Keyword-Centric Community Search over Large Heterogeneous Information Networks

Lianpeng Qiao[1](✉), Zhiwei Zhang[2], Ye Yuan[2], Chen Chen[2], and Guoren Wang[2]

[1] Northeastern University, Shenyang, China
qiaolp@stumail.neu.edu.cn
[2] Beijing Institute of Technology, Beijing, China
Yuan-ye@bit.edu.cn

Abstract. Community search in heterogeneous information networks (HINs) has attracted much attention in recent years and has been widely used for graph analysis works. However, existing community search studies over heterogeneous information networks ignore the importance of keywords and cannot be directly applied to the keyword-centric community search problem. To deal with these problems, we propose $k\mathcal{KP}$-core, which is defined based on a densely-connected subgraph with respect to the given keywords set. A $k\mathcal{KP}$-core is a maximal set of $\mathcal{P}$-connected vertices in which every vertex has at least one $\mathcal{KP}$-neighbor and k path instances. We further propose three algorithms to solve the keyword-centric community search problem based on $k\mathcal{KP}$-core. When searching for answers, the basic algorithm Basic-$k\mathcal{KP}$-core will enumerate all paths rather than only the path instances of the given meta-path $\mathcal{P}$. To improve efficiency, we design an advanced algorithm *Advk*$\mathcal{KP}$-core using a new method of traversing the search space based on trees to accelerate the searching procedure. For online queries, we optimize the approach with a new index to handle the online queries of community search over HINs. Extensive experiments on HINs are conducted to evaluate both the effectiveness and efficiency of our proposed methods.

Keywords: Keyword-centric · Community · Heterogeneous information networks

1 Introduction

Heterogeneous information networks (HINs) [11,17] are the networks involving multiple objects and multiple links denoting different types and relations, and has been widely used to model bibliographic networks, social media networks, and knowledge networks. Figure 1(a) depicts an HIN of the bibliographic network, which describes the relationships between different types of entities. In this network, vertices with labels A, P, V, and T represent authors, papers, venues, and time. It consists of four types of entities. The directed lines between the vertices denote their semantic relationships. For example, the links between a_1, a_2, p_1, v_1 indicate that the author a_1 and a_2 have written a paper p_1 accepted by CIKM.

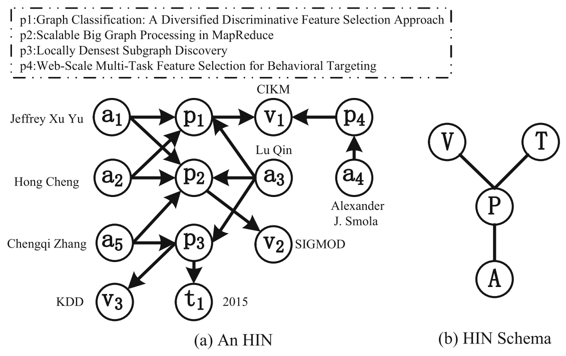

p1:Graph Classification: A Diversified Discriminative Feature Selection Approach
p2:Scalable Big Graph Processing in MapReduce
p3:Locally Densest Subgraph Discovery
p4:Web-Scale Multi-Task Feature Selection for Behavioral Targeting
CIKM
Jeffrey Xu Yu
Lu Qin
Hong Cheng
Alexander J. Smola
Chengqi Zhang
SIGMOD
KDD
2015
a1 a2 a3 a4 a5 p1 p2 p3 p4 v1 v2 v3 t1
V T P A
(a) An HIN
(b) HIN Schema

Fig. 1. An example HIN of DBLP network

On the other hand, many real-world networks have a significant community structure, in which vertices in a community are densely connected. Community search has essentially studies in graph analysis, and has attracted much attention in the literature. For HINs, there are also some works focusing on community search problem in them [11,27–29,33]. However, these studies focus on either structure density or keyword cohesiveness, and few of them consider all these at the same time.

Existing works on network community search can be classified into community detection [12,24,27–29,33] and community search [4,6,7]. Community detection algorithms aim to detect all communities for a graph, which means they are not designed for online query. Different from community detection, query-based community search has been studied [7,10,15], in which the vertices and keywords are given, and they aim to find the most cohesive subgrahs related the given vertices and keywords. The vertices and edges in HINs carry different semantic meanings. In this scenario, existing approaches for community search on homogeneous networks cannot solve the community search problem in HINs.

In this paper, we focus on searching communities concerning given keywords set in HINs, in which vertices are with a specific type (e.g., a community of authors in the bibliographical network, as shown in Fig. 1). For the keyword-centric community, we need to deal with three questions. (1) How to combine the keywords with the community? (2) How to measure the connectivity of two vertices of the same type? (3) How to measure the cohesiveness of a community?

For the first two questions, we adopt the meta-path concept [23] to connect two vertices since the vertices with the same type may not be connected directly in the HIN. For the third question, existing solutions adopt a minimum degree [7,9], k-truss [15], or k-clique [6,30] to measure the community cohesiveness.

The minimum degree is the most common metric to ensure every vertex is well engaged in the community. In this paper, we extend such metric for HINs.

For the problem of community search in HINs, the community returned by the queries should be the subgraph in which the distance between vertices is small. Also, the community need to be cohesive considering all the keywords given in the queries. Therefore, we propose a new model called $k\mathcal{KP}$-core in HINs. There are three requirements for a $k\mathcal{KP}$-core S: (1) every vertex v has at least k path instances of the given meta path $\mathcal{P}$ in S starting with v; (2) every vertex v has at least one instance contains the given set of keywords; (3) for the connected graph S, any two vertex u and v could be connected by a path p, and any two adjacent vertices in p should be connected by a path instance. Given an HIN $G(V, E)$, a query $q \in V$, keywords set K, and an integer k, our goal is to find a maximum $k\mathcal{KP}$-core containing q, in which all the vertices are with the same type of q and contain the keywords set K.

In summary, we make the following contributions.

- We propose a keyword-centric community model called $k\mathcal{KP}$-core and formulate the problem of the keyword-centric community search;
- We propose a baseline algorithm to search the community in HINs;
- We design a new method to traversal the search space based on trees to accelerate the community search algorithm, as shown in Algorithm 2 and Algorithm 3. We further propose the optimization for the approach as shown in Sect. 3.3.
- We conduct a series of experiments on real-world HINs to evaluate the effectiveness and efficiency of our algorithms.

The rest of this paper is organized as follows. In Sect. 2, we define the keyword-centric community search problem in HINs. In Sect. 3, we proposed several Algorithms to solve the problem. In Sect. 4, we conduct extensive experiments on real-world HINs to show the effectiveness and efficiency of our methods. We review the related work in Sect. 5 and conclude in Sect. 6.

2 Problem Definition

In this section, we introduce several definitions used in this paper. Furthermore, we define the problem of keyword-centric community search in HINs.

Definition 1 (HIN) [11,17]. *An HIN is a directed graph $G(V, E)$ with a vertex type mapping function $\psi : V \rightarrow \mathcal{A}$ and an edge type mapping function $\phi : E \rightarrow \mathcal{R}$, where each vertex $v \in V$ belongs to a vertex type $\psi(v) \in \mathcal{A}$, and each edge $e \in E$ belongs to an edge type (also called relation) $\phi(e) \in \mathcal{R}$.*

Definition 2 (HIN schema). *Given an HIN $G(V, E)$ with mappings $\psi : V \rightarrow \mathcal{A}$ and $\phi : E \rightarrow \mathcal{R}$, its schema G_s is an undirected graph defined over vertex types A and edge types (as relation) $\mathcal{R}$, i.e., $G_s(\mathcal{A}, \mathcal{R})$.*

Figure 1(b) is the schema of the HIN, in which the vertices labeled A, P, V, and T denote author, paper, venue, time. The schema describes all the edge types between all the vertex types that exist in HIN. In [11,17], HIN schema is defined on directed graph, our approach can also be applied to the HIN schema of directed graph.

In this paper, we try to find a community concerning given keywords set in an HIN, in which all the vertices have the same type, such type is called target type. Since the vertices in a community should be connected cohesively, we define the connection between vertices in an HIN using the symmetric meta-path $\mathcal{P}$, whose source object and target object are with the target type. To describe the cohesiveness between the vertices with the target type in a community, we extend the classic k-core as $k\mathcal{KP}$-core with a symmetric meta-path $\mathcal{P}$.

Definition 3 (Meta-path) [11]. *A meta-path $\mathcal{P}$ is a path defined on an HIN schema $G_s = (\mathcal{A}, \mathcal{R})$, and is denoted in the form $A_1 A_2 ... A_l$, where $A_i \in \mathcal{A}(1 \leq i \leq l)$ and $\phi(A_j, A_{j+1}) \in \mathcal{R}(1 \leq j \leq l-1)$.*

Definition 4 (Path instance). *Given an HIN $G = \{V, E\}$, a meta-path $\mathcal{P} = (A_1 A_2 ... A_l)$, the path $a_1 \rightarrow a_2 \rightarrow ... \rightarrow a_l$ between vertices a_1 and a_l is a path instance of $\mathcal{P}$, if it satisfies $\psi(a_i) = A_i(1 \leq i \leq l)$, and $\phi(a_j, a_{j+1}) = \phi(A_j, A_{j+1})(1 \leq j \leq l-1)$.*

According to Definition 4, we say that a vertex v is a $\mathcal{P}$-neighbor of vertex u if an instance of $\mathcal{P}$ can connect them. For example, in Fig. 1, $\mathcal{P} = (APVPA)$ is a meta-path of the given HIN, vertex a_4 is a $\mathcal{P}$-neighbor of vertex a_1, since an instance of $\mathcal{P} = (APVPA)$ can connect them. We say that two vertices u and v are $\mathcal{P}$-connected if a chain of vertices can connect them, such that every vertex is a $\mathcal{P}$-neighbor of its adjacent vertex in the chain. The definition of $\mathcal{P}$-connected and $\mathcal{P}$-neighbor denotes the connectivity between vertices in HIN and meta paths. Furthermore, we propose K-instance to denote the correlations between keywords in the nodes, which is introduced as follows.

Definition 5 ($\mathcal{K}$-instance). *Given an HIN $G(V, E)$, a meta-path $\mathcal{P}$ and a set of input keywords $K = \{k_1, k_2, ..., k_w\}$, a $\mathcal{K}$-instance of $\mathcal{P}$ is a path instance p whose labels set contains all the keywords in K.*

Example 1. Given an HIN as shown in Fig. 1 and $K = \{$"*Selection*", "*CIKM*"$\}$, the path $p = a_1 \rightarrow p_1 \rightarrow v_1 \rightarrow p_4 \rightarrow a_4$ is an $\mathcal{K}$-instance of $\mathcal{P} = (APVPA)$, since p_1 contains keyword "*Selection*" and v_1 contains keyword "*CIKM*".

Definition 6 ($\mathcal{KP}$-neighbor). *Given a meta-path $\mathcal{P}$ and a set of input keywords $K = \{k_1, k_2, ..., k_w\}$, we say that a vertex u is a $\mathcal{KP}$-neighbor of a vertex v, if they can be connected by a $\mathcal{K}$-instance of $\mathcal{P}$.*

Definition 7 ($deg(v, S)$). *Given an HIN, a set of vertices S and a meta-path $\mathcal{P}$, we define $deg(v, S)$ as the number of path instances of $\mathcal{P}$ between v and all the other vertices in $S \backslash \{v\}$.*

To ensure the cohesiveness of the keyword-centric community, existing works often use k-core to characterize the cohesiveness of a community. In this paper, we aim to find a community in an HIN containing a query vertex q, in which all the vertices have the same vertex type as $\psi(q)$. We use a symmetric meta-path $\mathcal{P}$ to connect vertices with the target type. Then we can extend the k-core model for HINs as follow.

Definition 8 ($k\mathcal{KP}$-core). *Given an HIN G, a keywords set K, a symmetric meta path $\mathcal{P}$ and an integer k, a $k\mathcal{KP}$-core is a maximal set S of $\mathcal{P}$-connected vertices, s.t. $\forall v \in S$, $deg(v, S) \geq k$ and v has at least one $\mathcal{KP}$-neighbor.*

Example 2. We use $degK(a, S)$ to present the number of $\mathcal{KP}$-neighbors of vertex a in $S \backslash \{a\}$. Consider the HIN $G(V, E)$ in Fig. 1, a meta-path $\mathcal{P} = (APVPA)$ and a keywords set $K = \{$"*Selection*", "*CIKM*"$\}$. Let $k = 3$. Then we can see that the induced subgraph of $\{a_1, a_2, a_3, a_4\}$ denoted as S is a $k\mathcal{KP}$-core. It has $deg(a_1, S) = 5$, $deg(a_2, S) = 5$, $deg(a_3, S) = 5$, $deg(a_4, S) = 3$ and $degK(a_1, S) = 3 > 1$, $degK(a_2, S) = 3 > 1$, $degK(a_3, S) = 3 > 1$, $degK(a_4, S) = 3 > 1$.

Based on Definition 8, we can find different types of communities. As shown in Example 2, we get a community of authors using the meta-path $\mathcal{P} = (APVPA)$, which represents a potential collaborative community of papers. Besides, we could get different communities of vertices with different types by using different meta-paths. Note that all the meta-paths we used in the rest of the paper are symmetric. Now we introduce the keyword-centric community search problem in HINs as follow:

Problem. Given an HIN G, a query vertex q, a keywords set K, a meta-path $\mathcal{P}$ and an integer k, the keyword-centric community we search in G is the corresponding $k\mathcal{KP}$-core containing q.

As shown in Example 2, let $q = a_1$, $K = \{$"*Selection*", "*CIKM*"$\}$, meta-path $\mathcal{P} = (APVPA)$ and $k = 3$. We could get the corresponding keyword-centric community $C(V, E)$, in which $V = \{a_1, a_2, a_3, a_4\}$ and the labels of the vertices include "Jeffrey Xu Yu", "Hong Cheng", "Lu Qin", and "Alexander J. Smola", since C is a maximum $k\mathcal{KP}$-core containing q, which means that there could be more collaborations between the authors in C. Note that Jeffrey Xu Yu and Alexander J. Smola have published papers with the same keyword "Selection" in the same venue "CIKM". According to Definition 8, we can know that the keyword-centric community satisfies the structural maximality and connectivity.

Theorem 1. *Given an HIN G, a query vertex q, keywords set K, a meta-path $\mathcal{P}$ and an integer k, the $k\mathcal{KP}$-core containing q is unique.*

Proof. Suppose that X and Y are two different $k\mathcal{KP}$-cores containing q, we could get that $X \cup Y$ could be a new $k\mathcal{KP}$-core. For each vertex $v \in Y$, we can get that p and v are $\mathcal{P}$-connected. Then we can get that all the vertices in X and Y are $\mathcal{P}$-connected. Since all the vertices in X and Y have k or more than k $\mathcal{P}$-neighbors, we can get that $X \cup Y$ is a new $k\mathcal{KP}$-core containing q, which is against the initial assumption.

3 Search Algorithm

We adopt $k\mathcal{KP}$-core as the model to search the keyword-centric communities in HINs. In this section, we present efficient solutions for the community search problem in HINs.

Algorithm 1: Basic-$k\mathcal{KP}$-core

```
Input: the HIN graph $G(V, E)$; query vertex $q$; the keywords set $K = \{k_1, k_2, ..., k_w\}$; the
       meta-path $\mathcal{P}$ and $k$
Output: the set of all $k\mathcal{KP} - core$.
1  collect the set $S$ of vertices with the same vertex type as $q$;
2  $deg(v, S) \leftarrow$ the number of all path instances of $\mathcal{P}$ starting with $v$ and end with $u$ for each
     $u \in S \backslash \{v\}$;
3  $\mathcal{KP}\_neighbor[v] \leftarrow \mathcal{KP}\_neighbor[v] \cup \{u\}$, if the path instance of $\mathcal{P}$ starting with $v$ and end
     with $u$ cover all the keywords in $K$, $\forall u \in S$;
4  foreach $v \in S$ do
5      if $deg(v, S) < k$ or $\mathcal{KP}\_neighbor[v] = \emptyset$ then
6          remove $v$ from $S$;
7          foreach $u \in S$ do
8              if $u$ is the $\mathcal{KP}$-neighbor of $v$ then remove $v$ from $\mathcal{KP}\_neighbor[u]$ ;
9              update $deg(u, S)$;
10 remove all the vertices which are not $\mathcal{P}$-connected with $q$ from $S$;
11 return $S$;
```

3.1 The Basic Algorithm

Based on the concept of $k\mathcal{KP}$-core, we present a basic algorithm, as shown in Algorithm 1 to find the maximum $k\mathcal{KP}$-core containing the query vertex q in an HIN. In general, it consists of four steps: (1) collect the set S of all vertices with target type (line 1); (2) for each vertex $v \in S$, enumerate all path instances of $\mathcal{P}$ starting with v and end with the nodes in $S\backslash\{v\}$, and find the set of $\mathcal{KP}$-neighbors of v in S. Incidentally, we use i-th($\mathcal{P}$) to present the i-th vertex type of $\mathcal{P}$ (lines 2–3); (3) remove the vertex v which has less than k path instances starting with v or has no $\mathcal{KP}$-neighbor from S and update the remaining vertices iteratively until there is no vertex in S can be removed (lines 4–9); (4) the remaining S is the keyword-centric community we are looking for (lines 10–11).

Example 3. Consider the HIN in Fig. 1 and the meta-path $\mathcal{P} = (APVPA)$. Let the query vertex be a_1, $k = 4$ and keywords $K = \{\text{“}Selection\text{”}, \text{“}CIKM\text{”}\}$. First, we compute the number of the path instances starting with the vertex in $S = \{a_1, a_2, a_3, a_4, a_5\}$ and the set of $\mathcal{KP}$-neighbors of the vertex in S. By enumerate all the path instances of meta path $\mathcal{P}$, we can get $deg(a_1, S) = 6$, $deg(a_2, S) = 6$, $deg(a_3, S) = 7$, $deg(a_4, S) = 3$, $deg(a_5, S) = 4$, $degK(a_1, S) = 3$, $degK(a_2, S) = 3$, $degK(a_3, S) = 3$, $degK(a_4, S) = 3$, $degK(a_5, S) = 0$. Since we get $deg(a_4, S) = 3 < 4$ and $degK(a_5, S) = 0 < 1$. We have to remove a_4 and a_5 from S. Because a_4 is the $\mathcal{KP}$-neighbor of $\forall v \in \{a_1, a_2, a_3\}$, we can update the $degK(v, S)$ $\forall v \in \{a_1, a_2, a_3\}$ which is 2, 2, 2 respectively. Since we remove a_4 and a_5 from S, we have to recompute the path instance starting with the vertex in $\{a_1, a_2, a_3\}$. Finally, we get $deg(a_1, S) = 4$, $deg(a_2, S) = 4$,

$deg(a_3, S) = 4$, $degK(a_1, S) = 2$, $degK(a_2, S) = 2$, $degK(a_3, S) = 2$ which means that $\{a_1, a_2, a_3\}$ is a keyword-centric community.

In Algorithm 1, we enumerate all the instances starting with each $v \in V$, and the complexity of this process can be bounded by $|S| * |V|^l$, where S is the set of vertices with the target type, and l is the length of the meta-path $\mathcal{P}$.

Algorithm 2: TraversalTreeBuild

```
Input: the HIN graph $G(V, E)$; the set $S$ of vertices with the target type; the keywords set
       $K = \{k_1, k_2, ..., k_w\}$; the meta-path $\mathcal{P}$; the parameter of length $h$
1  foreach $v \in S$ do
2      push the vertex $v$ into an empty queue $C$, $count(u, v) \leftarrow 1$ for each $u \in S$ if $(u, v) \in E$;
3      $i \leftarrow 1$;
4      while $i \le h$ AND $C \ne \emptyset$ do
5          $u \leftarrow C.pop()$;
6          foreach $z$ which have $(z, u) \in E$ and $\psi(z) = (i + 1) - th(\mathcal{P})$ do
7              $C.push(z)$;
8              $root[z] \leftarrow v$;
9              $count(z, v) \leftarrow count(z, v) + count(u, v)$;
10             insert $X \cup \{key[z]\}$ into $key(z, v)$ for each $X \in key(u, v)$;
11         $i++$;
```

Theorem 2. *Given an HIN $G(V, E)$, a keywords set K, a query vertex q and the meta-path $\mathcal{P}$. Let l be the length of $\mathcal{P}$. The complexity of Algorithm 1 is $O(|S| * n^l)$, where $n = |V|$.*

3.2 Advanced Algorithm

Algorithm 1 is very costly to enumerate all instances, starting with each vertex in S in step (2) and step (3). To speed up step (2) and step (3), we propose Algorithm 2 and Algorithm 3 using a new method of traversing the search space based on trees. Note that we use $K.has(v)$ to judge whether there is a keyword in K which is contained by v, $key[v]$ to express the keyword in K which is contained by v and $key(u, v)$ to express the set of keywords contained by the path from u to v in the trees. The general idea of the approach is to maintain the number of path instances and keywords set between u and v whose vertex types are the same as the target type and the middle vertex type of the given meta path $\mathcal{P}$ respectively. Based on these trees, we can save a lot of time when calculating the number of path instances of $\mathcal{P}$ and judging whether there is a K-instance between two nodes with target type.

The process of building the trees is shown in Algorithm 2. First, we push vertex v into queue C and initialize $count(u, v)$ (lines 1–2). Then, we iteratively push the vertices with the corresponding vertex type as $\mathcal{P}$ into C, until we get the number of the path instances and the group of the keywords sets between v and those vertices with the vertex type h-th$(\mathcal{P})$ (lines 3–11).

The next operation is shown in Algorithm 3. First of all, we collect the set S of all vertices with target type and get the set of all index trees rooted at the vertex in S (lines 1–3). Next, we move the vertices which are not $\mathcal{P}$-connected

Algorithm 3: Adv$k\mathcal{KP}$-core

```
Input: the HIN graph $G(V, E)$; query vertex $q$; the keywords set $K = \{k_1, k_2, ..., k_w\}$; the
       meta-path $\mathcal{P}$ and $k$
Output: the maximum set of $k\mathcal{KP} - core$ containing $q$.
1  collect the set $S$ of vertices with the same vertex type as $q$;
2  queue $\mathcal{Q} \leftarrow \emptyset$;
3  $TraversalTreeBuild(G, S, \mathcal{P}, \lceil \frac{sizeof(\mathcal{P})}{2} \rceil)$;
4  foreach $v \in S$ do
5      if $v$ and $q$ are not $\mathcal{P}$-connected then $S \leftarrow S \backslash \{v\}$; $\mathcal{Q}.push(v)$; ;
6  foreach $v \in S$ do
7      $deg(v, S) \leftarrow 0$; $k\_neighbor[v] \leftarrow \emptyset$;
8      foreach $u \in S \backslash \{v\}$ do
9          foreach leaf vertex $z$ of $v$ do
10             $deg(v, S) \leftarrow deg(v, S) + count(z, v) * count(z, u)$;
11             if $X \cup Y = K$ where $X \in key(z, v)$ and $Y \in key(z, u)$ then
12                 $k\_neighbor[v].push(u)$;
13     if $deg(v, S) < k$ OR $k\_neighbor[v] = \emptyset$ then $S \leftarrow S \backslash \{v\}$, $\mathcal{Q}.push(v)$; ;
14 foreach $u \in \mathcal{Q}$ do
15     foreach $v \in S$ do
16         $C \leftarrow$ the commen leaf nodes between $u$ and $v$;
17         foreach $z \in C$ do
18             $deg(v, S) \leftarrow deg(v, S) - count(z, v) * count(z, u)$;
19             $k\_neighbor[v].remove(z)$;
20         if $deg(v, S) < k$ OR $k\_neighbor[v] = \emptyset$ OR $v$ and $q$ are not $\mathcal{P}$-connected then
             $S \leftarrow S \backslash \{v\}$, $\mathcal{Q}.push(v)$;
21 return $S$;
```

with q from S to $\mathcal{Q}$ (lines 4–5). Then we get $deg(v, S)$ the number of the path instances starting with the v in S and $k\_neighbor[v]$ the set of $\mathcal{KP}$-neighbors of v. If we have $deg(v, S) < k$ or $k\_neighbor[v] = \emptyset$, we can move all vertices v from S to $\mathcal{Q}$ (lines 6–13). Afterwards, we remove all vertex $u \in \mathcal{Q}$ from S and update $deg(v, S)$ and $k\_neighbor[v]$ for all $v \in S \backslash \mathcal{Q}$ iteratively until there is no vertex can be removed from S (lines 14–20). Finally, we return S as the keyword-centric community we are searching for (line 21).

Example 4. Consider the HIN in Fig. 1(a) and the meta-path $\mathcal{P} = (APVPA)$. Let query vertex be a_1, $k = 4$ and $K = \{$"*Selection*", "*CIKM*"$\}$. First of all, we build the traversal trees rooted at each vertex in $S = \{a_1, a_2, a_3, a_4, a_5,\}$ as shown in Fig. 2. Consider the tree rooted at a_1 as a example, we can get $deg(a_1, S) = count(v_1, a_1) * (count(v_1, a_2) + count(v_1, a_3) + count(v_1, a_4)) + count(v_2, a_1) * (count(v_2, a_2) + count(v_2, a_3) + count(v_2, a_5)) = 1*(1+1+1)+1*(1+1+1) = 6$. Secondly, we can get $deg(a_2, S) = 6$, $deg(a_3) = 7$, $deg(a_4, S) = 3$, $deg(a_5, S) = 4$. Next, we can get $k\_neighbor[a_1] = \{a_2, a_3, a_4\}$, $k\_neighbor[a_2] = \{a_1, a_3, a_4\}$, $k\_neighbor[a_3] = \{a_1, a_2, a_4\}$, $k\_neighbor[a_4] = \{a_1, a_2, a_3\}$, $k\_neighbor[a_5] = \emptyset$. According to the above calculation, we can move a_4 and a_5 from S to $\mathcal{Q}$. Then we can update the $deg(v, S)$ and $k\_neighbor[v]$, $\forall v \in \{a_1, a_2, a_3\}$. According to lines 14–20 in Algorithm 3, we can get that $deg(a_1, S) = 6 - 1*1 - 1*1 = 4$, $deg(a_2, S) = 6-1*1-1*1 = 4$, $deg(a_2, S) = 6-1*1-1*1 = 4$, $k\_neighbor[a_1] = \{a_2, a_3\}$, $k\_neighbor[a_2] = \{a_1, a_3\}$, and $k\_neighbor[a_3] = \{a_1, a_2\}$. Finally we return $\{a_1, a_2, a_3\}$ as the keyword-centric community.

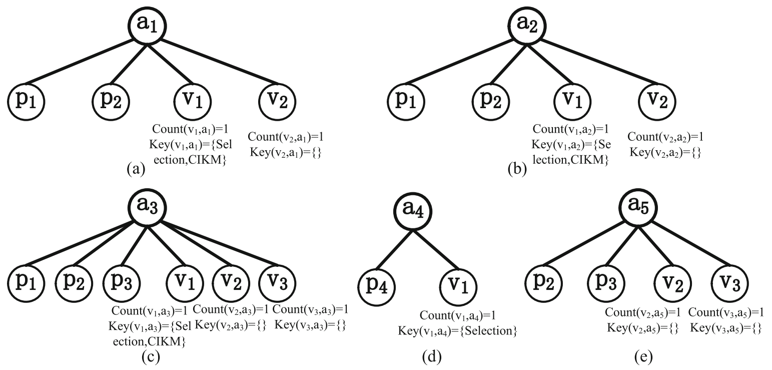


Fig. 2. An example for traversal trees

According to the above example, we can see that when we remove some vertices from S, we can quickly update $deg(v, S)$ and $k\_neighbor[v]$ $\forall v \in S$ using trees.

Theorem 3. *Given an HIN $G(V, E)$, a keywords set K, a query vertex q and the meta-path $\mathcal{P}$. Let l be the length of $\mathcal{P}$. The complexity of Algorithm 3 is $O(|S| * n^{\frac{l}{2}})$, where $n = |V|$.*

3.3 Optimization for the Approaches

In this section, We propose a new index to immediately get the numbers of the path instances of all meta paths, which can save much time when we handle the community search problem over HINs.

Algorithm 4: PreIndexTree

```
Input: The HIN graph $G(V, E)$; the HIN schema $G_s = (\mathcal{A}, \mathcal{R})$
1 insert all the meta paths of $G_s$ into the empty set $Q$;
2 foreach $v \in V$ do
3     construct a spanning tree rooted at $v$;
4     foreach $u \neq v$ AND $\psi(u) = \psi(v)$ do
5         $ins\_count(v, \mathcal{P}, u) \leftarrow$ the number of the path instances of $\mathcal{P}$ between $v$ and $u$, for
          each $\mathcal{P} \in Q$;
```

The process of building the index is shown in Algorithm 4. Based on the HIN schema, we maintain the number of the path instances of all the meta paths between v and the other vertices (lines 1–5). We can save a lot of time when calculating and updating the number of path instances as shown in lines 10 and 18 in Algorithm 3. The detail of the optimization approach is described in the following example.

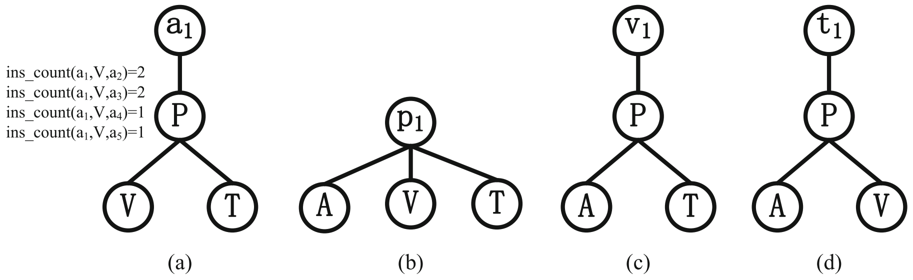


Fig. 3. A part of the new index trees

Example 5. Consider the HIN G in Fig. 1(a), the meta-path $\mathcal{P} = (APVPA)$ and the HIN schema $G_s = (\mathcal{A}, \mathcal{R})$. Let query vertex be a_1, $k = 4$ and $K = \{$"*Selection*", "*CIKM*"$\}$. First of all, we build the index trees rooted at each vertex in V respectively as shown in Fig. 3 which is the part of the index trees. Consider the index tree rooted at a_1 as a example as shown in Fig. 3(a), we can immediately get $deg(a_1, S) = 6$. Secondly we can get $deg(a_1, S) = ins\_count(a_1, V, a_2) + ins\_count(a_1, V, a_3) + ins\_count(a_1, V, a_4) + ins\_count(a_1, V, a_5) = 2 + 2 + 1 + 1 = 6$, $deg(a_2) = 6$, $deg(a_3) = 7$, $deg(a_4, S) = 3$, $deg(a_5, S) = 4$. Next, we can get $k\_neighbor[a_1] = \{a_2, a_3, a_4\}$, $k\_neighbor[a_2] = \{a_1, a_3, a_4\}$, $k\_neighbor[a_3] = \{a_1, a_2, a_4\}$, $k\_neighbor[a_4] = \{a_1, a_2, a_3\}$, $k\_neighbor[a_5] = \emptyset$. According to the above calculation, we can remove a_4 and a_5 from S. Then we can update the $deg(v, S)$ and $k\_neighbor[v]$, $\forall v \in \{a_1, a_2, a_3\}$ and get that $deg(a_1, S) = 6 - 1 * 1 - 1 * 1 = 4$, $deg(a_2, S) = 6 - 1 * 1 - 1 * 1 = 4$, $deg(a_3, S) = 7 - 1 * 1 - 2 * 1 = 4$, $k\_neighbor[a_1] = \{a_2, a_3\}$, $k\_neighbor[a_2] = \{a_1, a_3\}$, $k\_neighbor[a_3] = \{a_1, a_2\}$. Finally we return $\{a_1, a_2, a_3\}$ as the keyword-centric community.

4 Experiments

We now present the experimental results. We first discuss the experimental setup in Sect. 4.1.

4.1 Experimental Setup

To search the keyword-centric community in HINs, we implement three approaches called Baseline, AdvCore, and OptCore, respectively. Baseline is based on the basic algorithm Basic-$k\mathcal{KP}$-core, which is shown in Algorithm 1. AdvCore is the advanced approach with the *Advk*$\mathcal{KP}$-core algorithm as shown in Algorithm 2 and Algorithm 3. OptCore is the optimized approach as shown in Sect. 3.3. All algorithms are implemented in C++. All the experiments are conducted on a computer with Intel(R) Core(TM) i5-9500 CPU @ 3.00 GHz and 16G main memory. Windows 10 X64 operating system with kernel 18362.1139.

Table 1. Datasets used in the following experiments

Dataset	Vertices	Edges	Vertex types	Edge types
Foursquare	43,199	405,476	5	4
DBLP	682,819	1,951,209	4	3
IMDB	4,467,806	7,597,591	4	3
DBpedia	5,900,558	17,961,887	4	3

Datasets. We use four real datasets: Foursquare[1], DBLP[2], IMDB[3], and DBpedia[4]. Their detailed information is shown in Table 1. Foursquare contains the users' check-in records in the US, and there are five types of vertices in the dataset. DBLP contains the publication information in computer science areas, which has four types of vertices. IMDB contains the movie rating information since 2000, and there are four types of vertices in the dataset (actors, directors, writers, and movies). DBpedia is the data set extracted from Wikipedia.

Queries. For each dataset, we collect a set of meta-paths, and the size of the set is presented in Table 1. Based on the current works, we get that the default lengths of all meta-path we used in this paper do not exceed four unless otherwise specified. We collect all the possible meta-paths of the first two datasets, because the relationships in these two datasets are relatively small. For the other two datasets, there are a lot of relationships in these two datasets. Then we choose 50 meta-paths with the highest frequencies from the sets of the possible meta-paths of these two datasets, respectively. For each dataset, we generate 100 queries. To generate a query, we first randomly choose a meta-path. Then we first choose a vertex that has 50 instances or more starting with it and then get several keywords from a random instance of the meta-path starting with the chosen vertex. By default, we set k as 50, and for the results mentioned in the following, each value in the chart is the average result for these 100 queries.

4.2 Effectiveness Testing

4.2.1 Core Analysis

To analyze the proposed $k\mathcal{KP}$-core, we examine the size distribution of $k\mathcal{KP}$-core, where k ranges from 20 to 120. In this part, we only show results contain two different queries, which are $\mathcal{P}_1 = (TPAPT)$, $K_1 = \{\text{"20"}\}$ and $\mathcal{P}_2 = (APVPA)$, $K_2 = \{\text{"}\textit{report}\text{"}\}$ respectively. According to the result shown in Fig. 4, we can get that the proposed $k\mathcal{KP}$-core can achieve strong cohesiveness.

[1] https://sites.google.com/site/yangdingqi/home/foursquare-dataset.
[2] http://dblp.uni-trier.de/xml/.
[3] https://www.imdb.com/interfaces/.
[4] https://wiki.dbpedia.org/Datasets.

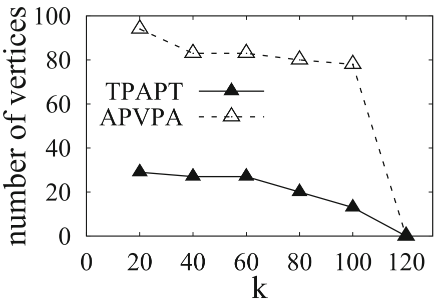


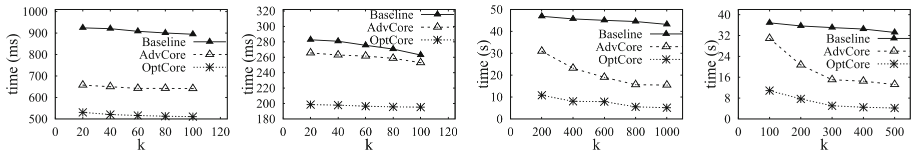
Fig. 4. Number of the vertices in $k\mathcal{KP}$-core

Table 2. Result of a case study on DBLP network.

$\mathcal{P}_1 = APVPA$	$\mathcal{P}_2 = APTPA$
$K_1 = \{Attack, Meltdown\}$	$K_2 = \{Attack, Meltdown\}$
Daniel Genkin, Daniel Gruss, Mickael Schwarz, Mike Hambury, Moritz Lipp, Paul Kocher, Stefan Mangard, Thomas Prescher, Werner Haas, Yuval Yarom	Daniel Genkin, Daniel Gruss, Diego Gragnaniello, Francesco Marra, Giovanni Poggi, Limin Zhang, Lu Feng, Luisa Verdoliva, Michael Schwarz, Mike Hamburg, Moritz Lipp, Paul Kocher, Pengyuan Lu, Stefan Mangard, Thomas Prescher, Werner Haas, Yuval Yarom

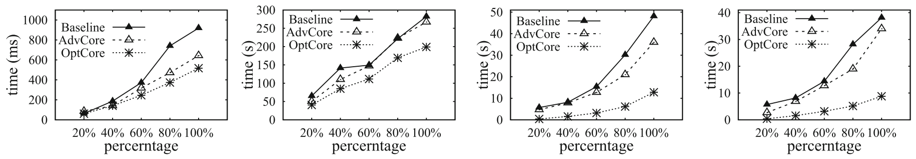
4.2.2 Case Study

We perform two queries on DBLP. In the first query, we set q = Prof. Paul Kocher, $\mathcal{P} = (APVPA)$, $K = \{$“*Attack*”, “*meltdown*”$\}$, and $k = 10$. Note that we regard the types of conference and journal as V. As shown in Table 2, the first community contains ten researchers who collaborated intensively. On the other hand, some researchers have published papers containing keyword “*Attack*” in a journal containing “*meltdown*”. This community includes those researchers who can cooperate in a specific field, which will help researchers find new collaborators to expand their research field. In the second query, we set q =Prof. Paul Kocher, $\mathcal{P} = (APTPA)$, $K = \{$“*Attack*”, “*Meltdown*”$\}$, and $k = 10$. We get the second community contains seventeen researchers, as shown in Table 2. We can see that the second community has seven more people than the first community because we use “T” to constraint the community instead of “V”. Compared with the first community, the second community realizes the discovery of potential collaborators without considering conferences or journals because the keywords “*meltdown*” contained by the vertices with vertex type “V” in DBLP are all lowercase. Therefore, the second community can help researchers find more potential partners than the first community.



(a) Foursquare (b) DBLP (c) IMDB (d) DBpedia

Fig. 5. Runtime of different algorithms



(a) Foursquare (b) DBLP (c) IMDB (d) DBpedia

Fig. 6. Scalability test of different algorithms

4.3 Efficiency Testing

Runtime of Baseline, AdvCore, and OptCore. We evaluate the runtime of Baseline, AdvCore, and OptCore for the keyword-centric community query in HINs. As shown in Fig. 5(a), OptCore is consistently faster than AdvCore and Baseline. Since Foursquare is sparse and small in scale, we can see that all these three algorithms have a short response time. As shown in Fig. 5(c), we can see that only OptCore can respond to the query within ten seconds. Unlike Foursquare, IMDB has many vertices and edges, and there are few vertex types in this data set, which means that for a meta path $\mathcal{P}$, there are many path instances of $\mathcal{P}$ in IMDB. Since OptCore is implemented based on the optimization approach, the response time of OptCore is short than the other two approaches.

Scalability Test. For each dataset, we randomly select 20%, 40%, 60%, 80%, and 100% vertices and get four subgraphs induced by these vertices, respectively. We run Baseline, AdvCore, and OptCore on all datasets. According to the results, as shown in Fig. 6, we can see that these three algorithms scale well with the number of vertices.

5 Related Work

Keyword Search. The keyword search over graphs mainly focuses on the connection between the vertices and the keywords in the query. The semantics used in the existing works can be roughly divided into two categories, one is tree semantics [8,14,18,22] and the other is subgraph semantics [19,21]. Among the tree semantics, Steiner trees are used in [2] to present a new backward search algorithm. In [8], a dynamic programming approach for finding all Steiner trees in graphs. The dynamic programming approach is feasible for input queries with a small number of keywords. The algorithm proposed in [13] follows Lawler's procedure [20] produces Steiner trees with polynomial delay. For the subgraph semantics, Kargar and An [19] find the subgraph containing all keywords in K, which is the set of the keywords. The authors use the sum of the shortest distance between all vertex pairs to measure the weight. Lei et al. [21] study the problem of clustering based on keywords. However, these semantics could be used to solve our problems.

Community Search. Community search aims to find connected subgraphs containing a query vertex. People use some metrics to ensure the cohesiveness of the community found in a graph. The minimum degree metric is the most frequent one used in the problem of the community search. It requires that the degree of each vertex in the community is at least k, which is similar to the constraint of the k-core [1,3,25]. For example, Sozio et al. proposed to find a community as the connected k-core containing the query vertex in [26]. Zhang et al. solve the keyword-centric community search problem over attribute graphs based on k-core in [32]. The other metrics used in the problem of community search are k-truss [5,15,31], k-clique [6,30] and K-ECC respectively. For example, Huang et al. and Chen et al. used k-truss as a metric to search the community in [4,16]; Yuan et al. proposed a k-clique percolation community model based on k-clique to solve the densest clique percolation community search problem in [30]. However, all these works focus on homogeneous graphs. We cannot use them to solve the keyword-centric community search problem over HINs.

6 Conclusion

In this paper, we study the problem of keyword-centric community search over HINs. We propose a basic algorithm, as shown in Algorithm 1 to find the community. However, the basic algorithm is very costly. Then we propose an advanced algorithm using a new method of traversing the search space based on trees. Since the trees are built based on the query vertex and the given meta-path $\mathcal{P}$, the advanced algorithm is not suitable for online query. According to that, we propose an optimization algorithm based on index trees to solve the problem. Extensive experiments on large real-world networks demonstrate the effectiveness and efficiency of our solution.

Acknowledgement. Zhiwei Zhang is supported by National Key R&D Program of China (Grant No. 2020YFB1707902), NSFC (Grant No. 62072035), Hong Kong GRF (Grant No. 12201518) and Zhejiang Lab (Grant No. 2020KE0AB04). Ye Yuan is supported by the NSFC (Grant No. 61932004) and the Fundamental Research Funds for the Central Universities (Grant No. N181605012). Guoren Wang is supported by the NSFC (Grant No. 61732003 and 61729201).

References

1. Batagelj, V., Zaversnik, M.: An O(m) algorithm for cores decomposition of networks. arXiv preprint cs/0310049 (2003)
2. Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., Sudarshan, S.: Keyword searching and browsing in databases using banks. In: ICDE, pp. 431–440. IEEE (2002)
3. Bonchi, F., Khan, A., Severini, L.: Distance-generalized core decomposition. In: ICDM, pp. 1006–1023 (2019)
4. Chen, L., Liu, C., Zhou, R., Li, J., Yang, X., Wang, B.: Maximum co-located community search in large scale social networks. Proc. VLDB Endow. **11**(10), 1233–1246 (2018)
5. Cohen, J.: Trusses: cohesive subgraphs for social network analysis. National Security Agency Technical Report 16, pp. 3–29 (2008)
6. Cui, W., Xiao, Y., Wang, H., Lu, Y., Wang, W.: Online search of overlapping communities. In: SIGMOD, pp. 277–288 (2013)
7. Cui, W., Xiao, Y., Wang, H., Wang, W.: Local search of communities in large graphs. In: SIGMOD, pp. 991–1002 (2014)
8. Ding, B., Yu, J.X., Wang, S., Qin, L., Zhang, X., Lin, X.: Finding top-k min-cost connected trees in databases. In: ICDE, pp. 836–845. IEEE (2007)
9. Fang, Y., Cheng, R., Luo, S., Hu, J.: Effective community search for large attributed graphs. Proc. VLDB Endow. **9**(12), 1233–1244 (2016)
10. Fang, Y., et al.: A survey of community search over big graphs. VLDB J. **29**(1), 353–392 (2020)
11. Fang, Y., Yang, Y., Zhang, W., Lin, X., Cao, X.: Effective and efficient community search over large heterogeneous information networks. Proc. VLDB Endow. **13**(6), 854–867 (2020)
12. Fortunato, S.: Community detection in graphs. Phys. Rep. **486**(3–5), 75–174 (2010)
13. Golenberg, K., Kimelfeld, B., Sagiv, Y.: Keyword proximity search in complex data graphs. In: SIGMOD, pp. 927–940 (2008)
14. Hristidis, V., Papakonstantinou, Y.: DISCOVER: keyword search in relational databases. In: VLDB, pp. 670–681. Elsevier (2002)
15. Huang, X., Cheng, H., Qin, L., Tian, W., Yu, J.X.: Querying k-truss community in large and dynamic graphs. In: SIGMOD, pp. 1311–1322 (2014)
16. Huang, X., Lakshmanan, L.V.: Attribute-driven community search. Proc. VLDB Endow. **10**(9), 949–960 (2017)
17. Huang, Z., Zheng, Y., Cheng, R., Sun, Y., Mamoulis, N., Li, X.: Meta structure: computing relevance in large heterogeneous information networks. In: KDD, pp. 1595–1604 (2016)
18. Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., Karambelkar, H.: Bidirectional expansion for keyword search on graph databases. In: VLDB, pp. 505–516 (2005)
19. Kargar, M., An, A.: Keyword search in graphs: finding r-cliques. Proc. VLDB Endow. **4**(10), 681–692 (2011)
20. Lawler, E.L.: A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. Manage. Sci. **18**(7), 401–405 (1972)
21. Li, G., Ooi, B.C., Feng, J., Wang, J., Zhou, L.: EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In: SIGMOD, pp. 903–914 (2008)

22. Liu, F., Yu, C., Meng, W., Chowdhury, A.: Effective keyword search in relational databases. In: SIGMOD, pp. 563–574 (2006)
23. Meng, C., Cheng, R., Maniu, S., Senellart, P., Zhang, W.: Discovering meta-paths in large heterogeneous information networks. In: WWW, pp. 754–764 (2015)
24. Newman, M.E., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E **69**(2), 026113 (2004)
25. Seidman, S.B.: Network structure and minimum degree. Soc. Netw. **5**(3), 269–287 (1983)
26. Sozio, M., Gionis, A.: The community-search problem and how to plan a successful cocktail party. In: KDD, pp. 939–948 (2010)
27. Sun, Y., Han, J., Zhao, P., Yin, Z., Cheng, H., Wu, T.: RankClus: integrating clustering with ranking for heterogeneous information network analysis. In: EDBT, pp. 565–576 (2009)
28. Sun, Y., Norick, B., Han, J., Yan, X., Yu, P.S., Yu, X.: PathSelClus: integrating meta-path selection with user-guided object clustering in heterogeneous information networks. TKDD **7**(3), 1–23 (2013)
29. Sun, Y., Yu, Y., Han, J.: Ranking-based clustering of heterogeneous information networks with star network schema. In: KDD, pp. 797–806 (2009)
30. Yuan, L., Qin, L., Zhang, W., Chang, L., Yang, J.: Index-based densest clique percolation community search in networks. IEEE Trans. Knowl. Data Eng. **30**(5), 922–935 (2017)
31. Zhang, Y., Yu, J.X.: Unboundedness and efficiency of truss maintenance in evolving graphs. In: SIGMOD, pp. 1024–1041 (2019)
32. Zhang, Z., Huang, X., Xu, J., Choi, B., Shang, Z.: Keyword-centric community search. In: ICDE, pp. 422–433. IEEE (2019)
33. Zhou, Y., Liu, L.: Social influence based clustering of heterogeneous information networks. In: KDD, pp. 338–346 (2013)