2019 IEEE 35th International Conference on Data Engineering (ICDE)

Keyword-Centric Community Search

Zhiwei Zhang*, Xin Huang*, Jianliang Xu*, Byron Choi*, Zechao Shang‡
* Department of Computer Science, Hong Kong Baptist University, Hong Kong
‡ Department of Computer Science,The University of Chicago, Chicago, IL, USA
{cszwzhang, xinhuang, xujl, bchoi}@comp.hkbu.edu.hk, zcshang@cs.uchicago.edu

***Abstract*—Community search that finds only the communities pertaining to the query input has been widely studied from simple graphs to attributed graphs. However, a significant limitation of previous studies is that they all require the input of query nodes, which makes it difficult for users to specify exact queries if they are unfamiliar with the queried graph. To address this issue, in this paper we study a novel problem of *keyword-centric community search* (KCCS) over attributed graphs. In contrast to prior studies, no query nodes, but only query keywords, need to be specified to discover relevant communities. Specifically, given an attributed graph G, a query Q consisting of query keywords W_Q, and an integer k, KCCS serves to find the largest subgraph of k-core of G that achieves the strongest keyword closeness w.r.t. W_Q. We design a new function of keyword closeness and propose efficient algorithms to solve the KCCS problem. Furthermore, a novel core-based inverted index is developed to optimize performance. Extensive experiments on large real networks demonstrate that our solutions are more than three times faster than the baseline approach, and can find cohesive communities closely related to the query keywords.**

***Index Terms*—keyword-centric, community search, attributed graph**

I. Introduction

Many real-world networks can be represented as graphs, such as social networks, collaboration networks, biological networks, and knowledge graphs. In these graphs, nodes often have important properties described by text, tags, or keywords. For example, authors in collaboration networks have research topics; users in social networks have roles; proteins in protein-protein interaction networks have molecular functions. Thus, it is natural to model these graphs as *attributed graphs*, in which textual attributes are associated with nodes to capture their properties. As an important functional component, a community naturally exists as a group of densely connected nodes in these graphs. Community detection is to discover all communities in the entire graph, which is a fundamental problem for complex network analysis. A related but different problem, called "community search", which identifies only the communities pertaining to the query input, has been widely studied recently [1]–[10].

In the literature, numerous community models for community search have been developed on various kinds of dense subgraphs, including quasi-clique [1], k-core [2]–[4], k-truss [5], [6], and densest subgraph [7]. More recently, attributed community search [8]–[10] has extended the community search problem from simple structural graphs to attributed graphs. However, one significant limitation of all these models

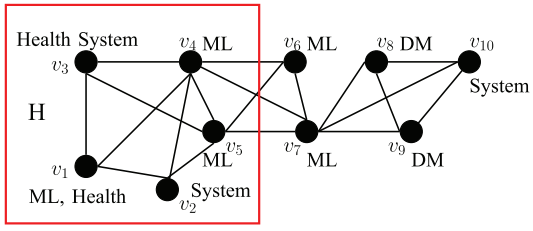


Fig. 1. An attributed graph and keyword-centric dense subgraph

is that they require the input of query nodes. In practice, it might be difficult for users to specify the exact query nodes, as they could be unfamiliar with the background information of the queried graph. Consider the community search on collaboration networks; users may issue a query to find the community working on machine learning and healthcare systems. This query contains no query nodes, but only query keywords. Such problem has not been investigated in any previous studies for community search [1]–[10].

This paper investigates the novel problem of *keyword-centric community search* (KCCS) over attributed graphs. Different from the traditional node-based community search, the keyword-centric community search has only query keywords, but no query nodes, as input. To ensure that the communities discovered are closely related to the query keywords, we propose and optimize a new function of keyword closeness by minimizing a node-to-keyword distance. Our keyword-centric community model has several major advantages. First, it makes users easier to issue queries to discover communities, even if they do not know any node in the graph. Second, it avoids the risk of generating buggy queries that consist of query nodes irrelevant to users' intention. Third, more importantly, it presents a novel approach for mining attributed communities.

Let us look at an example of researcher collaboration network G shown in Fig. 1. Each node v_i represents a researcher and each edge represents the collaboration relationship between two researchers. The keywords associated with the nodes represent their research interests, e.g., v_{10}'s research interest is "System". For illustration, consider a search for attributed communities containing the query keywords W_q ={"ML", "Health", "System"} based on the k-core community model. A k-core is a dense subgraph requiring that each node has at least k neighbors [2]–[4]. Assume that $k = 3$ for our query. To solve this problem, different community

2375-026X/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDE.2019.00045

search models have substantial differences:

- **Structural Community Search**. Structural community search [2]–[4] finds communities using query nodes. One natural method is to find a random set of query nodes, say v_1 and v_{10}, collectively covering all query keywords and then to apply the traditional algorithm of node-based community search [3] with query nodes $\{v_1, v_{10}\}$ as input. In this case, the entire graph G will be returned as the answer, since every node has a degree of 3 or above. However, most members have a long distance to reach the others in the community. Even worse, this community contains nodes with irrelevant attributes (e.g., v_8 and v_9 are associated with "DM" only).
- **Attributed Community Search**. Attributed community search [8], [9] focuses on identifying communities containing the query nodes and sharing homogeneous keywords. Such strict keyword constraints mean that it is likely that no desired communities are found. For instance, using the same query keywords W_q ={"ML", "Health", "System"} and a query vertex v_4 as input, [8] will find a 3-core community as the induced subgraph of G formed by nodes $\{v_4, v_5, v_6, v_7\}$ sharing one common attribute "ML".
- **Keyword-Centric Community Search**. Keyword-centric community search, proposed in this paper, aims to find communities using query keywords. Consider the 3-core community indicated by the rectangle in Fig. 1. Although not all the nodes in the community contain all the query keywords, each node can reach any keyword within one hop. This community is a good candidate to be a collaborative team for interdisciplinary research, which can be discovered by our search.

We formulate our keyword-centric community search (KCCS) queries over an attributed graph G as follows. Given a query $Q = (\mathsf{W_Q}, k)$ consisting of a set of keywords $\mathsf{W_Q}$ and an integer k, KCCS is to find a subgraph H of G such that the minimum degree of each node in H is at least k and the distance between the nodes and all the keywords is minimized. A straightforward solution to KCCS is enumerating all the node combinations that cover the query keywords, and transferring keyword-centric community search into node-based search. Existing query techniques can then be applied to find the k-core community with the highest keyword cohesiveness. However, this is not practical in most cases. Given a query Q with the keywords $\{w_1, w_2, \cdots, w_p\}$, we use V_{w_i} to denote the set of nodes containing the keyword w_i. Then there can exist $2^{|V_{w_i}|}$ node combinations for this single keyword. For all the keywords in Q, there exist $\prod_{i=1}^{p} 2^{|V_{w_i}|}$ different node combinations. Thus, one KCCS query will result in thousands or even millions of node-based community search queries, and the cost would be prohibitively high.

To tackle these challenges, we propose novel, efficient search algorithms to find the KCCS by iteratively removing the nodes with the largest distance to the query keywords from the community. However, this basic solution is still inefficient on large graphs, due to repeated removal iterations in the community exploration procedure. For the query efficiency, we propose an advanced approach based on the keyword-structure measure and the structure-keyword measure. With the keyword-structure measure of a node u, we can determine u's candidate communities. With the structure-keyword measure of u, the communities that u belongs to can be found without checking the distance from u to the keywords. Additionally, to further reduce the search space for the algorithms, we propose several optimizations, including search space reduction and keyword set reduction, along with a core-based inverted index. The following summarizes our contributions made in this paper.

- We formalize the KCCS problem for attributed graphs, where a function of keyword closeness is designed to measure the distance between community nodes and query keywords, and the minimum node degree is used to measure the structural cohesiveness. (Section III)
- By analyzing the properties of k-core and keyword closeness function, we propose the basic approach to the KCCS query. We show that the KCCS community can be found by iteratively removing the nodes with the largest distance to the keywords. (Section IV)
- We propose the keyword-structure measure and the structure-keyword measure, which evaluate the effect between the keyword distance and structural cohesiveness. Based on these, we develop an advanced approach. We further propose several well-designed optimizations and indexes to optimize performance. All these techniques can significantly prune non-answer nodes and keywords w.r.t. a given query. (Section V and Section VI)
- We conduct extensive experiments on real datasets to show that our proposed techniques can find keyword-centric communities effectively and efficiently. (Section VII)

II. Related Works

The related works to our study include keyword search on graphs and community search.

Keyword Search on Graphs: Keyword search related problems focus on the connection between the nodes and the keywords in the query. Different semantics have been proposed, such as r-radius Steiner graph [11], r-clique [12], community [13], and tree semantics [14]–[20]. Among the *tree semantics*, the group Steiner tree semantics is to find the minimum-cost connected tree that contains all keywords [17], but is known to be NP-complete. Due to the hardness of computing the group Steiner tree, the distinct root semantics [16] is introduced to find rooted trees where the cost of a tree is the sum of the shortest distance from the root to the nodes that contain the keywords. The connected tree semantics is first proposed in [18]. Different from the distinct tree semantics, there may exist multiple trees rooted at the same node in G. For the *subgraph semantics*, Kargar and An [12] find the subgraph containing all the keywords in Q. It uses the sum

of the shortest distance between all node pairs as the weight. In [13], it is assumed that fixed center-nodes exist for each community and the problem is to find the communities with center nodes capable of reaching all keywords. In keyword search problems, since the number of individual answers is large, some works focus on *summarizing results*. Li et al. [11] study the problem of clustering based on keywords. In [21], the summarization methods are proposed to cluster the answers to the keyword queries according to the answer graphs. They focus on the paths between keywords and group the paths with similar labels. In contrast to the above, our community model not only optimizes the closeness between query keywords and community nodes, but also requires the community nodes to be densely connected using the k-core model.

Community Search: Given a set of query nodes, community search over graphs is to find a densely connected subgraph that contains the input query nodes [2]. Several different community models have been proposed and studied, including quasi-clique [1], k-core [2], [3], [8], [22], k-truss [5], [6], and densest subgraph [7]. Recently, Fang et al. [8] propose a k-core based attributed community model that finds a k-core containing one query node with strictly homogeneous attributes. Although our work also adopts the k-core community model, our keyword distance function relaxes the constraint of strictly homogeneous keywords using a quantified measure. Huang et al. [9] propose a k-truss based model for attributed community search that finds a connected k-truss with the lowest communication cost and homogeneous attributes. Wang et al. [10] study a k-core based community model in directed attributed graphs. A comprehensive survey of recent studies on community search can be found in [23], [24]. In contrast to these studies, our problem does not need query nodes as input for community search, but focuses on finding the community closest to all the input keywords on the entire graph.

III. Problem Statement

In the following, we first present the notions of keyword closeness and structural cohesiveness, and then propose our keyword-centric community model and problem formulation.

We model an undirected attributed graph as $G = (V, E, \mathcal{L})$, where V and E represent a set of nodes and a set of edges of G, respectively. Each node $u \in V$ is associated with a label, denoted as $\mathcal{L}(u)$, consisting of a set of attribute-value pairs. That is, $\mathcal{L}(u) = \{(\alpha_1 : \beta_1), (\alpha_2 : \beta_2), \ldots\}$, where α_i is an attribute name and β_i is the attribute value of α_i. Given a keyword w, we say a node u contains the keyword w, if w is contained in any of the attribute values in $\mathcal{L}(u)$, denoted by $w \in \mathcal{L}(u)$. For example, consider a node u that represents an author named "David" who conducts research on health systems. The label of u is $\mathcal{L}(u) = \{$(name: "David"), (research: "Health Systems")$\}$, and u contains the keyword "Health". Given a keyword w, we use $V(w) = \{u \in V : w \in \mathcal{L}(u)\}$ to denote the set of nodes containing w in V. Table I summarizes the frequently used notations in this paper.

A. Keyword Closeness

For a subgraph $H \subseteq G$, we use $V(H)$ and $E(H)$ to denote the set of nodes and the set of edges for H, respectively. We define a path in H as a sequence of edges, denoted as $p = (v_1, v_2, \cdots, v_w)$, where $(v_i, v_{i+1}) \in E(H)$ for every v_i $(1 \leq i < w)$. For two nodes $u, v \in V(H)$, we denote by $\mathsf{dist}(u, v, H)$ the length of the shortest path between u and v in H, where $\mathsf{dist}(u, v, H) = +\infty$ if u and v are disconnected. Given a node u and a keyword w, node u can reach keyword w in H if there exists a path from u to a node $v \in V(w)$. The distance between node u and keyword w in H is denoted by $\mathsf{kdist}(u, w, H) = \min_{v \in V(w) \cap V(H)} \mathsf{dist}(u, v, H)$.

Example 3.1: In Fig. 1, the distance $\mathsf{dist}(v_4, v_{10}, G) = 2$ between the two nodes v_4 and v_{10}, because of the shortest path $p = (v_4, v_7, v_{10})$. Given a keyword $w =$"System" and a node v_4, the distance between w and v_4 is $\mathsf{kdist}(v_4, w, G) = 1$, as $v_3 \in V(w)$ and $\mathsf{dist}(v_4, v_3, G) = 1$. □

The keyword distance for multiple query keywords is defined as follows.

Definition 3.1:[Keyword Distance] Given a set of query keywords $\mathsf{W_Q} = \{w_1, w_2, \cdots, w_q\}$ and a node u in H, the keyword distance between u and $\mathsf{W_Q}$ in H is denoted by $\mathsf{kdist}(u, \mathsf{W_Q}, H) = \max_{w \in \mathsf{W_Q}} \mathsf{kdist}(u, w, H)$. □

Intuitively, the smaller the keyword distance, the stronger the relationship between nodes and query keywords. Note that when the context is obvious, we use $\mathsf{kdist}(u, k)$ and $\mathsf{kdist}(u, \mathsf{W_Q})$ to denote $\mathsf{kdist}(u, k, H)$ and $\mathsf{kdist}(u, \mathsf{W_Q}, H)$, respectively. Based on the definition of keyword distance, we define the keyword closeness for a graph as follows.

Definition 3.2:[Keyword Closeness] For a subgraph $H \subset G$ and a set of query keywords $\mathsf{W_Q} = \{w_1, w_2, \cdots, w_q\}$, the keyword closeness of H is denoted by $\mathsf{KwdC}(\mathsf{W_Q}, H) = \max_{\{u \in V(H), w \in \mathsf{W_Q}\}} \mathsf{kdist}(u, w, H)$. □

Note that if a subgraph H does not cover all query keywords of $\mathsf{W_Q}$, then the keyword closeness $\mathsf{KwdC}(\mathsf{W_Q}, H) = +\infty$.

Example 3.2: Consider the attributed graph G in Fig. 1 and query keywords $\mathsf{W_Q} =$ {"ML", "Health" "System"}. For the vertex v_4 in H (indicated by the rectangle), the keyword distance between v_4 and $\mathsf{W_Q}$ is $\mathsf{kdist}(v_4, \mathsf{W_Q}) = \max_{w \in \mathsf{W_Q}} \mathsf{kdist}(v_4, w) = 1$, since for any keyword $w \in \mathsf{W_Q}$, $\mathsf{kdist}(v_4, w) = 1$. The keyword closeness of graph H is $\mathsf{KwdC}(\mathsf{W_Q}, H) = \max_{\{u \in V(H), w \in \mathsf{W_Q}\}} \mathsf{kdist}(u, w, H) = 1$, since for each node u in H, $\mathsf{kdist}(u, \mathsf{W_Q}, H) = 1$. □

B. Structural Cohesiveness

Besides considering the closeness between query keywords and community nodes, we consider the structural cohesiveness among community nodes.For a subgraph $H \subseteq G$ and a node $v \in V(H)$, we denote the set of neighbors of v by $\mathsf{nbr}(v, H) = \{u \in V(H) : (v, u) \in E(H)\}$ and the degree of v in H as $\mathsf{deg}(u, H) = |\mathsf{nbr}(v, H)|$. Consider the example of attributed graph G in Fig. 1. For the subgraph H and a node v_4, $\mathsf{nbr}(v_4, H) = \{v_1, v_2, v_3, v_5\}$ and the degree $\mathsf{deg}(v_4, H) = 4$.

A k-core is defined as the largest subgraph H of G such that every vertex u has a degree of at least k in H (i.e., $\mathsf{deg}(u, H) \geq k$). The whole graph G in Fig. 1 is 3-core. Due to the structural properties and efficient computation of k-core, several community models adopt the k-core model for the structural cohesiveness [2]–[4]. Therefore, we also advocate the k-core model to define structural cohesiveness.[1]

Definition 3.3:[Structural Cohesiveness] For a subgraph $H \subset G$, the structural cohesiveness of H is defined as the minimum vertex degree in H, denoted by $\mathsf{StrC}(H) = \min_{u \in V(H)} \mathsf{deg}(u, H)$. □

C. Keyword-Centric Community Model

Combining the keyword closeness of $\mathsf{KwdC}(\mathsf{W_Q}, H)$ and structural cohesiveness of k-core, we define a keyword-centric community as follows.

Definition 3.4:[Keyword-Centric Community] Given an attributed graph G and a query $Q = (\mathsf{W_Q}, k)$ consisting of a set of keywords $\mathsf{W_Q} = \{w_1, w_2, \cdots, w_q\}$ and an integer k, a keyword-centric community H is the maximum subgraph of G, satisfying the following two conditions:

1) Structural cohesiveness: $\mathsf{StrC}(H) \geq k$.

2) Keyword closeness : There does not exist any subgraph H', such that $H' \not\subset H$, $\mathsf{StrC}(H') \geq k$, and $\mathsf{KwdC}(\mathsf{W_Q}, H') < \mathsf{KwdC}(\mathsf{W_Q}, H)$. □

In terms of structural cohesiveness, Condition (1) requires that the community is densely connected such that each node has at least k neighbors. In terms of keyword closeness, Condition (2) ensures that the community is as close as possible to all the query keywords. Note that this condition also implies that the community should cover all query keywords of $\mathsf{W_Q}$. If H does not contain all the keywords in $\mathsf{W_Q}$, the closeness would be $\mathsf{KwdC}(\mathsf{W_Q}, H) = +\infty$. By Def. 3.4, we also call the keyword-centric community H for query Q as $\mathsf{KC}_k(Q, G)$. The KCCS problem studied in this paper is formulated below.

Problem Statement: Given an attributed graph G and a query $Q = (\mathsf{W_Q}, k)$, the KCCS-problem is to find the keyword-centric community $\mathsf{KC}_k(Q, G)$.

Example 3.3: For the graph G in Fig. 1 and a query $Q = (\mathsf{W_Q}, 3)$ with $\mathsf{W_Q}$ = {"ML", "Health", "System"}, the subgraph H (indicated by the rectangle) is the keyword-centric community $\mathsf{KC}_k(Q, G)$, since H is the largest subgraphs such that (1) $\mathsf{StrC}(H) \geq 3$, as $\mathsf{deg}(u, H) \geq 3$ for $u \in V(H)$; and (2) H achieves the strongest keyword closeness as $\mathsf{KwdC}(\mathsf{W_Q}, H) = 1$. □

IV. Basic Approach

In this section, we introduce the basic approach to find the keyword-centric community for the KCCS query. For the self-containment of the paper, we first discuss k-core computation briefly. Then we present the properties of keyword distance and the basic approach for the KCCS query.

[1]Our proposed community model and query algorithms are not confined to the k-core structure, and can be extended to other cohesive structures, such as k-truss.

TABLE I
Frequently Used Notations

Notations	Definitions
$G(V, E, \mathcal{L})$	Attributed graph G with the node set V, the edge set E and label function as $\mathcal{L}$.
$\mathsf{kdist}(u, \mathsf{W_Q}, H)$	Distance between node u and query keywords $\mathsf{W_Q}$ in H.
$\mathsf{KwdC}(\mathsf{W_Q}, H)$	Keyword closeness of H to $\mathsf{W_Q}$ in Def. 3.2.
$\mathsf{StrC}(H)$	Structural cohesiveness of H in Def. 3.3.
$G^d(V^d, E^d)$	Induced graph of the nodes with $\mathsf{cd}^{\mathsf{L}}(u) \leq d$ for $u \in V(G)$.
$\mathsf{KC}_k^d(Q, G)$	The maximum subgraph H_d of G, such that $\mathsf{KwdC}(Q, H_d) = d$ and $\mathsf{StrC}(H_d) = k$.
$\mathsf{ksd}(u, Q, G)$	The minimal number of d, such that there exists a subgraph $H \subset G$ containing u with $\mathsf{StrC}(H) \geq k$ and for each node $v \in H$, and $\mathsf{cd}^{\mathsf{L}}(v) \leq d$.
$\mathsf{skd}(u, w, G)$	The minimal number of d, such that there exist a path p from u to the keyword w with $\mathsf{len}(p) \leq d$, and for each node $v \in p$, $\mathsf{cd}^{\mathsf{L}}(v) \leq d$.

A. k-core Computation

According to Condition (1) in Def. 3.4, for a KCCS query, the minimum degree of all the nodes in the result graph should be larger than or equal to k. We use G_k to denote the k-core of the graph G. For each node u, the core number of u, denoted as $\mathsf{C}(u)$, is the largest value for k, such that $u \in V(G_k)$. Thus, a k-core is the induced subgraph of the nodes, whose core number is not smaller than k. In general, computing the k-core is equivalent to computing the core number for every node.

The idea behind computing k-core is that, each node $u \in V(G)$ will be removed from G if $\mathsf{deg}(u, G) < k$. After that, the degree of the node $v \in \mathsf{nbr}(u)$ will be reduced by 1, and v will be removed further if $\mathsf{deg}(v) < k$. This procedure continues until $\mathsf{deg}(u) \geq k$ for each node u in the remaining graph. The complexity to compute the k-core is $O(E(G))$ [25].

Example 4.1: Consider the graph shown in Fig. 2. The whole graph is 2-core because $\mathsf{deg}(v) \geq 2$ for $v \in V(G)$. Also, if we aim to find the 3-core of the graph, v_{13} is the first to be removed since $\mathsf{deg}(v_{13}) < 3$. After removing v_{13}, the degree of v_1 and v_3 will be reduced by 1. After that, the minimum degree of the nodes in the remaining graph is 3. Thus, the remaining graph is a 3-core, as shown in Fig.3(a). □

In the following, we discuss the properties of keyword closeness and the basic approach for the KCCS query.

B. Properties of Keyword Closeness

According to Def. 3.4, the KCCS query is to find the subgraph $H = \mathsf{KC}_k(Q, G)$ of G with the strongest keyword closeness of Q. For ease of illustration, we use $\mathsf{KC}_k^d(Q, G)$ to denote the maximum subgraph H, on the condition that $\mathsf{StrC}(H) \geq k$ and $\mathsf{KwdC}(Q, H) \leq d$. Then, the KCCS community is $\mathsf{KC}_k^{d_{min}}$, in which d_{min} is the minimum value of d such that $\mathsf{KC}_k^d(Q, G)$ contains all keywords in W_Q. In the following, we concentrate on the computation of $\mathsf{KC}_k^d(Q, G)$. Note that, given a number d, there will exist a unique $\mathsf{KC}_k^d(Q, G)$. The reason is that, if there are more than one

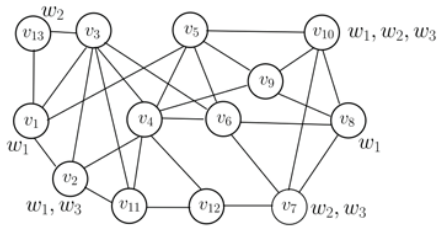


Fig. 2. A Graph as the Running Example of the Algorithms

Algorithm 1 Basic Approach for KCCS Query

Input: A Graph G, a KCCS query $Q(\mathsf{W_Q}, k)$;
Output: The keyword-centric community of G w.r.t. Q;
1: $G \leftarrow$ *CoreComp*(G, k), $d \leftarrow \mathsf{KwdC}(Q, G)$, $\mathsf{KC}_k^d \leftarrow G$;
2: **if** G does not contain all keywords in Q **then**
3: return G;
4: $G' \leftarrow G$;
5: **while** $G' \neq \emptyset$ and G' contains all keywords in Q **do**
6: remove u from G' if $\mathsf{kdist}(u) \geq d$;
7: $G' \leftarrow$ *CoreComp*(G', k);
8: $d' \leftarrow$ *UpdateDis*(G', Q, d);
9: $d \leftarrow d', \mathsf{KC}_k^{d'} \leftarrow G'$ if $d' < d$;
10: return $\mathsf{KC}_k^d(Q, G)$ if $\mathsf{KC}_k^d(Q, G) \neq \emptyset$ with the smallest d;
11: **procedure** UpdateDis(G, Q, d)
12: compute $\mathsf{kdist}(u, \mathsf{W_Q}, G)$ for each node $u \in G$;
13: return $\max_{u \in G} \mathsf{kdist}(u, \mathsf{W_Q}, G)$;

graph $G_1, G_2, \cdots$ satisfying the conditions that $\mathsf{StrC}(G_i) \geq k$ and $\mathsf{KwdC}(Q, G_i) = d$, we can always merge them as H. It can be found that $\mathsf{StrC}(H) > k$ and $\mathsf{KwdC}(Q, H) = d$.

According to Def. 3.2, given a graph G and a KCCS query $Q(\mathsf{W_Q}, k)$, assume that we aim to check whether $u \in V(\mathsf{KC}_k^d(Q, G))$. A basic approach is computing the k-core H of G, and then $\mathsf{kdist}(u, w, H)$ is computed for each keyword $w \in \mathsf{W_Q}$. If $\mathsf{KwdC}(Q, H) \leq d$, we have $\mathsf{KC}_k^d(Q, G) \subset H$. Otherwise, if $\mathsf{kdist}(u, \mathsf{W_Q}, G) > d$, u cannot belong to any $\mathsf{KC}_k^d(Q, G)$. With these properties, we design the basic algorithm for the KCCS query, which reduces the keyword closeness from the largest $\mathsf{kdist}(u, \mathsf{W_Q}, G)$ for $u \in V(G)$.

C. Basic Algorithm for the KCCS Query

The basic algorithm for the KCCS query is described in Algorithm 1. We use *CoreComp*(G, k) to denote the procedure of computing k-core in G based on the discussion in Section IV-A. Initially, the k-core of the graph G and $\mathsf{KwdC}(Q, G)$ is computed as G and d, respectively (line 1). If G does not contain all the keywords, it has $\mathsf{KwdC}(Q, G) = +\infty$. In this scenario, the k-core is returned (lines 2-3). Otherwise, the algorithm iteratively checks whether there exists $\mathsf{KC}_k^d(Q, G)$ containing all keywords in descending order of d. At first, G' is used to keep the remaining graph (line 4). If $\mathsf{kdist}(u, Q, G')$ is larger than or equal to $\mathsf{KwdC}(Q, G)$, which is d, the node u and all the incident edges are removed (lines 6). The k-core in the remaining graph is then computed (line 7), and the keyword distance in the remaining graph is updated as d' (line 8). If d' is smaller than d, d is updated as d', and the remaining graph is $\mathsf{KC}_k^{d'}$ (line 9). The non-empty graph with the smallest d is returned (line 10). In the procedure of updating the distance, $\mathsf{kdist}(u, \mathsf{W_Q}, G)$ is re-computed and the maximum $\mathsf{kdist}(u, \mathsf{W_Q}, G)$ is returned (lines 12-13).

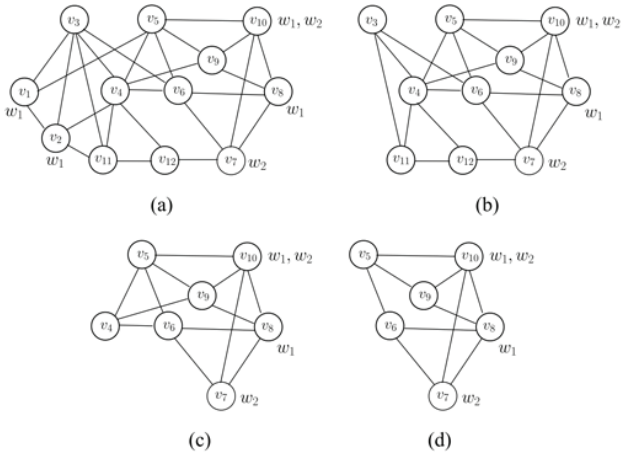


Fig. 3. Illustration of the Algorithms

Example 4.2: Consider the graph shown in Fig. 2 and the KCCS query $Q(\{w_1, w_2\}, 3)$. For clarity, we only show the labels of w_1 and w_2 in the figure. According to the discussion in Example 4.1, the corresponding 3-core is shown in Fig. 3(a). The graph G in Fig. 3(a) contains the query keywords w_1, w_2. Hence, it is a candidate graph. Also, it can be computed that $\mathsf{kdist}(v_2, w_2, G) = 3$, and $\mathsf{KwdC}(Q, G) = 3$. Thus, the graph shown in Fig. 3(a) is KC_3^3. In order to check whether there exists KC_3^2, in which $\mathsf{KwdC}(Q, G) \leq 2$, each node u with $\mathsf{kdist}(u, Q, G) > 2$ is removed. After removing v_2, it has $\deg(v_1) = 2$ and v_1 is removed afterwards. The remaining graph is shown in Fig. 3(b). After removing v_1, v_2, for node v_{11}, $\mathsf{kdist}(v_{11}, w_1, G)$ needs to be updated to $\mathsf{kdist}(v_{11}, w_1, G) = 3$. Thus, v_{11} is also removed. After removing v_{11}, $\deg(v_3)$ becomes 2. Hence, v_3 is further removed and the remaining graph is shown in Fig. 3(c). For the graph G in Fig. 3(c), $\mathsf{KwdC}(Q, G) \leq 2$ and it is a 3-core. This procedure continues and the final result is shown in Fig. 3(d). It is a 3-core, and $\mathsf{KwdC}(Q, G) = 1$. □

In Algorithm 1, since $\mathsf{kdist}(u)$ can be increased after removing nodes, $\mathsf{kdist}(u)$ for each keyword $w_i \in \mathsf{W_Q}$ must be updated in each iteration. Thus, in each iteration, it will cost $|\mathsf{W_Q}| * (|V(G)| + |E(G)|)$ to update the distance between each node and each keyword. Furthermore, the number of the iterations is bounded by $|V(G)|$. Consequently, the complexity of Algorithm 1 is shown in the following theorem.

Theorem 4.1: *Given a graph G and the* KCCS *query $Q(\mathsf{W_Q}, k)$, the complexity of Algorithm 1 is $O(|\mathsf{W_Q}| \cdot n \cdot (m + n))$, where $n = V(G)$ and $m = E(G)$.* □

V. The Advanced Approach for the KCCS Query

In this section, we introduce the advanced approach for the KCCS query. Unlike the basic approach, which reduces the keyword closeness from the largest $\mathsf{kdist}(u, Q, G)$ for

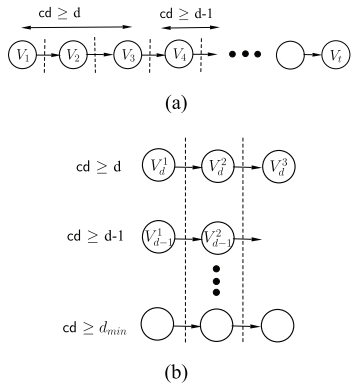


Fig. 4. Illustration of the Advanced Approach

$u \in V(G)$, our advanced approach functions in a bottom-up fashion by exploiting the new keyword-structure measure and structure-keyword measure.

For the basic approach, as illustrated in Fig. 4(a), in each iteration, it can only decide whether the node u belongs to $\mathsf{KC}_k^d(Q,G)$ with $d = \mathsf{KwdC}(\mathsf{W_Q}, G)$. In this scenario, the number of the iterations needed in Algorithm 1 is large. Further, the shortest distances between nodes and keywords must be re-computed in each iteration. Thus, the cost is quite high. Unlike the basic approach, the motivation of our advanced approach is that, if it can be determined that the node u is possible to appear in $\mathsf{KC}_k^d(Q,G)$ with $d < \mathsf{KwdC}(\mathsf{W_Q}, G)$, then we can set u as a candidate node for $\mathsf{KC}_k^d(Q,G)$. Thus, in the iteration of computing $\mathsf{KC}_k^d(Q,G)$, we only need to consider the induced subgraph of the candidate nodes. Furthermore, as illustrated in Fig. 4(b), we aim to determine the bound of d with the corresponding KC_k^d to which the node is possible to belong. These minimum values for different node sets, denoted as $V_{d'}^i$ in Fig.4(b), can be determined in one iteration. Thus, the number of iterations needed to achieve the result could be reduced significantly, and the cost will also be saved.

A. Problem Analysis

For ease of illustration, we use $\mathsf{cd}(u,Q,G)$ ($\mathsf{cd}(u)$ for short) to denote the minimum value of d such that $u \in \mathsf{KC}_k^d(Q,G)$. Given a graph G and a KCCS query, there are two factors that determine whether the node u appears in $\mathsf{KC}_k^d(Q,G)$. The first factor is about the influence from the keyword measure to the structural measure. Specifically, given the query Q, assume that u belongs to the k-core of G, and after removing a node v from G, there does not exist any k-core that contains u. In this scenario, whether u appears in $\mathsf{KC}_k^d(Q,G)$ is determined not only by $\mathsf{kdist}(u,\mathsf{W_Q})$, but also by $\mathsf{kdist}(v,\mathsf{W_Q})$. The reason is that, for the subgraph $H = \mathsf{KC}_k^d(Q,G)$ with $d < \mathsf{kdist}(v,\mathsf{W_Q})$, it has $v \notin V(H)$ due to the keyword closeness condition. In this scenario, u cannot appear in H either because $\mathsf{StrC}(H) \geq k$ and u does not belong to any k-core without v. The second factor concerns from the structural measure to the keyword measure. Consider a node u and all the paths $p_1, p_2 \cdots p_t$ from u to a keyword $w_i \in W_Q$. If we have found that for a number d, there exist nodes $v_i \in p_i$ $(1 \leq i \leq t)$ on the condition that $\mathsf{cd}(v_i) > d$, then $\mathsf{cd}(u)$ is determined not only by $\mathsf{kdist}(u,\mathsf{W_Q})$, but also by $\mathsf{cd}(v_i)$. The reason is that, if u appears in the subgraph $H = \mathsf{KC}_k^{d'}(Q,G)$ with $d' < d$, at least one node in the path p_i will not appear in H. In this scenario, u cannot reach the keyword w_i in H. Thus $u \notin V(H)$.

Considering the factors discussed above, we propose two evaluations which are the keyword-structure measure and structure-keyword measure. In the following, we use $\mathsf{cd}^{\mathsf{L}}(u,G)$ ($\mathsf{cd}^{\mathsf{L}}(u)$ for short) to denote the existing pruning bound of $\mathsf{cd}(u)$ in the graph G. In other words, u cannot belong to $\mathsf{KC}_k^d(Q,G)$ if $d < \mathsf{cd}^{\mathsf{L}}(u)$. Notice that for $\mathsf{cd}^{\mathsf{L}}(u)$ of each node u, it always has $\mathsf{cd}^{\mathsf{L}}(u) \geq \mathsf{kdist}(u,\mathsf{W_Q},G)$. The reason is that, given $H = \mathsf{KC}_k^d(Q,G)$, if u cannot reach $w_i \in \mathsf{W_Q}$ within d in G, u cannot reach w_i within d in H either. Thus, we have the following property for $\mathsf{cd}^{\mathsf{L}}(u)$ in G.

Property 1: $\mathsf{cd}^{\mathsf{L}}(u) \geq \mathsf{kdist}(u,\mathsf{W_Q},G)$.

B. Keyword-Structure Measurement

In this subsection, we first discuss the keyword-structure measurement, denoted as $\mathsf{ksd}(u)$, for each node u. According to the structural cohesiveness, we aim to find a subgraph H with $\mathsf{StrC}(H) \geq k$. Considering two nodes $u, v \in V(G)$, if v is removed from G and after that, $\deg(u) < k$, then u will also be removed from G. In this scenario, it indicates that there is no k-core H on the condition that $u \in V(H)$ and $v \notin V(H)$. Thus, for the keyword-structure measure of the node u, given a graph G and a query Q, $\mathsf{ksd}(u,Q,G)$ ($\mathsf{ksd}(u)$ for short) is the minimal number of d, such that there exists a subgraph $H \subset G$ containing u with $\mathsf{StrC}(H) \geq k$ and $\mathsf{cd}^{\mathsf{L}}(v,\mathsf{W_Q},H) \leq d$ for $v \in V(H)$. Considering the situation that after removing each node u with $\mathsf{cd}^{\mathsf{L}}(u) \geq d$ from G, the k-core of the remaining graph is H and we have the following theorem.

Theorem 5.1: *Given a graph G and a* KCCS *query $Q(\mathsf{W_Q},k)$, S is the induced subgraph of the node u with $\mathsf{cd}^{\mathsf{L}}(u) \leq d$, and the k-core of S is H. Then for any node $v \in V(G) - V(H)$, it has $\mathsf{ksd}(v) \geq d$.* □

Proof Sketch: If there exists a node $v \in V(G) - V(H)$ with $\mathsf{ksd}(v) < d$, then there exists a subgraph H' containing v, such that $\mathsf{StrC}(H') \geq k$ and for each node $h \in V(H')$, $\mathsf{cd}^{\mathsf{L}}(h) = d' < d$. Since S is the induced subgraph of the node u with $\mathsf{cd}^{\mathsf{L}}(u) \leq d$, it has $H' \subset S$. As H is the k-core of S, H' is the subgraph of H. Thus, we have $v \notin H'$ for the node $v \in V(G) - V(H)$. This contradicts the assumption. □

For each node u, $\mathsf{ksd}(u)$ can also be a pruning bound for $\mathsf{cd}(u)$, which is shown in the following theorem.

Theorem 5.2: *Given a* KCCS *query $Q(\mathsf{W_Q},k)$ and a graph G with $\mathsf{cd}^{\mathsf{L}}(u)$ for $u \in V(G)$, it has $\mathsf{ksd}(u) \leq \mathsf{cd}(u)$.* □

Proof Sketch: For the node u with $\mathsf{cd}(u) = d$, there exists a subgraph $H \subset G$ containing u, such that $\mathsf{cd}(v) = d$ for each node $v \in V(H)$ and $\mathsf{StrC}(H) \geq k$. As $\mathsf{cd}^{\mathsf{L}}(v) \leq \mathsf{cd}(v)$, H is also a subgraph with $\mathsf{cd}^{\mathsf{L}}(v) \leq d$ and $\mathsf{StrC}(H) \geq k$. As $H \subset G$, it has $\mathsf{ksd}(u,Q,G) \leq \mathsf{ksd}(u,Q,H) \leq d$. Thus, the theorem is proven. □

According to Theorem 5.2, u cannot belong to any $\mathsf{KC}_k^d(Q,G)$ if $d < \mathsf{ksd}(u)$. Thus $\mathsf{ksd}(u)$ can also be used as the pruning bound of $\mathsf{cd}(u)$, and to update $\mathsf{cd}^{\mathsf{L}}(u)$ if $\mathsf{cd}^{\mathsf{L}}(u) < \mathsf{ksd}(u)$. Thus, in the procedure to compute $\mathsf{cd}(u)$, the nodes can be removed in the decreasing order of their cd^{L} value. After removing the node u with $\mathsf{cd}^{\mathsf{L}}(u) \geq d$, if a node h is also removed in the k-core computation procedure, $\mathsf{cd}^{\mathsf{L}}(h)$ and $\mathsf{ksd}(h)$ can be updated as d according to Theorem 5.1 and Theorem 5.2. As $\mathsf{cd}^{\mathsf{L}}(h)$ can be updated as $\mathsf{ksd}(h)$, in the following, we focus on the relationship between the structure-keyword measure and the cd^{L} value for the nodes.

C. Structure-Keyword Measure

In this subsection, we propose the structure-keyword measure, denoted as $\mathsf{skd}(u,w,G)$. Given a graph G and a KCCS query Q, if $\mathsf{kdist}(u,\mathsf{W_Q}) \leq d$, then u can reach all the keywords in $\mathsf{W_Q}$ within d. However, considering the shortest path p from u to a keyword $w \in \mathsf{W_Q}$, if there exists a node $v \in p$ with $\mathsf{cd}^{\mathsf{L}}(v) > d$, then the path p cannot appear in $\mathsf{KC}_k^d(Q,G)$ as $v \notin V(\mathsf{KC}_k^d(Q,G))$. Thus, even if $\mathsf{kdist}(u,\mathsf{W_Q})$ is small in G, $\mathsf{cd}(u,Q,G)$ can also be quite large because of the $\mathsf{cd}^{\mathsf{L}}(v)$ of the node v on the paths from u to the keywords.

Considering these, we propose $\mathsf{skd}(u,w,G)$ ($\mathsf{skd}(u)$ for short) as the minimal number of d, such that there exists a path p from u to the keyword w at the condition that, $\mathsf{len}(p) \leq d$, and for each node $v \in p$, $\mathsf{cd}^{\mathsf{L}}(v) \leq d$. We use $\mathsf{skd}(u,\mathsf{W_Q},G)$ to denote $\max(\mathsf{skd}(u,w_i,G))$ for $w_i \in \mathsf{W_Q}$. With $\mathsf{skd}(u)$, we have the following theorem.

Theorem 5.3: *Given a graph G and a* KCCS *query Q, for each keyword $w_i \in \mathsf{W_Q}$ and $u \in V(G)$, it has $\mathsf{cd}(u) \geq \max(\mathsf{skd}(u,w_i,G))$ for $1 \leq i \leq |\mathsf{W_Q}|$.* □

Proof Sketch: Assume that $d = \mathsf{cd}(u)$ and $u \in V(\mathsf{KC}_k^d(Q,G))$. For each path p from u to a keyword $w_i \in \mathsf{W_Q}$ in the graph $\mathsf{KC}_k^d(w,G)$, p is also a path in G. If $d < \mathsf{skd}(u,w_i,G)$, then among all the nodes in the path p, there exists at least one node u, such that $\mathsf{cd}^{\mathsf{L}}(u) = \mathsf{skd}(u,w_i,G) > d$ or $\mathsf{len}(p) > d$. This contradicts to the assumption that $p \subset \mathsf{KC}_k^d(Q,G)$. Thus, the theorem is proven. □

Example 5.1: Consider the graph shown in Fig. 3(a). Assume that for each node in the graph, $\mathsf{cd}^{\mathsf{L}}(v_1) = \mathsf{cd}^{\mathsf{L}}(v_2) = 3$ and $\mathsf{cd}^{\mathsf{L}}(v_i) = 2$ $(3 \leq i \leq 12)$. Then, it can be derived that $\mathsf{skd}(v_{11},w_1) = 3$. Specifically, it can be checked that given $d = 2$, v_{11} cannot reach w_1 on a path p with $\mathsf{len}(p) \leq 2$, and each node v in the path has $\mathsf{cd}^{\mathsf{L}}(v) \leq 2$. Given d as 3, v_{11} can reach w_1 with the edge (v_{11},v_2) and for the nodes v_2, v_{11}, their cd^{L} values are not larger than 3. □

D. Computation of the Structure-Keyword Measure

In the following, we discuss the properties and the computation of the skd value for the nodes.

We focus on two aspects. The first concerns the keyword distance with $\mathsf{cd}^{\mathsf{L}}(u)$ and the corresponding $\mathsf{skd}(u)$ of the node u. The second is the relationship between $\mathsf{skd}(u)$ and the keyword centric communities. For ease of illustration, we use $V^{\underline{d}}$ to denote the node set $\{u|\mathsf{cd}^{\mathsf{L}}(u) \leq d\}$ and $G^{\underline{d}}(V^{\underline{d}},E^{\underline{d}})$ to denote the induced graph of $V^{\underline{d}}$. In this scenario, $\mathsf{KC}_k^d(Q,G) \subset G^{\underline{d}}$ if $\mathsf{KC}_k^d(Q,G)$ is not empty.

Comparing $G^{\underline{d}}$ and $\mathsf{KC}_k^d(Q,G)$, if $G^{\underline{d}}$ is a k-core, and each node can reach all the keywords within d. Then we have $G^{\underline{d}} \subset \mathsf{KC}_k^d(Q,G)$. At the same time, according to the previous discussion, it has $\mathsf{KC}_k^d(Q,G) \subset G^{\underline{d}}$. Thus, it can be derived that $\mathsf{KC}_k^d(Q,G) = G^{\underline{d}}$. Furthermore, if u can reach the keyword w within d in $G^{\underline{d}}$ for $u \in V^{\underline{d}}$, then there exists a path p from u to w, on the condition that, $\mathsf{cd}^{\mathsf{L}}(v) \leq d$ for each node $v \in p$ and $\mathsf{len}(p) \leq d$. Thus we have $\mathsf{ksd}(v) \leq d$. Based on these observations, we have the following lemma.

Lemma 5.1: *Given a graph G and a query Q, assume that there exists an integer d such that $G^{\underline{d}}(V^{\underline{d}},E^{\underline{d}})$ is a k-core. If for each node $u \in V^{\underline{d}}$, it has $\mathsf{skd}(u,Q,G^{\underline{d}}) \leq d$. Then $\mathsf{KC}_k^d(Q,G) = G^{\underline{d}}$.* □

According to Lemma 5.1, if $\mathsf{skd}(u) \leq d$ for each node $u \in V^{\underline{d}}$, $G^{\underline{d}}$ is $\mathsf{KC}_k^d(Q,G)$. Otherwise, there exists at least one node $v \in V^{\underline{d}}$ such that $\mathsf{skd}(v) > d$. In this scenario, $\mathsf{cd}^{\mathsf{L}}(v)$ can be updated as $\mathsf{skd}(v)$ according to Theorem 5.3. In the following, we discuss how to compute $\mathsf{skd}(u)$ for $G^{\underline{d}}$ with different d values. A straightforward solution to compute $\mathsf{skd}(u)$ is to enumerate d from 0 to $\mathsf{KwdC}(Q,G)$, and in each $G^{\underline{d}}$, we check whether u can reach all the keywords within d. However, considering $\mathsf{skd}(u,\mathsf{W_Q},G)$ and $\mathsf{skd}(u,\mathsf{W_Q},G^{\underline{d}})$ with different d values, the structure-keyword measure has the following properties:

Property 2: $\mathsf{skd}(u,\mathsf{W_Q},G^{\underline{d}}) = \mathsf{skd}(u,\mathsf{W_Q},G)$ if $d \geq \mathsf{skd}(u,\mathsf{W_Q},G)$.

Property 3: $\mathsf{skd}(u,\mathsf{W_Q},G^{\underline{d}}) \geq \mathsf{skd}(u,\mathsf{W_Q},G)$ if $d < \mathsf{skd}(u,\mathsf{W_Q},G)$.

These properties hold because for the number $d' = \mathsf{skd}(u,\mathsf{W_Q},G)$, there exists a path p from u to the keywords in G, such that $\mathsf{len}(p) \leq d'$ and $\mathsf{cd}^{\mathsf{L}}(v) \leq d'$ for $v \in p$. If $d \geq d'$, p will also appear in $G^{\underline{d}}$. For the case of $d < \mathsf{skd}(u,Q,G)$, according to Theorem 5.3, we have $d < \mathsf{skd}(u,\mathsf{W_Q},G) \leq \mathsf{cd}(u)$. Thus u cannot belong to $\mathsf{KC}_k^d(Q,G)$ with $d < \mathsf{skd}(u,\mathsf{W_Q},G)$. Considering these, we only need to consider the case of $d \geq \mathsf{skd}(u,\mathsf{W_Q},G)$, in which $\mathsf{skd}(u,\mathsf{W_Q},G^{\underline{d}}) = \mathsf{skd}(u,\mathsf{W_Q},G)$.

Based on Lemma 5.1, Property 2 and Property 3, we can compute the structure-keyword measure for every node u iteratively in G. $\mathsf{KC}_k^d(Q,G)$ will be found when for every node u in $G^{\underline{d}}$, the corresponding core-based distance is not larger than d. Otherwise, the node u with $\mathsf{skd}(u) > d$ will update $\mathsf{cd}^{\mathsf{L}}(u)$. Based on these, we design the advanced approach for the KCCS query as shown in the next subsection.

E. Advanced Approach for the KCCS Query

Considering the analysis about keyword-structure measure and structure-keyword measure, we design the advanced approach for the KCCS query as shown in Algorithm 2. The idea is to compute $\mathsf{cd}(u)$ for nodes, and d_{min} is the minimum value of d, such that $\mathsf{KC}_k^d(Q,G)$ contains all the keywords in $\mathsf{W_Q}$. Given a KCCS query $Q(\mathsf{W_Q},k)$ for the graph G, if the k-core of G does not contain all the keywords, there is no qualified

Algorithm 2 Advanced Approach for the KCCS Query

Input: A Graph G, a KCCS query $Q(\mathsf{W_Q}, k)$;
Output: The keyword-centric community of G w.r.t. Q;

1: $G \leftarrow$ *CoreComp*(G, k);
2: return G if G does not contain all $\mathsf{W_Q}$;
3: $\mathsf{cd}^{\mathsf{L}}(u) \leftarrow \mathsf{kdist}(u, \mathsf{W_Q}, G), \mathsf{skd}(u) \leftarrow 0$ for $u \in V(G)$;
4: $d \leftarrow \mathsf{KwdC}(Q, G)$, $\mathsf{R} \leftarrow \emptyset$;
5: **while** $\mathsf{cd}^{\mathsf{L}}(u)$ has been updated for $u \in V^{\underline{d}}$ **do**
6: **for** d' from d to 0 **do**
7: $\mathsf{R} \leftarrow$ *CoreMaintain*$(G^{\underline{d'}}, k, d')$;
8: $\mathsf{cd}^{\mathsf{L}}(v) \leftarrow \max(\mathsf{cd}^{\mathsf{L}}(u), d')$ for $v \in \mathsf{R}$;
9: *ComputeD*(G, w_i) for each $w_i \in W_Q$;
10: **for** d' from 0 to d **do**
11: **if** $\mathsf{skd}(u, \mathsf{W_Q}, G^{\underline{d'}}) \leq d'$ for each $u \in G^{\underline{d'}}$ **then**
12: $d \leftarrow d'$;
13: $\mathsf{cd}^{\mathsf{L}}(u) \leftarrow \max(\mathsf{skd}(u, \mathsf{W_Q}, G), \mathsf{cd}^{\mathsf{L}}(u))$ for $u \in V(G)$;
14: $d_{min} \leftarrow d$ if $\mathsf{W_Q} \in \mathsf{KC}_k^d(Q, G)$ and $\mathsf{W_Q} \notin \mathsf{KC}_k^{d'}(Q, G)$ for $d' < d$;
15: return $G^{\underline{d_{min}}}$;

keyword-centric community, and the k-core is returned (lines 1-2). After that, $\mathsf{cd}^{\mathsf{L}}(u)$ is set to $\mathsf{kdist}(u, \mathsf{W_Q}, G)$ (line 3). Initially, d is set to $\mathsf{KwdC}(Q, G)$, and R is used to keep the removed nodes due to the structural cohesiveness condition, and is set to empty (line 4). During the while loop, if $\mathsf{cd}^{\mathsf{L}}(u)$ for the node u in $G^{\underline{d}}$ will not be updated, then $\mathsf{KC}_k^d(Q, G)$ will be found as $G^{\underline{d}}$ (line 5). For each value d' from d to 0, the node set R is computed (line 7). Note that we make a small modification to the *CoreComp* procedure for the *CoreMaintain* procedure. In *CoreMaintain*$(G^{d'}, k, d')$, the node u is removed from $G^{d'}$ if $\mathsf{cd}^{\mathsf{L}}(u) \geq d'$, and the k-core of the remaining graph is G_1. *CoreMaintain*$(G^{d'}, k, d')$ returns R as $V(G^{d'}) - V(G_1)$. Due to the space limit, we do not show the details of *CoreMaintain* procedure. For each node $v \in R$, $\mathsf{cd}^{\mathsf{L}}(v)$ is updated as $\max(\mathsf{cd}^{\mathsf{L}}(u), \mathsf{skd}(u))$ (line 8). As $\mathsf{cd}^{\mathsf{L}}(u)$ has been updated, $\mathsf{ksd}(u, \mathsf{W_Q}, G)$ is computed in the *ComputeD* procedure, which is shown in Algorithm 3. If there exists d' such that $\mathsf{ksd}(u, \mathsf{W_Q}, G) \leq d'$ for each $u \in G^{\underline{d'}}$, then we have found $\mathsf{KC}_k^{d'}(Q, G)$, and we can only compute $\mathsf{KC}_k^d(Q, G)$ with $d < d'$ (lines 10-12). Otherwise, the $\mathsf{cd}^{\mathsf{L}}(u)$ will be updated as $\mathsf{ksd}(u)$ (line 13). At last, d_{min} is the minimum of d such that all the keywords $\mathsf{W_Q}$ appear in $\mathsf{KC}_k^d(Q, G)$ (line 14) and $G^{\underline{d_{min}}}$ will be returned as the keyword-centric community.

As shown in Algorithm 2, $\mathsf{skd}(u, w, G)$ is computed for each $w \in \mathsf{W_Q}$ and $u \in V(G)$. In the *ComputeD* procedure shown in Algorithm 3, $\mathsf{skd}(u)$ is computed in ascending order of the cd^{L} value. At first, $N(i)$ keeps the node u if the $\mathsf{cd}^{\mathsf{L}}(u) = i$ (line 1). $\mathsf{dis}(v)$ is the current distance between the node v and the keyword w. It is set as 0 initially for the nodes in $V(w)$ and $|V(G)| + 1$ otherwise. Then we enumerate different values from 0 to $|V(G)|$, to check whether $\mathsf{skd}(u, w, G)$ is i. At first, nodes in $N(i)$ are pushed into a priority queue as PQ (line 4). The top node u in PQ should have the smallest distance on the condition that all the nodes on the paths from u to w have their cd^{L} values not larger than i. Thus, $\mathsf{skd}(u, w, G)$ is set as i (line 6). Next, $\mathsf{dis}(v)$ will be computed for the neighbours of u that have not been searched.

Algorithm 3 ComputeD

Input: A Graph G, a keyword w;
Output: The $\mathsf{skd}(u, w, G)$ for $u \in V(G)$;

1: $N(i) \leftarrow u$ if $\mathsf{cd}^{\mathsf{L}}(u) = i$ for $u \in V(w)$;
2: $\mathsf{dis}(v) \leftarrow 0$ for $v \in V(w)$, $\mathsf{dis}(v) \leftarrow |V(G)| + 1$ for $v \notin V(w)$;
3: **for** i from 0 to $|V(G)|$ **do**
4: initialize a priority queue PQ based on dis and push the unsearched node $u \in N(i)$ into PQ if $\mathsf{cd}^{\mathsf{L}}(u) \leq i$;
5: **while** PQ is no empty **do**
6: $u \leftarrow \mathrm{Pop}(Q)$, $\mathsf{skd}(u, w, G) \leftarrow i$;
7: **for** each node $v \in \mathsf{nbr}(u)$ that has not been searched **do**
8: $\mathsf{dis}(v) \leftarrow \min(\mathsf{dis}(v), \mathsf{dis}(u) + 1)$;
9: **if** $\mathsf{dis}(v) \leq i$ **then**
10: push v into PQ;
11: **else**
12: push v into $N(i + 1)$;
13: return $\mathsf{skd}(u)$ for each node $u \in V(G)$;

TABLE II
ILLUSTRATION FOR ALGORITHM 2

$V(G)$	cd^{L}-initially	cd^{L}-1st	skd-1st
v_1	2	3	3
v_2	3	3	3
v_3	2	2	3
v_4	2	2	2
v_5	1	1	1
v_6	1	1	1
v_7	1	1	1
v_8	1	1	1
v_9	1	1	1
v_{10}	0	1	1
v_{11}	2	2	3
v_{12}	2	2	2

If $\mathsf{dis}(v) \leq i$, v is pushed into PQ (lines 9-10). Otherwise, it indicates that v can reach w within $i+1$ on the condition that the cd^{L} value for the nodes on the paths are not larger than i. In this scenario, v is inserted into $N(i+1)$ (line 12). At last, $\mathsf{ksd}(u)$ will be returned for $u \in V(G)$ (line 13).

Example 5.2: Consider the graph G shown in Fig. 3(a) and the query $Q = \{\{w_1, w_2\}, 3\}$. The $\mathsf{cd}^{\mathsf{L}}(u)$ value for each node u is shown in Table II. Initially, $\mathsf{cd}^{\mathsf{L}}(u)$ is set as $\mathsf{kdist}(u, \mathsf{W_Q}, G)$. For all nodes in Fig. 3(a), $\mathsf{kdist}(v_2)$ is the largest, and it equals $\mathsf{KwdC}(Q, G)$ as 3. In the first iteration, v_1, v_2 are inserted into R when the procedure *CoreMaintain*$(G^3, 3, 3)$ is called, because $\deg(v_1) = 2$ after removing v_2. Thus $\mathsf{cd}^{\mathsf{L}}(v_2)$ is updated as 3. Similarly, when *CoreMaintain*$(G^3, 3, 1)$ is called, it has $v_{10} \in R$, and then $\mathsf{cd}^{\mathsf{L}}(v_{10})$ is updated as 1.

In the *ComputeD* procedure, we use the keyword w_1 as an example. Initially, we have $N(1) = \{v_8, v_{10}\}$, $N(2) = \emptyset$, $N(3) = \{v_1, v_2\}$. When $i = 1$ as shown in line 3, the nodes in $N(1)$ will be pushed into PQ. When dealing with v_{10} from PQ, the dis values of v_5, v_6, v_7, v_9 will be updated as 1, and these nodes are inserted into PQ. Such a procedure carries on. When dealing with the node v_1, which is the neighbor of v_5, $\mathsf{dis}(v_1)$ is updated as 2. Thus, v_1 will be pushed into $N(2)$. After processing $N(1)$, $N(2)$ contains the nodes v_3, v_4, v_{12}. Similarly, for v_{11}, $\mathsf{dis}(v_{11})$ is set to 3, and it is inserted into $N(3)$. Thus $\mathsf{skd}(v_{11}) = 3$. The skd values for the nodes are shown in the skd-1st column in the table.

After computing the skd values for the nodes, consider d' as 1 as shown in line 10 in Algorithm 2. It can be found that, for the nodes $v_5, v_6, v_7, v_8, v_9, v_{10}$, their cd^{L} values are not larger than 1 and have not been updated. Then according to Algorithm 2, $\mathsf{skd}(v_i, Q, G^{\underline{1}}) \leq 1$ for $v_i \in V^{\underline{1}}$. Thus, d is set as 1 directly. In this scenario, the graph that needs to be considered is shown in Fig. 3(d). Then, it can be checked that the graph in Fig. 3(d) is the result of this KCCS query. □

F. Complexity Analysis

Given a KCCS query $Q(\mathsf{W_Q}, k)$ and the graph G, assume that $\mathsf{KC}_k^{d_{min}}(Q, G)$ is the keyword-centric community. As shown in Algorithm 1, the basic approach is to iteratively compute $\mathsf{kdist}(u, Q, G)$ for each node u in descending order of $\mathsf{KwdC}(Q, G)$ value. However, removing the node u with the largest kdist value does not always reduce $\mathsf{KwdC}(Q, G)$. It is possible that, after removing some nodes, the largest distance among all the shortest paths increases. Assume that it needs to scan the graph t_d times to achieve the graph G', with $\mathsf{KwdC}(Q, G') = d$. Then given a graph G and a query Q, the number of iterations in Algorithm 1 would be $\sum_{i=d_{min}}^{\mathsf{KwdC}(Q,G)} t_i$.

In contrast to the basic approach, Algorithm 2 computes $\mathsf{skd}(u)$, and updates the pruning bound of $\mathsf{cd}(u)$, for all the nodes in one iteration. In this scenario, the number of iterations needed to compute KC_k^d would be $\max(t_i)$ for $d_{min} \leq i \leq \mathsf{KwdC}(Q, G)$. Comparing the cost of the advanced approach with the basic approach, the cost is reduced from $(\sum_{i=d_{min}}^{\mathsf{KwdC}(Q,G)} t_i) \times (n + m)$ to $\max(t_i) \times (n + m)$ for $d_{min} \leq i \leq \mathsf{KwdC}(Q, G)$.

Also, in the advanced approach, the graph $G^{\underline{d_{min}}}$ in the first iteration would include all the nodes with $\mathsf{kdist}(u) \leq d_{min}$ for $u \in V(G)$. In each of the following iterations, at least one node $v \in V(G^{\underline{d_{min}}}) - V(\mathsf{KC}_k^{d_{min}})$ would have $\mathsf{skd}(v) > d_{min}$. The reason is that, if there is no such node v with $\mathsf{skd}(v) \leq d_{min}$ in the previous iteration and $\mathsf{skd}(v) > d_{min}$ in the following iteration, according to Lemma 5.1, it can be derived that $\mathsf{KC}_k^{d_{min}} \subset G^{\underline{d_{min}}}$. On the other hand, for each node $u \in G^{\underline{d_{min}}}$, the keyword distance is not larger than d_{min}, and $G^{\underline{d_{min}}}$ is a k-core. In this scenario, $G^{\underline{d_{min}}} \subset \mathsf{KC}_k^{d_{min}}$. Thus $G^{\underline{d_{min}}}$ is the $\mathsf{KC}_k^{d_{min}}$. In summary, in each iteration, at least one node in $V(G^{\underline{d_{min}}}) - V(\mathsf{KC}_k^{d_{min}})$ is removed.

VI. Optimizations

In this section, we introduce optimizations of our approaches, which rely on *keyword set reduction* and *indexing*.

A. Keyword Set Reduction

According to Algorithm 2, given a query $Q(\mathsf{W_Q}, k)$, it needs to call the *ComputeD* procedure $|\mathsf{W_Q}|$ times to compute $\mathsf{skd}(u, w, G)$ for each $w \in \mathsf{W_Q}$. Therefore, the cost of the advanced approach is proportional to the number of keywords in Q. Thus, reducing the number of keywords in the query improves the efficiency of the algorithm. Our optimization of keyword set reduction is based on the observation that for two keywords w, w', if $V(w) \subset V(w')$, then it can be derived that $\mathsf{kdist}(u, w, G) \geq \mathsf{kdist}(u, w', G)$ for each node $u \in V(G)$. The reason is that each shortest path from the node u to w is also a path from u to w'. Furthermore, according to the definition of $\mathsf{skd}(u)$, it can be derived that, $\mathsf{skd}(u, w', G) \geq \mathsf{skd}(u, w, G)$. As for the node u and a KCCS query $Q(\mathsf{W_Q}, k)$, $\mathsf{skd}(u, \mathsf{W_Q}, G) = \max(\mathsf{skd}(u, w_i, G))$ for $w_i \in \mathsf{W_Q}$, there is no need to consider w_i if there exists a keyword $w_j \in \mathsf{W_Q}$ and $V(w_j) \subset V(w_i)$.

B. Indexing

For the KCCS query, we use three different kinds of indexes. First of all, for the keywords in the graph, we use the inverted index. That is, given a graph G, for each label w in (G), we construct a linked list as $InvertI(w) = \{u_1, u_2, \cdots, u_k\}$, if w appears in the label of $u_i \in V(G)$. Secondly, we use the core-index for each node u. That is, for each node u in the k-core of G, we keep the core number as $\mathsf{C}(u) = k$. Thus the nodes in the k-core can be computed by accessing every node u with $\mathsf{C}(u) \geq k$. We also use $CoreI(k)$ to keep the nodes, whose core number is k. As these indexes are commonly used, we omit their details here.

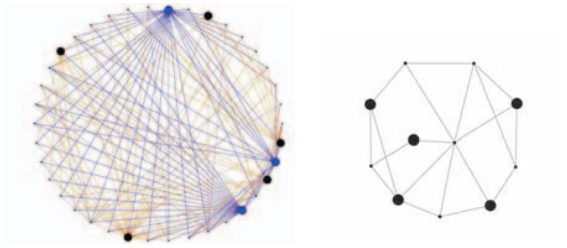
In addition, we construct a core-based inverted index. That is, given a graph $G(V, E, \mathcal{L})$, for each label w in G, we use $CI(w, k)$ to keep the set of nodes such that, for each node $u \in CI(w, k)$, w appears in $\mathcal{L}(u)$ and $\mathsf{C}(u) = k$. Given a keyword w and an integer k, all the nodes with the keyword w in the k-core can be accessed as the union of the nodes in $CI(w, k')$ with $k' \geq k$.

Example 6.1: Consider the graph shown in Fig. 2. The whole graph is a 2-core and the remaining graph except v_{13} is a 3-core. So $CoreI(2) = \{v_{13}\}$ and $CoreI(3) = \{v_1, v_2, \cdots, v_{12}\}$. Also, it can be found that $CI(3, w_1) = \{v_1, v_2, v_8, v_{10}\}$ because v_1, v_2, v_8, v_{10} contain the keyword w_1 and belong to a 3-core. Similarly, we have $CI(2, w_2) = \{v_{13}\}$, $CI(3, w_2) = \{v_7, v_{10}\}$, $CI(3, w_3) = \{v_2, v_7, v_{10}\}$. Given the KCCS query Q with $k = 3$ and $\mathsf{W_Q} = \{w_1, w_2, w_3\}$, it can be derived that $V(w_2) = \{v_7, v_{10}\}$ which contains all the nodes with the keyword w_2 in the 3-core. Similarly, $V(w_3) = \{v_2, v_7, v_{10}\}$. Thus, $V(w_2) \subset V(w_3)$ and w_3 does not need to be considered. Overall, v_{13} is removed from G and w_3 is removed from Q. □

VII. Performance Evaluation

In this section, we show the experimental results by comparing the basic approach shown in Algorithm 1, our advanced approach shown in Algorithm 2, and the advanced approach with optimizations. We show the results of our approaches in two aspects. The first concerns the effectiveness of the KCCS queries for finding the communities. For the effectiveness, we show that the results of the query containing the cohesive structure of the query keywords. The other concerns the efficiency of the comparison the basic and the advanced approaches.

Datasets: We compare our approaches using two real datasets, DBPedia and DBLP. The DBPedia knowledge graph is extracted from Wiki and contains 4.56 million nodes with 35.17



(a) KCCS in Q_1 (b) r-clique in Q_1

Fig. 5. The KCCS and r-clique found in DBLP citation network

million edges. In the DBLP datasets, there are four tables: papers, conferences, authors and citations. Each tuple in these tables is modeled as one node in the graph with the publication information as the labels and the edges between them are based on the foreign key reference. The corresponding graph has 6.07 million nodes with 9.77 million edges.

A. Effectiveness Testing

In order to demonstrate the effectiveness of the KCCS queries, we show and analyze 3 representative queries.

Q_1: Given the KCCS query as $Q = (\{$"protein", "interaction", "network", "topology", "algorithm"$\}, 8)$, we aim to find the interdisciplinary papers working on the research of the topological structure analytic algorithms and the protein-protein interactions in the DBLP citation network. As the traditional keyword search methods also generate the results with keywords, we compare our model with the representative keyword search semantics r-clique [12]. The r-clique method finds a group of nodes that cover all the query keywords and the distance between each two nodes is no greater than r. We apply r-clique to the DBLP citation network using the same set of the keywords in Q. We set the parameter $r = 2$, which achieves the same keyword distance as our KCCS method. Due to the expensive candidate enumerations of all the r-cliques by an exact method (not finished within 48 hours), we treated all output r-cliques in 48 hours as candidates, and computed the answer with the largest density. The results are shown in Figure 5. The following observations are made:

- **Community Cohesiveness**. Our KCCS method finds a community H_c with 146 nodes, 1,181 edges, and a graph density of 8.09 (for illustration, part of H_c is shown in Figure 5(a)). The keyword closeness of H_c is $\mathsf{KwdC}(\mathsf{W_Q}, H_c) = 2$, which indicates that each node can reach any keyword within 2 hops. In contrast, the result obtained by the r-clique method is shown in Figure 5(b), where the graph H_r has 11 nodes, 19 edges, and a graph density of 1.73. Clearly, compared with H_r, H_c has a more cohesive structure. The nodes in H_c, which are densely connected, naturally form a community, whereas the nodes in H_r are very loosely connected.
- **Relevance to Query Keywords**. In Figure 5(a), the large dark nodes represent the papers containing some of the query keywords, such as "Nemofinder: Dissecting Genome-wide Protein-protein Interactions with Meso-scale Network Motifs". The blue nodes represent the papers not containing any query keyword in the title. But, indeed these papers are a set of representative papers working on "topology graph analysis algorithms" for "protein-protein-interaction network". One example of the blue nodes is "Gspan: Graph-based Substructure Pattern Mining", which is an important work for frequent subgraph mining in graph databases for biological and chemical applications. This paper is highly cited with a very high of degree in Figure 5(a). This suggests that our KCCS model is capable to find a group of papers working on the same topic such that they are highly cited in a densely connected structure and all of them are close to the query keywords. In contrast, in Figure 5(b), the large dark nodes represent the papers containing some of the query keywords, such as "A Comparative Study on Network Motif Discovery Algorithms" and "Frequent Subgraph Mining on a Single Large Graph using Sampling Techniques". These papers cover the query keywords independently and may be working on totally different topics.
- **Efficiency**. The processing time of the KCCS query is 69.9s, which is much faster than the r-clique keyword search method.

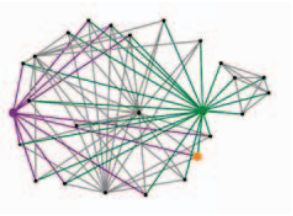


Fig. 6. The KCCS found in DBLP data set for Q_2

In addition, we have implemented another community model based on keyword coverage. The keyword-coverage community model adopts the same k-core structure as KCCS and requires to cover all the query keywords. It differs from KCCS in that the keyword distance is not considered. The resulting subgraph contains more than 227,000 nodes, and it is difficult to make sense out of the discovered "community". The maximum distance between the nodes and the query keywords is 6, which also suggests a weak keyword closeness.

Overall, our KCCS method is more efficient and can find a more interesting community highly related to the query keywords.

Q_2: Given that the KCCS query $Q = (\{$"privacy", "preserving", "publish"$\}, 6)$ on the DBLP dataset, the query aims to find the works on publishing data with privacy preserving guarantees. We aim to find the subgraph in which, the authors who have published at least six related papers or the papers have been cited at least six times. In the result graph of Q, denoted as G, we have $\mathsf{KwdC}(Q, G) = 2$. In other words, there is a 6-core graph if the distance bound between nodes and keywords is 2. One connected component of the 6-core is shown in Fig. 6. The green node represents the researcher "Ke Wang" and the purple node represents the researcher "Jian

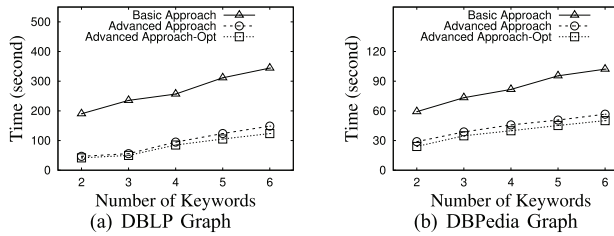


Fig. 7. Performance varying the number of keywords

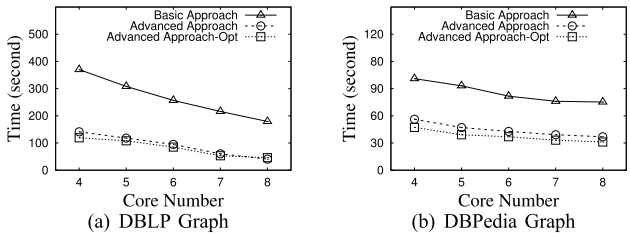


Fig. 8. Performance varying the core number

Pei". They have worked extensively on privacy preserving data mining. One example is represented as the node colored orange, which represents the paper "Minimality attack in privacy preserving data publishing". This paper has been cited 296 times.

Q_3: Assume that the KCCS query is $Q = (\{$"Stephen Curry", "Basketball", "Players"$\}, 4)$ on the DBPedia dataset. The query aims to find information about the famous basketball player named "Stephen Curry". The result G has $\mathsf{KwdC}(Q, G) = 2$. For the nodes in the community, most nodes have the labels as "NBA" and "Lebron James". "NBA" is the American basketball association and "Lebron James" is another famous basketball star who has collaborated multiple times with "Stephen Curry" in Olympic teams.

B. Efficiency Testing

In this section, we show the efficiency of our approaches. We compare between the basic approach (Algorithm 1), the advanced approach (Algorithm 2), and the advanced approach with the optimizations. For the inverted index, the core-index and the core-based inverted index introduced in Section VI, the construction time cost for the DBLP dataset is 5.52s, 57.80s, 64.57s, respectively, and for the DBPedia dataset, the time is 6.73s, 17.30s, 17.61s, respectively. We test the efficiency by varying the number of keywords and the core number in the query. To test scalability, we vary the graph data size.

Query Generation: We randomly generate a total of 100 queries according to DBLP's and DBPedia's vocabularies. The number of the keywords in $\mathsf{W_Q}$ of each Q ranges from 2 to 6. The criterion of selecting the keywords is as follows. We randomly select the first keyword w_1 from the dataset's label set. After that, we find all the nodes $V(w_1)$ that contain the keyword w_1 and also mark all the nodes u if it is within 2-hop from $v \in V(w_1)$. Then we select other keywords randomly from all the keywords appearing in the labels of the nodes marked. We report the average execution time of each data set for different approaches.

Varying the Number of Keywords: We vary the total number of keywords in each query from 2 to 6. The time costs of executing the KCCS queries in DBLP and DBPedia are shown in Fig. 7(a) and Fig. 7(b) respectively.

For the KCCS query processing, as the figures show, the execution time for each query increases as the number of keywords increases. The reason is that, with more keywords, the number of nodes containing the keywords also increases. Thus, more iterations are needed to find the shortest distance between each node and each keyword. Also, during the query processing, the maximum shortest distance among all the keywords is computed. With more keywords, computing $\mathsf{KwdC}(u, Q, G)$ for each node u costs more time.

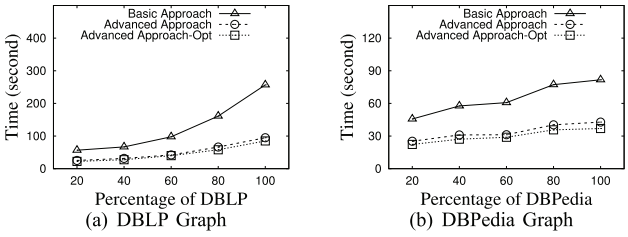
Our advanced approaches always perform much better than the basic one. The reason is that, after removing the nodes with the largest $\mathsf{kdist}(u, \mathsf{W_Q})$, the basic approach needs to update the shortest distance for each keyword. With more keywords in the query, the cost increases significantly. Our advanced approaches are based on the $\mathsf{skd}(u)$ and $\mathsf{ksd}(u)$. By comparing $\mathsf{skd}(u)$ and $\mathsf{cd}^{\mathsf{L}}(u)$, $\mathsf{KC}_k^d(Q, G)$ can be found directly. Thus, the advanced approaches need fewer iterations to find the results of the query, because more nodes can be removed earlier. With fewer number of iterations and lower cost in each iteration, the advanced approaches perform 3-5 times faster than the basic approach.

The approach with optimizations and indexes offers the best performance. Also, with larger $|\mathsf{W_Q}|$ in Q, the benefits of using optimizations and indexes become larger. The reason is that, during the processing, the existence of all the keywords in the k-core must be identified, because some nodes are removed. The index saves on time cost in this operation. Also, with more keywords, there is a higher possibility that some keywords can be removed from Q according to the discussion in Section VI-A, and the performance can be improved.

Varying the Core Number: We vary the core number for each query from 2 to 6. The time costs of executing the queries in DBLP and DBPedia are shown in Fig. 8(a) and Fig. 8(b) respectively.

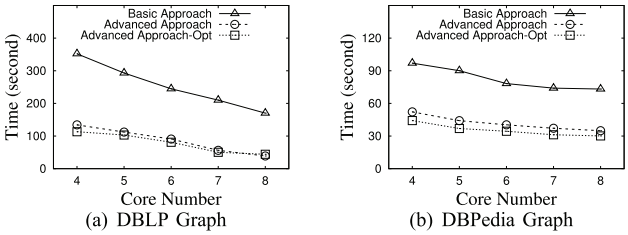
With a larger core number, the time cost of all the approaches for the KCCS queries decreases. This is because given the KCCS query, the corresponding k-core in the datasets is smaller with a larger k. Thus the remaining graph for consideration becomes smaller. In this case, the cost of both the basic and advanced approaches is reduced. The reason is that, a k-core with a larger k indicates that there are a fewer number of nodes containing the keywords in Q. Thus, $\mathsf{KwdC}(Q, G)$ becomes larger and fewer iterations are needed during the computation.

Compared the performance between DBLP and DBPedia, even if DBPedia has larger nodes and edges, the time cost for the query in DBPedia is much smaller than that in DBLP. The reason is as follows. DBPedia is a knowledge graph and for the subgraph related to one area, the shortest distance between



(a) DBLP Graph (b) DBPedia Graph

Fig. 9. Performance varying the graph size



(a) DBLP Graph (b) DBPedia Graph

Fig. 10. Performance of all the queries

nodes is much smaller. Thus, given a KCCS query $Q(\mathsf{W_Q}, k)$ and assuming that G_k is the k-core of the initial graph, the $\mathsf{KwdC}(Q, G_k)$ in DBPedia is much smaller than that in DBLP. Thus, DBPedia needs fewer iterations to find the result with the minimum KwdC value, even if it has about 25 million more edges than DBLP.

Varying the Graph Size: In order to show the scalability of our approaches, we vary the graph size in this experiment. We randomly select a subset of edges from the whole graph and the number of selected edges ranges from 20% to 100% for DBLP and DBPedia. The results for DBLP and DBPedia are shown in Fig. 9(a) and Fig. 9(b) respectively.

With a larger graph size, the time cost of all the approaches increases. The increasing rate of our advanced approaches is much slower than the basic approach. As with a larger edge set, the initial k-core contains more nodes and edges. Thus, computing the shortest distance and k-core in each iteration costs more in the basic approach. Instead of updating the shortest distance between nodes and keywords, our advanced approaches consider k-core computation and the shortest distance computation at the same time. Thus, for a node with smaller distance to the keyword, it is possible that due to the k-core condition, its $\mathsf{KwdC}(u, Q)$ is quite large. Our approaches can remove such nodes in earlier iterations and reduce the graph sizes significantly.

VIII. Conclusions

In this paper, we have studied the novel problem of *keyword-centric community search* over attributed graphs, to find a keyword-centric community with a densely connected structure while achieving the strongest keyword closeness w.r.t. the query. We propose efficient algorithms as well as the optimizations and indexes. Extensive experiments on large real networks demonstrate the superiority of our methods in terms of both effectiveness and efficiency.

IX. Acknowledgment

This work was supported by the grants from NSFC 61602395, NSFC 61702435, RGC 12232716, RGC 12258116, RGC 12200817,RGC 12200917, and RGC 12201518.

References

[1] W. Cui, Y. Xiao, H. Wang, Y. Lu, and W. Wang, "Online search of overlapping communities," in *SIGMOD*, 2013, pp. 277–288.

[2] M. Sozio and A. Gionis, "The community-search problem and how to plan a successful cocktail party," in *KDD*, 2010, pp. 939–948.

[3] N. Barbieri, F. Bonchi, E. Galimberti, and F. Gullo, "Efficient and effective community search," *Data Mining and Knowledge Discovery*, vol. 29, no. 5, pp. 1406–1433, 2015.

[4] W. Cui, Y. Xiao, H. Wang, and W. Wang, "Local search of communities in large graphs," in *SIGMOD*, 2014, pp. 991–1002.

[5] X. Huang, L. V. Lakshmanan, J. X. Yu, and H. Cheng, "Approximate closest community search in networks," *PVLDB*, vol. 9, no. 4, pp. 276–287, 2015.

[6] E. Akbas and P. Zhao, "Truss-based community search: a truss-equivalence based indexing approach," *Proceedings of the VLDB Endowment*, vol. 10, no. 11, pp. 1298–1309, 2017.

[7] Y. Wu, R. Jin, J. Li, and X. Zhang, "Robust local community detection: On free rider effect and its elimination," *PVLDB*, vol. 8, no. 7, 2015.

[8] Y. Fang, R. Cheng, S. Luo, and J. Hu, "Effective community search for large attributed graphs," *Proc. VLDB Endow.*, vol. 9, no. 12, pp. 1233–1244, Aug. 2016. [Online]. Available: http://dx.doi.org/10.14778/2994509.2994538

[9] X. Huang and L. V. S. Lakshmanan, "Attribute-driven community search," *Proc. VLDB Endow.*, vol. 10, no. 9, pp. 949–960, May 2017. [Online]. Available: https://doi.org/10.14778/3099622.3099626

[10] Z. Wang, Y. Yuan, G. Wang, H. Qin, and Y. Ma, "An effective method for community search in large directed attributed graphs," in *International Conference on Mobile Ad-Hoc and Sensor Networks*. Springer, 2017, pp. 237–251.

[11] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou, "Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data," in *Proc. of SIGMOD'08*, 2008.

[12] M. Kargar and A. An, "Keyword search in graphs: Finding r-cliques," *Proc. VLDB Endow.*, vol. 4, no. 10, pp. 681–692, Jul. 2011.

[13] L. Qin, J. X. Yu, L. Chang, and Y. Tao, "Querying communities in relational databases," in *Proc. of ICDE'09*, 2009.

[14] L. Qin, J. X. Yu, and L. Chang, "Keyword search in databases: the power of rdbms," in *Proc. of SIGMOD'09*, 2009.

[15] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, "Keyword searching and browsing in databases using banks," in *Proc. of ICDE'02*, 2002.

[16] H. He, H. Wang, J. Yang, and P. S. Yu, "Blinks: ranked keyword searches on graphs," in *Proc. of SIGMOD'07*, 2007.

[17] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin, "Finding top-k min-cost connected trees in databases," in *Proc. of ICDE'07*, 2007.

[18] V. Hristidis and Y. Papakonstantinou, "Discover: Keyword search in relational databases," in *Proc. of VLDB'02*, 2002.

[19] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar, "Bidirectional expansion for keyword search on graph databases," in *Proc. of VLDB'05*, 2005.

[20] F. Liu, C. T. Yu, W. Meng, and A. Chowdhury, "Effective keyword search in relational databases," in *Proc. of SIGMOD'06*, 2006.

[21] Y. Wu, S. Yang, M. Srivatsa, A. Iyengar, and X. Yan, "Summarizing answer graphs induced by keyword queries," *PVLDB*, vol. 6, no. 14, 2013.

[22] R.-H. Li, L. Qin, J. X. Yu, and R. Mao, "Influential community search in large networks," *PVLDB*, vol. 8, no. 5, 2015.

[23] X. Huang, L. V. Lakshmanan, and J. Xu, "Community search over big graphs: Models, algorithms, and opportunities," in *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*. IEEE, 2017, pp. 1451–1454.

[24] X. Huang, L. V. S. Lakshmanan, and J. Xu, *Community Search over Big Gaphs*. Synthesis Lectures on Data Management, Morgan & Claypool Publishers, 2019.

[25] V. Batagelj and M. Zaversnik, "An o(m) algorithm for cores decomposition of networks," *CoRR*, vol. cs.DS/0310049, 2003. [Online]. Available: http://arxiv.org/abs/cs.DS/0310049