest cost. Based on the table's results, the selected root is A, and the keywords corresponding to the red values are the vertices closest to the root in each group. Therefore, the selected keyword group is $\{A, B, D, G, H\}$, leading to the construction of an approximate minimum group Steiner tree with a size of 3.

A. Time complexity

The **GCTKG** algorithm performs $|V|$ iterations of $CTChoose$ and g iterations of $getSP$, with a time complexity of $O(|V| \times t_{CTChoose} + g \times t_{getSP})$. The $CTChoose$ algorithm runs g loops to find the minimum distance from each group to vertex v and sums these values. In practice, the length of the list that needs to be computed for the minimum value in each loop will not exceed $\max\{|K_{Q_1}| \mid K_{Q_i} \in Q\}$, which is the maximum size of the groups involved in the query. In the worst-case scenario, this value is taken as $|V|$, making the time complexity of $CTChoose$ $O(g|V|)$. Therefore, the overall time complexity of the **GCTKG** algorithm is $O(g|V|^2 + g|V|) = O(n^2 g)$, which is on par with the **KeyKG+** algorithm.

B. Approximate ratio

Drawing on the proof strategy of the **2-star** algorithm, we will now demonstrate the approximation ratio of our algorithm. Let the approximate solution be denoted as $|T_c| = \min \sum_{K_{Q_i}} \text{dist}(v, K_{Q_i}, G\langle V, E \rangle), v \in V$, where $\sum_{K_{Q_i}} \text{dist}(v, K_{Q_i}, G\langle V, E \rangle)$ represents the sum of the shortest distances from vertex v to each keyword group K_{Q_i} in the set of keyword groups $Q = K_{Q_1}, K_{Q_2}, ..., K_{Q_g}$ on the graph $G = \langle V, E \rangle$, that is, $\text{dist}(v, K_{Q_i}, G\langle V, E \rangle) = \min \text{dist}(v, k, G\langle V, E \rangle), k \in K_{Q_i}$. When a tree $T\langle V_T, E_T \rangle$ is known to be contained within the graph $G\langle V, E \rangle$, an inequality that readily follows is:

$$\sum_{K_{Q_i} \in Q} \text{dist}(v, K_{Q_i}, G\langle V, E \rangle) \leq \sum_{K_{Q_i} \in Q} \text{dist}(v, K_{Q_i}, T\langle V_T, E_T \rangle) \tag{3}$$

Because $V_T \subseteq V$ and $E_T \subseteq E$, the vertices and edges that form the shortest path between v and K_{Q_i} in $G\langle V, E \rangle$ are necessarily in V and E, but not necessarily in V_T and E_T. Now, we identify a set of vertices $K = k_1, k_2, ..., k_g$ with $k_i \in K_{Q_i}$, and a root v_c such that:

$$\sum_{k_i \in K} \text{dist}(v_c, k_i, G\langle V, E \rangle) \leq \sum_{K_{Q_i} \in Q} \text{dist}(v, K_{Q_i}, G\langle V, E \rangle), \quad \forall v \in V \tag{4}$$

In this case, the root v_c is the groups center of **GCTKG**, and $K = \{k_1, k_2, ..., k_g\}$ represents the keyword vertices of the various groups involved in the approximate solution obtained by **GCTKG**. At this vertex, $|T_c| = \sum_{k_i \in K} \text{dist}(v_c, k_i, G\langle V, E\rangle)$. Assuming the exact solution is $T^* = \langle V_{T^*}, E_{T^*}\rangle$, to prove that $\frac{|T_c|}{|T^*|} \leq \frac{g}{2}$, we need to scale up the approximation solution. From equations (3) and (4), we have:

$$\begin{aligned} |T_c| &\leq \min\{ \sum_{K_{Q_i} \in Q} \text{dist}(v, K_{Q_i}, G\langle V, E\rangle), v \in V\} \\ &\leq \min\{ \sum_{K_{Q_i} \in Q} \text{dist}(v, K_{Q_i}, T^*\langle V_{T^*}, E_{T^*}\rangle), v \in V_{T^*}\} \end{aligned} \tag{5}$$

This reduces the proof to showing that $\frac{\min\{\sum_{K_{Q_i} \in Q} \text{dist}(v, K_{Q_i}, T^*\langle V_{T^*}, E_{T^*}\rangle), v \in V_{T^*}\}}{|T^*|} \leq \frac{g}{2}$. Suppose $v_c^* \in V_{T^*}$ satisfies:

$$\begin{aligned} &\sum_{k_{c_i}^* \in K_c^*} \text{dist}(v_c^*, k_{c_i}^*, T^*\langle V_{T^*}, E_{T^*}\rangle) \\ &= \min\{ \sum_{K_{Q_i} \in Q} \text{dist}(v, K_{Q_i}, T^\langle V_{T^*}, E_{T^*}\rangle), v \in V_{T^*}\} \end{aligned} \tag{6}$$

We only need to prove that $\frac{\sum_{k_{c_i}^* \in K_c^*} \text{dist}(v_c^*, k_{c_i}^*, T^*\langle V_{T^*}, E_{T^*}\rangle)}{|T_c|} \leq \frac{g}{2}$, which is to say that when running the **GCTKG** algorithm on the tree of the optimal solution, each edge is counted no more than $\frac{g}{2}$ times. We will now prove this approximation ratio in 2 parts.

1) Characteristics of Group Center: In the process described above, the vertex v_c^* lies on a path P_c in $T^*\langle V_{T^*}, E_{T^*}\rangle$: this path is a smooth, non-branching route, and if it were to be severed, $T^*\langle V_{T^*}, E_{T^*}\rangle$ would be divided into two parts containing m and n keywords, respectively, with $m + n = g$. Among all paths that meet these conditions, P_c is the one that minimizes the absolute difference $|m - n|$, and the vertex v_c^* is the connection vertex that joins the part with the greater number of keywords. Let's assume that $n \leq m$, with the n-keyword part being N, the connection vertex being a, and the m-keyword part being M, with the connection vertex being b.

(i) If the vertex v_c^* is within N, the weight from v_c^* to the keyword vertices within N is reduced compared to being on P_c:

$$\begin{aligned} \Delta_1 \triangleq\ & n \times \text{dist}(b, a, T^*) + \sum_{k_i \in N} \text{dist}(a, k_i, T^*) \\ & - \sum_{k_i \in N} \text{dist}(v_c^*, k_i, T^*) \\ \leq\ & n \times \text{dist}(b, v_c^*, T^*) \\ =\ & n \times \text{dist}(b, a, T^*) + n \times \text{dist}(a, v_c^*, T^*) \end{aligned} \tag{7}$$

This equality holds if and only if the vertex v_c^* is at vertex a. The weight from v_c^* within N to the keyword vertices in M has increased:

$$\begin{aligned} \Delta_2 \triangleq\ & m \times \text{dist}(v_c^*, a, T^*) + \sum_{k_i \in M} \text{dist}(a, k_i, T^*) \\ & - \sum_{k_i \in M} \text{dist}(b, k_i, T^*) \\ =\ & m \times \text{dist}(v_c^*, a, T^*) + m \times \text{dist}(a, b, T^*) \end{aligned} \tag{8}$$

This leads to the following inequality:

$$\begin{aligned} \Delta_1 - \Delta_2 &\leq (n - m) \times \text{dist}(v_c^*, a, T^*) \\ &\quad + (n - m) \times \text{dist}(a, b, T^*) \\ &\leq 0 \end{aligned} \tag{9}$$

The equality holds if and only if $m = n$ and the vertex v_c^* is at vertex a. Therefore, if the vertex v_c^* is not at vertex a, it cannot be within N. If we assume that the vertex v_c^* is at vertex a but $m \neq n$, then $\Delta_1 - \Delta_2 < 0$, meaning that the cost at vertex a is greater than at vertex b; if $m = n$, then vertices a and b are equivalent in effect. This proves that the vertex v_c^* cannot be on the subtree with fewer keywords.

(ii) If the vertex v_c^* is within M but not at vertex b, then v_c^* must be on some subtree M' of M with v_c^* as the root, and the number of keywords on this subtree cannot exceed $\frac{g}{2}$ (if it did, the edge connecting subtree M' would correspond to a smaller $|m - n|$, which would not meet the selection criteria for the edge). Thus, the problem of v_c^* being within M' can be transformed into v_c^* being within N, and we know that v_c^* cannot be within M'. Therefore, v_c^* cannot be within M at a vertex other than b.

(iii) If the vertex v_c^* is on P_c, the cost of the solution is:

$$\begin{aligned} &\sum_{k_i \in N} \text{dist}(a, k_i, T^*) + n \times \text{dist}(v_c^*, a, T^*) + m \times \text{dist}(v_c^*, b, T^*) \\ &\quad + \sum_{k_i \in M} \text{dist}(b, k_i, T^*) \\ &= \sum_{k_i \in N} \text{dist}(a, k_i, T^*) + \sum_{k_i \in M} \text{dist}(b, k_i, T^*) \\ &\quad + n \times \text{dist}(a, b, T^*) + (m - n) \times \text{dist}(v_c^*, b, T^*) \end{aligned} \tag{10}$$

The solution attains its minimum value when v_c^* is at vertex b.

2) Analysis and Calculation of Repetitive Paths: After completing the proof for step 1, it is easy to observe that the number of times the edges on path P_c and within N are repeated will not exceed $\frac{g}{2}$. Within M, if an edge is repeated more than $\frac{g}{2}$ times, it implies that this edge connects to a subtree with more than $\frac{g}{2}$ keywords, but the number of keywords in this subtree cannot exceed m. In this case, the absolute difference in the number of keywords on either side of this edge would be less than $|m-n|$, which conflicts with P_c. Therefore, the number of times edges within M are repeated will not exceed $\frac{g}{2}$. In summary, the number of times paths are recalculated in $\sum_{k_{c_i}^* \in K_c^*} \text{dist}(v_c^*, k_{c_i}^*, T^*\langle V_{T^*}, E_{T^*}\rangle)$ does not exceed $\frac{g}{2}$, which means $\frac{\sum_{k_{c_i}^* \in K_c^*} \text{dist}(v_c^*, k_{c_i}^*, T^*\langle V_{T^*}, E_{T^*}\rangle)}{|T_c|} \leq \frac{g}{2}$. Therefore, $\frac{|T_c|}{|T^*|} \leq \frac{g}{2}$.

IV. GCTKG+

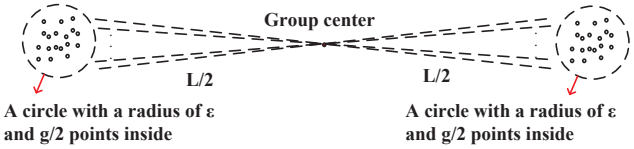


Fig. 6. A worst-case scenario for GCTKG.

Let's analyze the worst-case scenario for **GCTKG**. As illustrated in Fig. 6, suppose the central vertex obtained by the algorithm coincides exactly with the midvertex after the corresponding keyword set is evenly divided, with $\frac{g}{2}$ keywords at each end. Each end's keywords are within a radius of ϵ, meaning the cost of constructing the tree structure to make vertices within each end mutually reachable does not exceed $\frac{g\cdot\epsilon}{2}$. In this case, the cost of the exact solution is:

$$|T^*| = 2 \times \left(\frac{g\cdot\epsilon}{2} + \frac{L}{2}\right) = g\cdot\epsilon + L \tag{11}$$

where L represents the distance between the two ends of the tree. For the cost of the **GCTKG** algorithm, we have:

$$|T_c| \geq 2 \times \left(\frac{L}{2}\cdot\frac{g}{2}\right) = \frac{gL}{2} \tag{12}$$

Thus, the approximation ratio is:

$$\frac{|T_c|}{|T^*|} \geq \frac{\frac{gL}{2}}{g\cdot\epsilon + L} = \frac{g}{2}\cdot\frac{1}{g\cdot\frac{\epsilon}{L}+1} \tag{13}$$

When L is sufficiently large, we have:

$$\lim_{L\to\infty} \frac{g}{2}\cdot\frac{1}{g\cdot\frac{\epsilon}{L}+1} = \frac{g}{2} \tag{14}$$

Therefore, in the worst-case scenario, the approximation ratio of the algorithm is $\frac{g}{2}$.

Next, we consider the tree structure characteristics of the exact solution. The tree obtained by **GCTKG** is less like a branching tree and more similar to the one constructed by **KeyKG+**, where the root is directly connected to the leaf vertices via the shortest paths. In this structure, each subtree is a non-branching path containing only a single keyword vertex (serving as the leaf node). On the other hand, the exact solution's tree would typically have multiple branching vertices within the subtrees, indicating the presence of shared paths that can reduce the overall solution cost. This suggests that a naive idea derived from **GCTKG** would be to optimize the subtrees further. Specifically, by solving for the center of each subtree, duplicate edges within the subtrees can be merged, leading to a reduction in the overall cost of the final solution. This refinement could potentially improve the approximation ratio, leading to more efficient solutions while maintaining the same computational complexity.

In the worst-case scenario, as shown in Fig. 7, we divide the **GCTKG** tree into two parts, labeled Part 1 and Part 2,

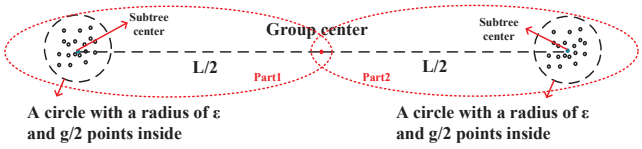


Fig. 7. Optimized tree for worst-case scenario.

Algorithm 5 GCTKG+

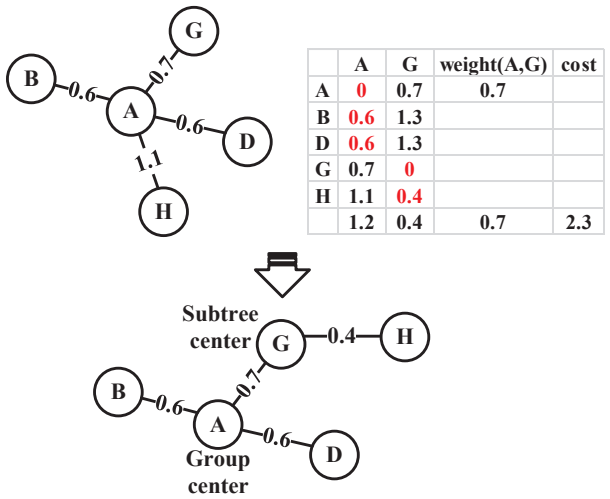
Input: knowledge graph $\mathbb{G} \leftarrow \langle V, E\rangle$,keyword groups $\mathbb{K} \leftarrow \{K_1, K_2, \ldots, K_g\}$,vertex shortest distance table T_D,shortest path predecessor table T_{SP}

Output: A GST of $mathbbG$ covering $K_1, K_2, \ldots, K_g$

Same operation for $GCTKG$ to get $d_{CT}, v_{CT}, \mathcal{K}_{\mathcal{CT}}$.
$\mathcal{K} \leftarrow$ each element in $\mathcal{K}_{\mathcal{CT}}$ is treated as a set
Priority queue $PQ \leftarrow (d_{CT}, (\mathcal{K}, v_{CT}, oldCT = None))$
$CTpath = \phi$
while PQ is not ϕ **do**
    $gpset \leftarrow PQ.pop()$
    if $gpset.v_{CT}$ is None **then**
        if $|gpset.\mathcal{K}| == 1$ **then**
            $CTpath.append([gpset.oldCT, gpset.\mathcal{K}[0], T_D[gpset.oldCT][gpset.\mathcal{K}[0]]])$
        else
            Same operation for $GCTKG$ to get $d_{CT}, v_{CT}, \mathcal{K}_{\mathcal{CT}}$ in $gpset.\mathcal{K}$
            if $v_{CT} == gpset.oldCT$ **then**
                Add all of the $gpset.\mathcal{K}$ to $CTpath$
            else
                $PQ.push(d_{CT}, (gpset.\mathcal{K}, v_{CT}, None))$
                $CTpath.append([gpset.oldCT, v_{CT}, T_D[gpset.oldCT][v_{CT}]])$
            end if
        end if
    else
        for all $nei \in gpset.v_{CT}.neighbors()$ **do**
            Divide $gpset.\mathcal{K}$ into two parts represented as $\mathcal{K}_{nei}$ closer to nei and $\mathcal{K}_{CT}$ closer to $gpset.v_{CT}$.
            Compute the $cost_{nei}$ of nei-rooted tree covering $\mathcal{K}_{nei}$ and the $cost_{CT}$ of v_{CT}-rooted tree covering $\mathcal{K}_{CT}$.
            $d_{CT} = cost_{nei} + cost_{CT} + weight(v_{CT}, nei)$
        end for
        Find the minimum d_{CT} and relevant $\mathcal{K}_{nei}$ and $\mathcal{K}_{CT}$.
        if $d_{CT} > gpset.d_{CT}$ **then**
            Add all of the $gpset.\mathcal{K}$ to $CTpath$
        else
            $PQ.push(d_{CT}, (\mathcal{K}_{nei}, None, gpset.v_{CT}))$
            $PQ.push(d_{CT}, (\mathcal{K}_{CT}, None, gpset.v_{CT}))$
        end if
    end if
end while
Use $CTpath$ to create $\mathbb{T}_{GST}$
return $\mathbb{T}_{GST}$

and enclose them in red dashed ellipses. For both parts, we compute the central vertex, known as the subtree center (in blue), by applying the **GCTKG** method to the internal group centers and keyword sets. This results in an optimized tree where the connection from the group center to the keywords is replaced by a path from the group center to the subtree center, and then from the subtree center to the keywords. We can further apply this process recursively within each subtree, refining the structure until the subtree center stabilizes.

By visualizing the data, we can identify clusters of vertices that are close to one another. These clusters may form subtrees that reduce the overall cost. To replicate this effect algorithmically, we propose a heuristic approach: we traverse the neighboring vertices of the group center and divide the keyword set into two parts. One part, denoted as $\mathcal{K}_{CT}$, consists of keywords closer to the group center, while the other part, denoted as $\mathcal{K}_{nei}$, contains keywords closer to the neighboring vertices. We then calculate the cost of the trees formed by the group center and its neighboring vertices, each connecting to their respective closest keywords. The sum of these costs is added to the weight of the edge connecting the group center to the neighboring vertices. By selecting the combination with the lowest total cost, we designate the corresponding $\mathcal{K}_{nei}$ as the set of keyword vertices that could potentially form a subtree. Finally, we combine $\mathcal{K}_{nei}$ with the group center to find the new center vertex, which we refer to as the subtree center. This approach leads to the design of the **GCTKG+** algorithm.



	A	G	weight(A,G)	cost
A	0	0.7	0.7	
B	0.6	1.3		
D	0.6	1.3		
G	0.7	0		
H	1.1	0.4		
	1.2	0.4	0.7	2.3

Fig. 8. Use GCTKG+ to optimize the example.

The **GCTKG+** algorithm is presented in Algorithm 5, where the method for calculating the center vertex in lines 1 and 11 remains consistent with the **GCTKG** approach. In line 2, the element added to the priority queue, $(d_{CT}, (\mathcal{K}, v_{CT}, oldCT))$, represents the cost of the current tree (or subtree), the set of keywords associated with the current tree (or subtree), the center vertex of the current tree (or subtree), and the parent node of the current subtree, respectively.

Lines 20 to 30 in the algorithm handle the traversal of the neighboring vertices for the center of the current tree (or subtree). After optimization, if the solution cost is greater than the previous cost, it is concluded that no further optimization is needed for the current part, and it is directly included in the final tree structure. Line 8 specifies that if there is only one keyword within the current subtree, no further optimization is needed, and this subtree is included directly in the final constructed tree. Line 12 indicates that if the center of the current subtree remains the parent node, the center no longer moves, so further optimization is unnecessary, and the subtree is added directly to the final tree. Line 15 ensures that if the tree (or subtree) does not meet the above conditions for stopping optimization, it is added to the priority queue, with its center and parent node being incorporated into the final constructed tree.

Referring back to the example in Fig. 1 and the approximate solution in Fig. 5, we designed the solution method for **GCTKG+** as shown in Fig. 8. Based on the solution from **GCTKG**, we traverse the neighbor vertices of the group center A and easily find that vertex G is closer to the set $\{G, H\}$. The optimized heuristic solution yields a cost of 2.3, which is lower than the unoptimized solution, indicating the need to find the center vertex for the set $\{A, G, H\}$. The center vertex for $\{A, G, H\}$ is calculated to be G. Since the subtree (G, H) contains only one keyword (excluding the root G), the optimization stops. The final optimized solution cost is 2.3.

A. Time complexity

In the priority queue loop, the keyword set is divided into two non-empty parts during each iteration, ensuring that the loop will not exceed $O(g)$ iterations. Within each iteration, either the **GCTKG** operation or the operation for traversing neighbor vertices is selected. In the worst-case scenario, traversing the neighbor vertices results in a time complexity of $O(ng)$, but this is no greater than the time complexity of **GCTKG**. Therefore, we can consider the time complexity of each loop iteration to be $O(n^2 g)$. As a result, the worst-case time complexity of the **GCTKG+** algorithm is $O(n^2 g^2)$. However, since the solution produced by **GCTKG+** is progressive (i.e., the solution at each iteration is never worse than the one before), the time complexity, assuming a fixed number of iterations, is $O(n^2 g)$.

B. Approximate ratio

It is evident that after optimizing **GCTKG**, the cost of **GCTKG+** will not exceed that of **GCTKG**, hence the approximation ratio of **GCTKG+** is $\frac{g}{2}$.

V. EXPERIMENTS

Our experimental platform consists of an Ubuntu 22.04 system with 32GB of memory and an Intel i9 14900k CPU. We excluded cache time from the experiments, as it is a one-time operation with minimal impact due to ample memory. The

comparative methods used are **KeyKG+** and **PrunedDP++**. Since **KeyKG+** has already demonstrated its efficiency compared to **BANKS-II**, we did not include **BANKS-II** in this comparison. For algorithms with exponential time complexity [16], we did not perform comparisons beyond the exact algorithm, as such algorithms are impractical for real-world applications. Additionally, this experiment does not include comparisons with Steiner tree algorithms [37]–[40]. All code was implemented in Python for convenience.

A. Datasets

We conducted experiments on three datasets: FB15k, FB15k-237, and LinkedMDB, all of which are in the form of triplets.

1) FB15k: FB15k is a knowledge graph dataset derived from the Freebase dataset, introduced by Bordes et al. in 2013. It contains about 480000 triplets after enhancement. Each triplet in FB15k represents a relationship between a head entity, a relationship, and a tail entity. For example, the triplet (/m/01lsmm, /location/country/capital, /m/02hrh0) represents the relationship between a country (/m/01lsmm) and its capital (/m/02hrh0). The FB15k dataset spans various domains, including people, organizations, locations, movies, books, music, and more. To increase the data size, we combined the training, validation, and test sets for this experiment.

2) FB15k-237: FB15k-237 is an extended version of the FB15k dataset, introduced by Toutanova et al. in 2015. Unlike FB15k, FB15k-237 contains only 237 relations but includes a significantly larger number of triplets, totaling 310,116. This includes 272,115 training triplets, 17,535 validation triplets, and 20,466 test triplets. The relations in FB15k-237 are filtered from the original FB15k dataset, retaining only those with at least 50 training triplets. These relations cover various domains such as people, organizations, locations, movies, books, music, and more, with a broader range of topics compared to the original FB15k dataset. The data processing pipeline used for FB15k-237 follows the same procedure as that for FB15k.

3) LinkedMDB: LinkedMDB is a movie database published in the form of linked data. It contains extensive information about movies, actors, directors, production companies, and more, aiming to provide researchers and developers with a rich, structured, and linkable resource of movie data. The dataset covers a wide range of information from early films to modern movies, with data stored in RDF (Resource Description Framework) format, supporting semantic web technologies that allow the relationships between data to be understood and processed by machines. LinkedMDB includes entity types such as works of film, people, characters, companies, awards, and more, with each entity having multiple attributes like movie titles, release dates, ratings, plot summaries, birthdates, nationalities, and professions for people. Additionally, LinkedMDB provides associations between entities, such as the movies directed by a director, the films an actor has appeared in, and the genres a movie belongs to, which facilitate complex data queries and analysis.

B. Queries

Currently, there is no widely accepted method for generating queries with a large number of groups [36]. To address this, we adopted a random generation approach for the FB15k and FB15k-237 datasets. Specifically, g vertices were randomly selected based on entity IDs, and string matching was used to expand these vertices into groups, ensuring that each keyword corresponds to at least one mapped vertex on the graph. For the LinkedMDB dataset, we applied a size constraint on group generation. In this case, the number of vertices within each group was designed to follow a Gaussian distribution with a mean of 5 and a standard deviation of 2.5. We further ensured that each group contained a minimum of 1 vertex and a maximum of 10 vertices.

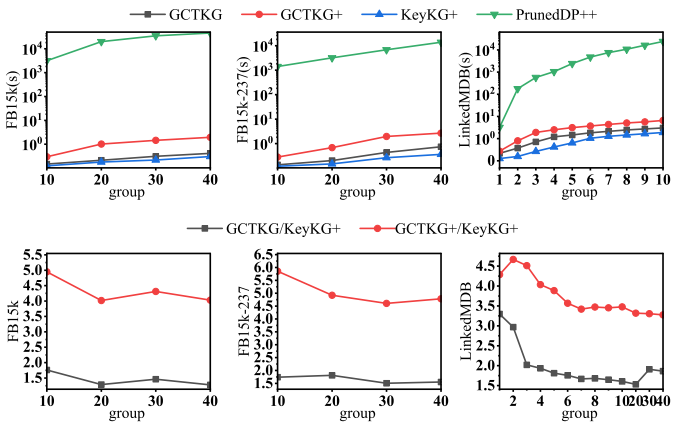


Fig. 9. Query time of 4 methods in 3 datasets.

C. Edge weighting

Existing research methods do not provide a standard way to define edge weights. Here, we aim to set the weights in such a way that they satisfy the triangle inequality (though this is not a strict requirement), and therefore we assign the same value to all edge weights.

D. Query time analysis

When g is particularly large, the time required to find the exact solution for the minimum Steiner tree becomes prohibitively high. The progressive nature of **PrunedDP++** ensures that each iteration produces a better solution than the previous one. Therefore, we divided the 200 queries for each dataset group as follows:

For algorithms with fast runtimes, including **GCTKG+**, **GCTKG**, and **KeyKG+**, all 200 queries were executed across all groups on the three datasets. For **PrunedDP++**, we randomly selected 20 queries from each group to compute the exact solution. For the remaining queries, we set the solution obtained after 100 seconds of execution as the **PrunedDP++** result, simulating its response time in practical engineering scenarios.

For FB15k and FB15k-237, we set g to 10, 20, 30, and 40, generating 200 queries for each value of g using the specified

query generation method. For LinkedMDB, g was set to 1–9, 10, 20, 30, and 40. Within the range of $g = 1$–3, the runtime of **PrunedDP++** remained acceptable in an engineering context, so no 100-second limit was applied. For $g = 20$, 30, and 40, where the runtime of **PrunedDP++** exceeded the acceptable range for the experiment, only the progressive solutions obtained within the 100-second limit were recorded.

To align the theoretical time complexity of **GCTKG+** with that of **KeyKG+**, we imposed a 20-cycle limit on **GCTKG+**.

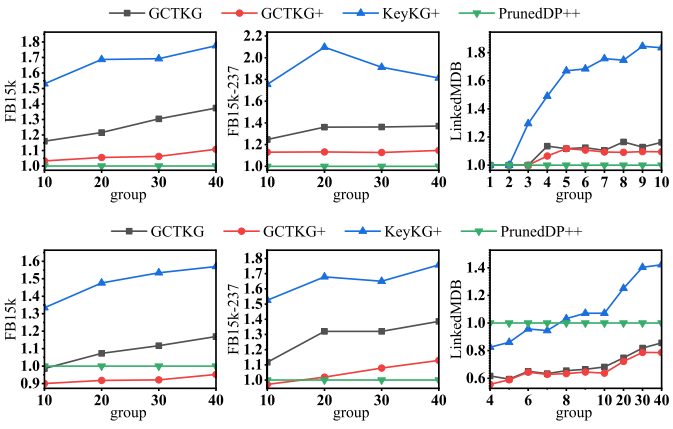


Fig. 10. Approximation ratio of 4 methods in 3 datasets.

The first row of Fig. 9 illustrates the query time costs of the four algorithms across three datasets. The time cost for computing the exact solution is orders of magnitude higher than that of the other algorithms, and its growth follows the exponential progression of g. Among the three approximation algorithms, **KeyKG+** demonstrates the lowest time cost. This difference can be attributed to the additional loop overhead introduced by the **GCTKG** and **GCTKG+** algorithms during the search for central vertices. Specifically, **GCTKG+** incurs higher time costs due to its traversal of all vertices multiple times to identify subtree centers, coupled with the potential for the $getSP$ function to iterate up to $2g$ times.

The second row of Fig. 9 presents the ratios of time costs for **GCTKG+** to **KeyKG+** and **GCTKG** to **KeyKG+**, corresponding to the graphs in the first row. Despite **GCTKG** and **GCTKG+** having higher time costs than **KeyKG+**, the proposed improved algorithm exhibits a consistent upper bound on the time cost ratio relative to **KeyKG+**. On FB15k, the time cost ratio between **GCTKG+** and **KeyKG+** remains below 5, while the ratio between **GCTKG** and **KeyKG+** does not exceed 2. On FB15k-237, the time cost ratio between **GCTKG+** and **KeyKG+** stays under 6, and the ratio for **GCTKG** to **KeyKG+** remains below 2. On LinkedMDB, the time cost ratio for **GCTKG+** to **KeyKG+** does not surpass 5, and for **GCTKG** to **KeyKG+**, it does not exceed 4. These experimental results validate our analysis that the time complexities of the three algorithms are comparable.

E. Approximation ratio

Fig. 10 illustrates the approximation ratios of the four algorithms. The first row of graphs shows the approximation ratios relative to the exact solution. When the number of groups is large, none of the three approximation algorithms reached their theoretical worst-case approximation ratio. For the FB15k and FB15k-237 datasets, no shortest path longer than 4 hops was observed, and for LinkedMDB, no shortest path longer than 5 hops was found. Consequently, the actual theoretical upper bound for the approximation ratio is $\frac{D\cdot(g-1)}{g-1} = D$, where D represents the maximum number of hops in the shortest path between any two vertices in the knowledge graph. Among the three approximation algorithms, **GCTKG** and **GCTKG+** demonstrate higher accuracy compared to **KeyKG+**, with **GCTKG+** aligning more closely with the curve of the exact solution. Notably, in FB15k-237, the approximation ratio of **KeyKG+** exceeded $\frac{D}{2}$, whereas in all three datasets, the approximation ratios of **GCTKG** and **GCTKG+** remained below $\frac{D}{2}$. This finding indirectly confirms that the approximation ratios of the two optimized algorithms do not exceed $\frac{g}{2}$.

The second row of graphs in Fig. 10 depicts the approximation ratios of each algorithm when the solution obtained by **PrunedDP++** within the 100-second limit is treated as the exact solution. Since **PrunedDP++** might not compute the true exact solution within the time constraint, the approximation ratios can occasionally fall below 1, reflecting instances where the approximation algorithms outperform the exact algorithm under the imposed time limit. On the FB15k and LinkedMDB datasets, **GCTKG+** consistently outperforms the exact algorithm within the 100-second constraint, highlighting its effectiveness in these scenarios. In contrast, **KeyKG+** frequently underperforms relative to the exact algorithm within the 100-second limit across all three datasets, further emphasizing the comparative advantage of **GCTKG+** in terms of solution quality under time-constrained conditions.

On the LinkedMDB dataset, the performance difference between **GCTKG** and **GCTKG+** is relatively minor. This can be attributed to differences in graph density and vertex degree distributions across datasets. Both FB15k and FB15k-237 contain approximately 15,000 entity vertices, with FB15k having around 480,000 edges and FB15k-237 having about 300,000 edges, making these dense graphs where shared edges are more prevalent. As a result, the optimization provided by **GCTKG+** is more pronounced in these datasets. In contrast, LinkedMDB is a significantly larger but sparser graph, containing approximately 200,000 vertices and 6,000,000 edges. With a lower proportion of vertices having extremely high degrees, the likelihood of shared edges in the solutions generated by **GCTKG** is comparatively smaller. Consequently, the optimization effect of **GCTKG+** is less noticeable on LinkedMDB, reflecting the impact of graph structure on the algorithm's performance.

VI. Future Work

For the storage of shortest paths for each vertex in the graph, our future work will focus on the following directions:

(i) Efficient Organization of Groups: We plan to design a storage strategy that organizes the shortest path tables for vertices within the same group into a single physical storage

unit. This will enable on-demand reading and facilitate the repeated use of shortest distances directly from disk arrays, especially under limited memory conditions. The goal is to preserve the algorithm's performance while minimizing the overhead of disk reads.

(ii) Compressed Storage for Shortest Distance Tables: Our experiments indicate that the shortest distance tables often contain consecutive repeated values, making them well-suited for lossless compression. We aim to develop a method for compressing these tables that allows direct access to the compressed data during queries. This approach would enable efficient, on-demand memory access to the entire table with minimal memory usage, eliminating the need for time-intensive disk cache operations.

VII. Conclusion

In this paper, we address the minimum Group Steiner tree problem in the context of keyword search within knowledge graphs and propose two algorithms, **GCTKG** and **GCTKG+**. The **GCTKG** algorithm efficiently constructs approximate solutions by identifying group centers, while **GCTKG+** builds on this approach by incorporating subtree center identification to further enhance solution accuracy. Experimental evaluations on three public datasets demonstrate that the proposed algorithms achieve time complexities comparable to existing state-of-the-art polynomial-time approximation algorithms, while delivering improved solution accuracy. Additionally, we analyze the interplay between graph sparsity, the presence of local central vertices, and the notable optimization effects of **GCTKG+**. Finally, we outline future directions for enhancing the methods, including strategies for efficient storage and optimization of shortest path computations.

Acknowledgment

This work was supported by the project (fund number 8KD005(2023)-18).

References

[1] Edge, Darren, et al. "From local to global: A graph rag approach to query-focused summarization." arXiv preprint arXiv:2404.16130 (2024).

[2] Zhao, Penghao, et al. "Retrieval-augmented generation for ai-generated content: A survey." arXiv preprint arXiv:2402.19473 (2024).

[3] Cheng, Gong, and Evgeny Kharlamov. "Towards a semantic keyword search over industrial knowledge graphs." 2017 IEEE international conference on big data (big data). IEEE, 2017.

[4] Han, Shuo, et al. "Keyword search on RDF graphs-a query graph assembly approach." Proceedings of the 2017 ACM on Conference on Information and Knowledge Management. 2017.

[5] Le, Wangchao, et al. "Scalable keyword search on large RDF data." IEEE Transactions on knowledge and data engineering 26.11 (2014): 2774-2788.

[6] Ding, Bolin, et al. "Finding top-k min-cost connected trees in databases." 2007 IEEE 23rd international conference on data engineering. IEEE, 2006.

[7] Ihler, Edmund. "The complexity of approximating the class Steiner tree problem." International Workshop on Graph-Theoretic Concepts in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991.

[8] Reich, Gabriele, and Peter Widmayer. "Beyond Steiner's problem: A VLSI oriented generalization." International Workshop on Graph-theoretic Concepts in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989.

[9] Sun, Yahui, et al. "Finding group steiner trees in graphs with both vertex and edge weights." Proceedings of the VLDB Endowment 14.7 (2021): 1137-1149.

[10] Shi, Yuxuan, Gong Cheng, and Evgeny Kharlamov. "Keyword search over knowledge graphs via static and dynamic hub labelings." Proceedings of The Web Conference 2020. 2020.

[11] He, Hao, et al. "BLINKS: ranked keyword searches on graphs." Proceedings of the 2007 ACM SIGMOD international conference on Management of data. 2007.

[12] Bateman, C. Douglass, et al. "Provably good routing tree construction with multi-port terminals." Proceedings of the 1997 international symposium on Physical design. 1997.

[13] Kacholia, Varun, et al. "Bidirectional expansion for keyword search on graph databases." (2005): 505-516.

[14] Li, Rong-Hua, et al. "Efficient and progressive group steiner tree search." Proceedings of the 2016 International Conference on Management of Data. 2016.

[15] Ihler, Edmund. "The complexity of approximating the class Steiner tree problem." International Workshop on Graph-Theoretic Concepts in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991.

[16] Charikar, Moses, et al. "Rounding via trees: deterministic approximation algorithms for group Steiner trees and k-median." Proceedings of the thirtieth annual ACM symposium on Theory of computing. 1998.

[17] Bhalotia, Gaurav, et al. "Keyword searching and browsing in databases using BANKS." Proceedings 18th international conference on data engineering. IEEE, 2002.

[18] Hwang, Frank K., and Dana S. Richards. "Steiner tree problems." Networks 22.1 (1992): 55-89.

[19] Ihler, Edmund. "Bounds on the quality of approximate solutions to the group Steiner problem." International Workshop on Graph-Theoretic Concepts in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990.

[20] Reich, G., and P. Widmayer. Approximate minimum spanning trees for vertex classes. Technical Report, Inst. fur Informatik, Freiburg Univ, 1991.

[21] Akiba, Takuya, Yoichi Iwata, and Yuichi Yoshida. "Fast exact shortest-path distance queries on large networks by pruned landmark labeling." Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. 2013.

[22] Angelidakis, Haris, Yury Makarychev, and Vsevolod Oparin. "Algorithmic and hardness results for the hub labeling problem." Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, 2017.

[23] Pramanik, Soumajit, et al. "UNIQORN: unified question answering over RDF knowledge graphs and natural language text." Journal of Web Semantics 83 (2024): 100833.

[24] Abujabal, Abdalghani, et al. "Comqa: A community-sourced dataset for complex factoid question answering with paraphrase clusters." arXiv preprint arXiv:1809.09528 (2018).

[25] Dubey, Mohnish, et al. "Lc-quad 2.0: A large dataset for complex question answering over wikidata and dbpedia." The Semantic Web–ISWC 2019: 18th International Semantic Web Conference, Auckland, New Zealand, October 26–30, 2019, Proceedings, Part II 18. Springer International Publishing, 2019.

[26] Christmann, Philipp, et al. "Look before you hop: Conversational question answering over knowledge graphs using judicious context expansion." Proceedings of the 28th ACM International Conference on Information and Knowledge Management. 2019.

[27] Whangbo, Taeg-Keun. "Efficient modified bidirectional A* algorithm for optimal route-finding." New Trends in Applied Artificial Intelligence: 20th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2007, Kyoto, Japan, June 26-29, 2007. Proceedings 20. Springer Berlin Heidelberg, 2007.

[28] Coffman, Joel, and Alfred C. Weaver. "An empirical performance evaluation of relational keyword search techniques." IEEE Transactions on Knowledge and Data Engineering 26.1 (2012): 30-42.

[29] Li, Guoliang, et al. "Progressive keyword search in relational databases." 2009 IEEE 25th International Conference on Data Engineering. IEEE, 2009.

[30] Lappas, Theodoros, Kun Liu, and Evimaria Terzi. "Finding a team of experts in social networks." Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining. 2009.

[31] Majumder, Anirban, Samik Datta, and K. V. M. Naidu. "Capacitated team formation problem on social networks." Proceedings of the 18th ACM SIGKDD international conference on knowledge discovery and data mining. 2012.
[32] Wang, Xinyu, Zhou Zhao, and Wilfred Ng. "Ustf: A unified system of team formation." IEEE Transactions on Big Data 2.1 (2016): 70-84.
[33] Bordes, Antoine, et al. "Translating embeddings for modeling multi-relational data." Advances in neural information processing systems 26 (2013).
[34] Toutanova, Kristina, and Danqi Chen. "Observed versus latent features for knowledge base and text inference." Proceedings of the 3rd workshop on continuous vector space models and their compositionality. 2015.
[35] https://www.cs.toronto.edu/ oktie/linkedmdb/linkedmdb-latest-dump.zip
[36] Miller, Alexander, et al. "Key-value memory networks for directly reading documents." arXiv preprint arXiv:1606.03126 (2016).
[37] Kasneci, Gjergji, et al. "Star: Steiner-tree approximation in relationship graphs." 2009 IEEE 25th International Conference on Data Engineering. IEEE, 2009.
[38] Robins, Gabriel, and Alexander Zelikovsky. "Improved steiner tree approximation in graphs." SODA. 2000.
[39] Byrka, Jaroslaw, et al. "An improved LP-based approximation for Steiner tree." Proceedings of the forty-second ACM symposium on Theory of computing. 2010.
[40] Imase, Makoto, and Bernard M. Waxman. "Dynamic Steiner tree problem." SIAM Journal on Discrete Mathematics 4.3 (1991): 369-384.
[41] Fast Shortest-path Distance Queries on Road Networks by Pruned Highway Labeling