Effective and Efficient Community Search over Large Heterogeneous Information Networks

Yixiang Fang, Yixing Yang, Wenjie Zhang, Xuemin Lin, Xin Cao
University of New South Wales, Australia
yixiang.fang@unsw.edu.au, yixing.yang@unsw.edu.au
wenjie.zhang@unsw.edu.au, lxue@cse.unsw.edu.au, xin.cao@unsw.edu.au

ABSTRACT

Recently, the topic of community search (CS) has gained plenty of attention. Given a query vertex, CS looks for a dense subgraph that contains it. Existing studies mainly focus on homogeneous graphs in which vertices are of the same type, and cannot be directly applied to heterogeneous information networks (HINs) that consist of multi-typed, interconnected objects, such as the bibliographic networks and knowledge graphs. In this paper, we study the problem of community search over large HINs; that is, given a query vertex q, find a community from an HIN containing q, in which all the vertices are with the same type of q and have close relationships.

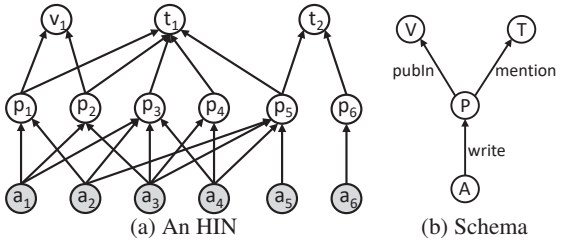
To model the relationship between two vertices of the same type, we adopt the well-known concept of *meta-path*, which is a sequence of relations defined between different types of vertices. We then measure the cohesiveness of the community by extending the classic minimum degree metric with a meta-path. We further propose efficient query algorithms for finding communities using these cohesiveness metrics. We have performed extensive experiments on five real large HINs, and the results show that the proposed solutions are effective for searching communities. Moreover, they are much faster than the baseline solutions.

PVLDB Reference Format:
Yixiang Fang, Yixing Yang, Wenjie Zhang, Xuemin Lin, Xin Cao. Effective and Efficient Community Search over Large Heterogeneous Information Networks. *PVLDB*, 13(6): 854-867, 2020.
DOI: https://doi.org/10.14778/3380750.3380756

1. INTRODUCTION

Heterogeneous information networks (HINs) are networks with multiple typed objects and multiple typed links denoting different semantic relations. These graph data sources are prevalent in various domains, including bibliographic information networks, social media, and knowledge graphs. Figure 1(a) illustrates an HIN of the DBLP network, which describes the relationship among entities of different types (i.e., author, paper, venue, and topic). In specific, it consists of six authors (i.e., a_1, $\cdots$, a_6), six papers (i.e., p_1, $\cdots$, p_6), one venue (i.e., v_1), and two topics (i.e., t_1 and t_2). The directed lines denote their semantic relationship. For example, the authors a_1 and a_2 have *written* a paper p_1, which *mentions* the topic t_1, *published* in the venue v_1.

Proceedings of the VLDB Endowment, Vol. 13, No. 6
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3380750.3380756



(a) An HIN (b) Schema

Figure 1: An example HIN with the DBLP network schema.

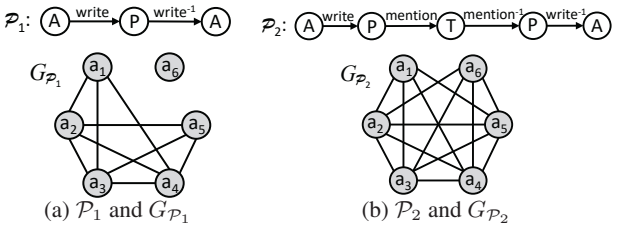
In this paper, we study the problem of Community Search over HINs (or CSH problem). To the best of our knowledge, this paper is the first work about CS over HINs. Given an HIN G and a query vertex $q \in G$, our goal is to find a community, or a set of vertices, from G containing q, in which all the vertices are with the same type of q and they are closely related. Particularly, the community satisfies the *meta-path*-based cohesiveness (i.e., its vertices are intensively connected by instances of a specific meta-path). The concept of meta-path has been extensively studied [62]; it is a sequence of vertex types and edge types between two given vertex types. In Figure 2(a), a meta-path $\mathcal{P}_1$, defined on authors (A) and papers (P), describes two authors with co-authorship. In Figure 1(a), the authors $\{a_1, a_2, a_3, a_4\}$ form a cohesive community, in which each pair of authors can be connected by a path instance of $\mathcal{P}_1$.

Table 1: Works on network community retrieval.

Network Type	Community Detection (CD)	Community Search (CS)
Homogeneous	[2, 14, 31, 39, 40, 49]	[15, 16, 26, 35, 37, 60] [9, 23, 24, 36, 41, 42]
Heterogeneous	[59, 61, 63–65, 81]	**CSH** (This work)

Prior works. Existing works on network community retrieval can be roughly classified into *community detection* (CD) and *community search* (CS). Some representative works are summarized in Table 1. Generally, CD algorithms aim to identify all communities for a graph [31, 49, 63–65, 81]. These studies are not "query-based", i.e., they are not customized for a query request (e.g., a user-specified query vertex). Moreover, for large graphs, they often take a long time to detect all the communities, so they are not suitable for *online* queries. To address these issues, the query-based CS approaches (e.g., [15, 26, 35, 60]) have been recently studied. However, previous CS approaches mainly focus on *homogeneous* networks where all the vertices are of the same type, while in HINs, both vertices and edges are multiply typed and they carry different semantic meanings, so it does not make sense to mix up them for performing CS using existing approaches.

CSH problem. In this paper, we focus on searching communities in HINs, in which vertices are with a specific type (e.g.,



(a) $\mathcal{P}_1$ and $G_{\mathcal{P}_1}$ (b) $\mathcal{P}_2$ and $G_{\mathcal{P}_2}$

Figure 2: The induced homogeneous graphs using $\mathcal{P}_1$ and $\mathcal{P}_2$.

a community of authors in the DBLP network). Conceptually, a community is a set of vertices which are *connected cohesively*. To formulate the CSH problem, we face two key questions: (1) How to *connect* two vertices of the same type? (2) How to measure the *cohesiveness* of a community?

For the first question, it is nontrivial to answer, since vertices of the same type (e.g., conferences in the DBLP network) may not be connected directly in the HIN. To connect vertices, we adopt the well-known concept of *meta-path* (e.g., $\mathcal{P}_1$ in Figure 2(a)), or a sequence of relations defining a composite relation between its starting type and ending type [48, 62].

For the second question, existing CS solutions often adopt well-known metrics, e.g., minimum degree [15, 24, 42, 60], k-truss [35, 37], and k-clique [16, 78], to measure the community cohesiveness. Among them, the *minimum degree* is the most common metric [26], which ensures that each vertex is well engaged in the community, i.e., it has at least k neighbors in the community. In this paper, we extend this metric for HINs; that is, for each vertex v of a community $\mathcal{C}$, there are at least k other vertices, which can be connected to v via instances of a particular meta-path $\mathcal{P}$, within $\mathcal{C}$. To answer this query, a simple solution is to build a homogeneous graph $G_{\mathcal{P}}$ by linking pairs of vertices if there is an instance of $\mathcal{P}$ between them, and then run an existing CS solution (e.g., [60]). For example, consider the HIN in Figure 1(a). Let q=a_1, $\mathcal{P}$=$\mathcal{P}_1$, and k=3. We then can build a homogeneous graph $G_{\mathcal{P}_1}$ as shown in Figure 2(a), and get a community $\mathcal{C}_1$=$\{a_1, \cdots, a_5\}$.

Although the basic metric above is straightforward, it may lead to some vertices weakly engaged in the community. Consider the author a_5, for instance, in the community $\mathcal{C}_1$. Although a_5 has three co-authors, a_5 publishes only one paper p_5, while each other author has three papers. In other words, a_5 may be a junior researcher, but is included in a community of senior researchers. This is because the edge "$a_5 \rightarrow p_5$" is shared by three path instances of $\mathcal{P}_1$. Moreover, this scenario could be even common for long meta-paths. For example, let us replace $\mathcal{P}_1$ by $\mathcal{P}_2$ in the query above. We can obtain another homogeneous graph $G_{\mathcal{P}_2}$ (Figure 2(b)) and get a community $\mathcal{C}_2$=$\{a_1, \cdots, a_6\}$. Notice that all the path instances linked to a_6 share edges "$a_6 \rightarrow p_6$", "$p_6 \rightarrow t_2$", and "$p_5 \rightarrow t_2$". Clearly, if any of them is removed, a_6 will be isolated, and thus it is weakly engaged in $\mathcal{C}_2$. The weak engagement issue above cannot be trivially resolved by normalizing the strength between vertex pairs (e.g., using PathSim [62]), since it does not change the topology of $G_{\mathcal{P}}$. In addition, as noted in [50], transforming an HIN to a homogeneous graph may not be meaningful because it may cause issues of high degrees and high clustering coefficients.

Recall that existing CS studies [15, 24, 41, 42, 60] require each vertex in the community has at least k neighbors, which are linked by *different* edges. This imposes strong cohesiveness since each vertex is still engaged in the community after removing any (k–1) edges. Inspired by this concept, we propose to use *different* meta-paths to model the cohesiveness. Specifically, we introduce two novel metrics by using *edge-* and *vertex-disjoint* paths, which require that each vertex v is connected to at least k vertices via *edge-* and *vertex-disjoint* paths of $\mathcal{P}$ respectively. For example, consider

(a) Disjoint paths from a_1 (b) Disjoint path from a_5

Figure 3: Illustrating edge- and vertex-disjoint paths.

the query above (q=a_1, $\mathcal{P}$=$\mathcal{P}_1$, and k=3). By using edge-disjoint paths, a_1 can be connected to three vertices, while a_5 is connected to at most one vertex, as depicted in Figure 3 (each path is represented by a specific kind of dashed line). We then get a new community $\mathcal{C}_1'$=$\{a_1, \cdots, a_4\}$ for a_1. Notice that $\mathcal{C}_1' \subseteq \mathcal{C}$ and $a_5 \notin \mathcal{C}_1'$. Clearly, $\mathcal{C}_1'$ is more cohesive than $\mathcal{C}_1$ since each vertex is still engaged in $\mathcal{C}_1'$ after removing any two edges. Similarly, by using vertex-disjoint paths, we can find the community $\mathcal{C}_1'$ for a_1.

To further illustrate this, we conduct a case study on a small DBLP network by two queries with different cohesiveness metrics. The results are reported in Table 2, where "Basic" denotes the metric directly extended from the minimum degree metric. In the first query, we let q be Prof. Xuemin Lin (a researcher in the database area), and set k=5 and $\mathcal{P}$=$\mathcal{P}_1$. The two communities contain six researchers collaborated intensively, but the first one has several additional authors who published just one or two papers. For the second query, we let q be SIGMOD conference, and set k=5 and $\mathcal{P}$=$\mathcal{P}_3$. Clearly, by edge-disjoint metric, we can find all the six top database conferences, while the basic metric cannot achieve this.

Main features. The CSH query has some nice features: (1) It can find *different types* of communities. As shown in Table 2, it finds communities of authors, as well as venues, by using different meta-paths. (2) The query can be *personalized*. As aforementioned, different meta-paths carry different relationships. By specifying different meta-paths for a single query vertex, we can get communities with different semantic relationships. (3) The query can be evaluated in an *online* manner. As we will show later, we have developed efficient query algorithms, allowing the community to be generated quickly upon a query request.

Applications. Here are typical applications of the CSH query: (1) *Event planning.* For instance, to hold a workshop, a researcher can issue a CSH query with the meta-path $\mathcal{P}_1$ on the DBLP network and hold the workshop based on the returned community. (2) *Social marketing.* An e-commence platform (e.g., Alibaba) often maintains an HIN of users and products. To boost sales figures for a product x, advertisement messages can be sent to groups of users with co-purchase behaviours before, where the groups can be identified by CSH queries on users who have purchased x. (3) *Recommendation.* In a movie database (e.g., IMDB), to suggest movies for a user u, we can find u's community $\mathcal{C}$ by a CSH query with a meta-path linking two users by a movie, and then recommend movies based on $\mathcal{C}$. (4) *Biological data analysis [18, 54].* For example, in the HIN of genes and diseases, identifying a community of genes could reveal the hidden links in different diseases [18].

Challenges and contributions. The minimum degree metric is also used in defining a k-core [5, 57], which is the largest subgraph of a homogeneous graph such that each vertex's degree is at least k. Inspired by this model, we propose three core models, namely *basic*, *edge-disjoint*, and *vertex-disjoint* (k, $\mathcal{P}$)-cores, by incorporating a meta-path $\mathcal{P}$ into the k-core model respectively. Then, a CSH query can be answered by computing a specific core containing the query vertex. However, computing these cores is very challenging.

For the basic (k, $\mathcal{P}$)-core, the community can be computed from the induced homogeneous graph $G_{\mathcal{P}}$. However, $G_{\mathcal{P}}$ is often much denser than traditional graphs. For example, on the DBLP dataset, the average degree of vertices with type "A" is 3.46, but the average degree of vertices with type "A" in $G_{\mathcal{P}_2}$ is 363.63, where

Table 2: Results of a case study on a small DBLP network.

$\mathcal{P}_1$: A –write→ P –write⁻¹→ A		$\mathcal{P}_3$: V –publn⁻¹→ P –write⁻¹→ A –write→ P –publn→ V	
Basic	Edge-disjoint	Basic	Edge-disjoint
Xuemin Lin Jeffrey Xu Yu Lu Qin Ying Zhang Wenjie Zhang Lijun Chang Other 13 authors	Xuemin Lin Jeffrey Xu Yu Lu Qin Ying Zhang Wenjie Zhang Lijun Chang	SIGMOD, ICDE PVLDB, CIKM EDBT, DASFAA CCS, SSP, ICC GLOBECOM USENIX Security	SIGMOD, ICDE PVLDB, CIKM EDBT, DASFAA

$\mathcal{P}_2$=$(APTPA)$. Hence, this imposes great challenges for computing basic $(k, \mathcal{P})$-cores, especially when $\mathcal{P}$ is very long. To alleviate this issue, we develop an advanced algorithm `FastBCore`, which does not need to enumerate all the path instances.

For edge- and vertex-disjoint $(k, \mathcal{P})$-cores, the first challenge is that they cannot be computed from $G_{\mathcal{P}}$, as it does not reveal any information about whether path instances are disjoint or not. Thus, we have to compute them from the HIN, which it is harder than computing the basic $(k, \mathcal{P})$-core. The second challenge is how to efficiently compute the maximum numbers of vertices linked by edge- and vertex-disjoint paths respectively. The exact algorithms of computing them are based on the max-flow algorithm, which however is very costly, since they may take up to $\mathcal{O}(|V_F| \cdot |E_F|)$ time, where V_F and E_F denote the sets of vertices and edges in the flow network, so it is costly for large HINs. To alleviate this issue, we develop a linear-time approximation algorithm with theoretical guarantee. Besides, we have developed a batch peeling strategy to speedup the peeling process when computing the cores.

In addition, although online algorithms above are fast, they may be inefficient when the queries are executed frequently. We further improve the efficiency by developing a space-efficient index, which allows the query to be completed in optimal query time cost.

We have conducted extensive experiments on five real large HINs. The results show that the communities based on edge- and vertex-disjoint $(k, \mathcal{P})$-cores are more cohesive than those of the basic one, and the online query algorithms are generally fast. Meanwhile, the basic $(k, \mathcal{P})$-core takes the least running time, while vertex-disjoint $(k, \mathcal{P})$-core is the most time consuming one. In addition, the index-based queries are much faster than online queries.

Outline. We formulate the CSH problem in Section 2. In Sections 3 and 4, we present online algorithms and index-based algorithms. We report experimental results in Section 5. We review the related work in Section 6 and conclude in Section 7.

2. PROBLEM DEFINITION

2.1 Preliminaries

We summarize notations frequently used in this paper in Table 3.

DEFINITION 1. ***HIN [38,62].*** *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 [38, 62].*** *Given an HIN G= (V, E) with mappings $\psi : V \rightarrow \mathcal{A}$ and $\phi : E \rightarrow \mathcal{R}$, its schema T_G is a directed graph defined over vertex types $\mathcal{A}$ and edge types (as relations) $\mathcal{R}$, i.e., T_G=($\mathcal{A}$, $\mathcal{R}$).*

The HIN schema describes all allowable edge types between vertex types, where each edge type can describe one-to-one, one-to-many, or many-to-many relationship. Figure 1(b) shows the schema of DBLP network, where the vertices labelled "A", "P", "V", and "T" denote author, paper, venue, and topic, respectively. Note that if there is an edge R from vertex type A to vertex type B, the inverse edge R^{-1} naturally exists from B to A.

Table 3: Notations and meanings.

Notation	Meaning
G=(V, E)	an HIN with vertex set V and edge set E
$\psi(v)$ ($\phi(e)$)	the vertex (edge) type of a vertex v (edge e)
$\mathcal{P}$	a symmetric meta-path defined on the schema of G
l	the length of $\mathcal{P}$ (equals to the #. of edges of $\mathcal{P}$)
n_i	#. of vertices whose types match with i-th vertex in $\mathcal{P}$
$G_{\mathcal{P}}$	a homogeneous graph induced by a meta-path $\mathcal{P}$ on G
$\mathbf{B}_k$ ($\mathbf{B}_{k,\mathcal{P}}$)	a basic (k, $\mathcal{P}$)-core
$\mathbf{E}_k$ ($\mathbf{E}_{k,\mathcal{P}}$)	an edge-disjoint (k, $\mathcal{P}$)-core
$\mathbf{V}_k$ ($\mathbf{V}_{k,\mathcal{P}}$)	a vertex-disjoint (k, $\mathcal{P}$)-core
$\alpha(v, S)$	the b-degree, i.e., #. of path instances starting from the vertex v and ending at vertices in a set S
$\beta(v, S)$ ($\gamma(v, S)$)	the e-degree (v-degree), i.e., the maximum #. of edge-disjoint(vertex-disjoint) path instances which start from vertex v and end at vertices in a set S

DEFINITION 3. ***Meta-path [62].*** *A meta-path $\mathcal{P}$ is a path defined on an HIN schema T_G=($\mathcal{A}$, $\mathcal{R}$), and is denoted in the form $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \cdots \xrightarrow{R_l} A_{l+1}$, where l is the length of $\mathcal{P}$, $A_i \in \mathcal{A}$, and $R_i \in \mathcal{R}$ ($1 \leq i \leq l$).*

We also use vertex type names to denote a meta-path, i.e., $\mathcal{P}$=(A_1 $A_2 \cdots A_{l+1}$), if there exist no multiple edges between the same pair of vertex types. We call a meta-path $\mathcal{P}'$ the *reverse* meta-path of $\mathcal{P}$, if $\mathcal{P}'$ is the reverse path of $\mathcal{P}$ in T_G, and denote it by $\mathcal{P}^{-1}$. We say $\mathcal{P}$ is symmetric, if it is the same with $\mathcal{P}^{-1}$. For example, the meta-path $\mathcal{P}_1$ (Figure 2(a)) can be written as $\mathcal{P}_1$=(APA). Since its reverse meta-path is still $\mathcal{P}_1$, it is a symmetric meta-path.

We call a path p=$a_1 \rightarrow a_2 \cdots \rightarrow a_{l+1}$ between vertices a_1 and a_{l+1} a *path instance* of $\mathcal{P}$, if $\forall i$, the vertex a_i and edge e_i=(a_i, a_{i+1}) satisfy $\psi(a_i)$=A_i and $\phi(e_i)$=R_i. For example, in Figure 1(a), the path $a_1 \rightarrow p_1 \rightarrow a_2$ is a path instance of $\mathcal{P}_1$. Here we use lower-case letters (e.g., a_1) to denote vertices in an HIN, and upper-case letters (e.g., A) to denote vertex types. We say that a vertex u is a *$\mathcal{P}$-neighbor* of a vertex v, if they can be connected by an instance of $\mathcal{P}$. We say that two vertices u and v are *$\mathcal{P}$-connected*, if there exists a chain of vertices from u to v, such that any vertex is a $\mathcal{P}$-neighbor of its adjacent vertex in the chain.

To characterize the cohesiveness of a community, existing works [15, 60] often use k-core [5, 26, 57]. Given a homogeneous graph H and an integer k (k>0), the k-core is the largest subgraph of H, denoted by H_k, such that each vertex has a degree of k or more within H_k. In Figure 2(a), for example, 1-core, 2-core, and 3-core contain vertices $\{a_1, \cdots, a_5\}$.

2.2 Problem Definition

In this paper, we aim to find a community from an HIN containing a query vertex q, in which all the vertices are with the type $\psi(q)$, also called *target* type. Conceptually, vertices in the community should *connect cohesively*. To connect vertices with the target type, we adopt a symmetric meta-path $\mathcal{P}$, whose starting and ending types are with target type. To characterize cohesiveness, we extend the classic k-core as $(k, \mathcal{P})$-cores by incorporating a symmetric meta-path $\mathcal{P}$. Note that all the meta-paths mentioned later in this paper are symmetric; for nonsymmetric meta-paths, we can extend the core model for directed graphs [32] in a similar manner.

Let $\mathcal{P}$ be a meta-path linking two vertices with the target type. Given a vertex v and a set S of vertices with target type, we define $\alpha(v, S)$, called basic-degree or b-degree, as the number of $\mathcal{P}$-neighbor of v within the set S. Based on the concept of b-degree, we introduce the basic $(k, \mathcal{P})$-core model as follows.

DEFINITION 4. ***Basic (k, $\mathcal{P}$)-core.*** *Given an HIN G and an integer k, a basic (k, $\mathcal{P}$)-core of G is a maximal set $\mathbf{B}_{k,\mathcal{P}}$ of $\mathcal{P}$-connected vertices, s.t. $\forall v \in \mathbf{B}_{k,\mathcal{P}}$, $\alpha(v, \mathbf{B}_{k,\mathcal{P}}) \geq k$, where vertices of $\mathbf{B}_{k,\mathcal{P}}$ are with the type linked by $\mathcal{P}$.*

As aforementioned, the basic (k, $\mathcal{P}$)-core may include vertices that are weakly engaged in the core since many path instances may share the same edges. To tackle this issue, we develop novel core models based on the *disjoint* paths. Specifically, consider a vertex v with target type and let $\Psi[v]$ be a set of path instances of $\mathcal{P}$ starting from v. We say that $\Psi[v]$ is a set of *edge-disjoint* paths, if for any two path instances $p_1, p_2 \in \Psi[v]$, their i-th ($1 \leq i \leq l$) edges are different and (l+1)-th vertices are different. Similarly, we say that $\Psi[v]$ is a set of *vertex-disjoint* paths, if for any two path instances $p_1, p_2 \in \Psi[v]$, their i-th ($2 \leq i \leq l$+1) vertices are different. Note that two edge- or vertex-disjoint paths may share the same edges or vertices, which will be in different positions of these two paths.

A vertex is often involved in multiple sets of edge-disjoint or vertex-disjoint paths. In this paper, we consider the maximum set and use $\beta(v, S)$ (resp., $\gamma(v, S)$) to denote the maximum number of edge-disjoint (resp., vertex-disjoint) path instances starting from vertex v and ending at vertices in a set S. For simplicity, we call $\beta(v, S)$ (resp., $\gamma(v, S)$) e-degree (resp., v-degree) of v. In Figure 1(a), let v=a_1 and S=$\{a_1, \cdots, a_6\}$. If $\mathcal{P}$=(APA), then $\beta(a_1, S)$=$\gamma(a_1, S)$=3, where the maximum sets of edge- and vertex-disjoint paths are shown in Figure 3(a). Similarly, if $\mathcal{P}$= $(APTPA)$, then we have $\beta(a_1, S)$=3 and $\gamma(a_1, S)$=1.

DEFINITION 5. ***Edge-disjoint (k, $\mathcal{P}$)-core.*** *Given an HIN G and an integer k, an edge-disjoint (k, $\mathcal{P}$)-core of G is a maximal set $\mathbf{E}_{k,\mathcal{P}}$ of $\mathcal{P}$-connected vertices s.t. $\forall v \in \mathbf{E}_{k,\mathcal{P}}$, $\beta(v, \mathbf{E}_{k,\mathcal{P}}) \geq k$, where vertices of $\mathbf{E}_{k,\mathcal{P}}$ are with the type linked by $\mathcal{P}$.*

DEFINITION 6. ***Vertex-disjoint (k, $\mathcal{P}$)-core.*** *Given an HIN G and an integer k, an vertex-disjoint (k, $\mathcal{P}$)-core of G is a maximal set $\mathbf{V}_{k,\mathcal{P}}$ of $\mathcal{P}$-connected vertices s.t. $\forall v \in \mathbf{V}_{k,\mathcal{P}}$, $\gamma(v, \mathbf{V}_{k,\mathcal{P}}) \geq k$, where vertices of $\mathbf{V}_{k,\mathcal{P}}$ are with the type linked by $\mathcal{P}$.*

In the context without ambiguity, we write $\mathbf{B}_{k,\mathcal{P}}$, $\mathbf{E}_{k,\mathcal{P}}$, and $\mathbf{V}_{k,\mathcal{P}}$ as $\mathbf{B}_k$, $\mathbf{E}_k$, and $\mathbf{V}_k$, respectively. We say that a vertex v has a core number k, if it is the largest k such that there is a corresponding (k, $\mathcal{P}$)-core containing v. In other words, each vertex of a (k, $\mathcal{P}$)-core must have a corresponding core number of k or more.

EXAMPLE 1. *Consider Figure 1(a) and let $\mathcal{P}$=(APA). For basic (k, $\mathcal{P}$)-cores, $\mathbf{B}_k$=$\{a_1, \cdots, a_5\}$ where $1 \leq k \leq 3$. For edge-disjoint (k, $\mathcal{P}$)-cores, $\mathbf{E}_1$= $\{a_1, \cdots, a_5\}$, and $\mathbf{E}_2$=$\mathbf{E}_3$= $\{a_1, a_2, a_3, a_4\}$. For vertex-disjoint (k, $\mathcal{P}$)-cores, $\mathbf{V}_k$=$\mathbf{E}_k$ where $1 \leq k \leq 3$. There is no $\mathbf{B}_4$, $\mathbf{E}_4$, or $\mathbf{V}_4$. We take author a_5 as an example to illustrate core numbers. Its core numbers are 3, 1, and 1, under the basic, edge-disjoint, and vertex-disjoint core models, respectively.*

Now we formally introduce the CSH problem as follows.

PROBLEM 1. *Given an HIN G, a query vertex q, a symmetric meta-path $\mathcal{P}$, an integer k (k>0), and a specific (k, $\mathcal{P}$)-core model, return the corresponding (k, $\mathcal{P}$)-core containing q.*

In Example 1, let q=a_1, $\mathcal{P}$= (APA), and k=3. If we specify the basic (k, $\mathcal{P}$)-core as the community model, we get a community $\mathbf{B}_3$; If the edge- and vertex-disjoint (k, $\mathcal{P}$)-core is adopted, we can get communities $\mathbf{E}_3$ and $\mathbf{V}_3$ respectively. Clearly, since any (k, $\mathcal{P}$)-core is a maximal set of $\mathcal{P}$-connected vertices, the community satisfies the properties of structural maximality and connectivity.

2.3 Properties of (k, $\mathcal{P}$)-cores

Below, we show some interesting properties of (k, $\mathcal{P}$)-cores.

PROPOSITION 1. *Given an HIN and a meta-path $\mathcal{P}$, the basic (k, $\mathcal{P}$)-cores are nested, i.e., for any $\mathbf{B}_{k+1} \neq \emptyset$, there exists a $\mathbf{B}_k$ such that $\mathbf{B}_{k+1} \subseteq \mathbf{B}_k$. Similarly, the property holds for edge- and vertex-disjoint (k, $\mathcal{P}$)-cores.*

PROOF. The proposition directly follows the observation. □

PROPOSITION 2. *Given an HIN and a meta-path $\mathcal{P}$ with l=2, for any $\mathbf{E}_k$, there exists a $\mathbf{V}_k$ such that $\mathbf{V}_k$=$\mathbf{E}_k$.*

PROOF. The proposition directly follows the observation. □

PROPOSITION 3. *Given an HIN, a meta-path $\mathcal{P}$, and an integer k, for any two basic (k, $\mathcal{P}$)-cores $\mathbf{B}_k$ and $\mathbf{B}'_k$, if $\mathbf{B}_k \cap \mathbf{B}'_k \neq \emptyset$, then $\mathbf{B}_k$=$\mathbf{B}'_k$. Similarly, the edge- and vertex-disjoint (k, $\mathcal{P}$)-cores have such a property.*

PROOF. We prove the proposition by contradiction. Suppose that $\mathbf{B}_k \neq \mathbf{B}'_k$. Then, $\mathbf{B}_k \cup \mathbf{B}'_k$ is a larger basic (k, $\mathcal{P}$)-core by Definition 4, which contradicts the maximality of $\mathbf{B}_k$ and $\mathbf{B}'_k$. □

THEOREM 1. *Given an HIN G and a meta-path $\mathcal{P}$, for any $\mathbf{V}_k$, there exists an $\mathbf{E}_k$ and a $\mathbf{B}_k$ such that $\mathbf{V}_k \subseteq \mathbf{E}_k \subseteq \mathbf{B}_k$.*

PROOF. By Definition 6, $\forall v \in \mathbf{V}_k$, it has a set of k vertex-disjoint path instances of $\mathcal{P}$. For each $2 \leq i \leq l$, since the i-th vertices of these paths different, the edges connecting i-th vertices and (i+1)-th vertices must be different. In other words, these k vertex-disjoint paths are also edge-disjoint paths, and thus $\mathbf{V}_k \subseteq \mathbf{E}_k$. Similarly, we have $\mathbf{E}_k \subseteq \mathbf{B}_k$. Hence, the theorem holds. □

3. ONLINE QUERY ALGORITHMS

In this section, we develop efficient algorithms for the CSH problem. In particular, for each core model, we develop a basic algorithm and an advanced algorithm. While basic algorithms are straightforward, they are not as efficient as advanced algorithms, as shown by our experiments. We summarize all the advanced algorithms in Table 4, where more ★ means higher cohesiveness, n_i is the number of vertices with i-th vertex type in $\mathcal{P}$, $d_{i,i+1}$ is the maximum number of vertices with (i+1)-th vertex type that are connected to a vertex with i-th vertex type in $\mathcal{P}$, $\sigma_2 < {n_1}^2$, and $c \leq 4$. Clearly, querying $\mathbf{B}_k$ takes the least time while computing $\mathbf{V}_k$ is

Table 4: Overview of the cohesiveness and complexity.

Core	Algorithm	Cohesiveness	Complexity of advanced algorithm
$\mathbf{B}_k$	FastBCore	★★★	$\mathcal{O}(n_1 \cdot d_{1,2} + n_1 \sum_{i=1}^{l} n_i \cdot d_{i,i+1})$
$\mathbf{E}_k$	BatchECore	★★★★	$\mathcal{O}(\sigma_2(n_1 \cdot d_{1,2} + n_1 \cdot \sum_{i=2}^{l} n_i \cdot d_{i,i+1}))$
$\mathbf{V}_k$	BatchVCore	★★★★★	$\mathcal{O}(c \cdot \sigma_2(n_1 \cdot d_{1,2} + n_1 \cdot \sum_{i=2}^{l} n_i \cdot d_{i,i+1}))$

the most time consuming one. On the other hand, $\mathbf{E}_k$ tends to be more cohesive than $\mathbf{B}_k$, because in $\mathbf{E}_k$, each vertex is still engaged in the community after removing any (k–1) edges. Also, $\mathbf{V}_k$ is more cohesive than $\mathbf{E}_k$ because the edge-disjoint paths may share the same vertices while the vertex-disjoint paths share neither an edge nor a vertex. In summary, there is a trade-off between the cohesiveness of results and query efficiency, i.e., a more cohesive core takes higher time computational cost in most cases. We remark that in certain extreme cases (e.g., the Yeast PPI network [20]), the HIN is simply comprised by large stars and each vertex is linked to only one or two edge- and vertex-disjoint paths, the edge- and vertex-disjoint core models may not achieve higher cohesiveness.

Our later experiments also indicate that for moderate and large graphs, it is better to adopt $\mathbf{E}_k$ or $\mathbf{V}_k$ because computing them

achieves good quality within reasonable time cost; while if the number of vertices with the target type is very large, $\mathbf{B}_k$ should be a better option as its computation is faster than others.

3.1 Algorithms for Basic (k, $\mathcal{P}$)-cores

In this section, we present two query algorithm for the basic core model, including a basic algorithm and an advanced algorithm.

3.1.1 A Basic Algorithm

A basic algorithm is to build an induced homogeneous graph $G_{\mathcal{P}}$ first and then return the connected k-core containing q from $G_{\mathcal{P}}$. In specific, it consists of three steps: (1) collect a set S of all vertices with target type; (2) for each vertex $v \in S$, enumerate the set $\Psi[v]$ of all path instances of $\mathcal{P}$ starting with v and add an edge between v and each of its $\mathcal{P}$-neighbors using $\Psi[v]$; and (3) find the connected k-core containing q. This algorithm, however, is very costly for long meta-paths on large HINs since in step (2), the size of $\Psi[v]$ could be exponentially large, i.e., $\mathcal{O}(n^l)$, where n is the maximum number of vertices for a vertex type in $\mathcal{P}$ and l is the length of $\mathcal{P}$. To speedup step (2), we propose a *batch search* strategy. Instead of

Algorithm 1: The algorithm: `HomBCore`.

```
   Input: G, q, P, k;
   Output: B_k;
1  collect the set S of vertices with the target type;
2  for each vertex v ∈ S do
3      initialize a set X={v};
4      for i ← 1 to l do
5          Y ← ∅;
6          for each vertex u ∈ X do
7              for each neighbor t of u do
8                  if (u,t) matches with i-th edge of P then Y.add(t);
9          X ← Y;
10     for each vertex u ∈ X do add an edge between v and u;
11 B_k ← compute the connected k-core containing q from G_P;
12 return B_k;
```

enumerating all the path instances, we decompose $\mathcal{P}$ into a list of edges and find matched vertices for each of them in a batch manner. We call the improved algorithm `HomBCore`, which is shown in Algorithm 1. We first find S (line 1). Then, we initialize a set X for each vertex v (lines 2-3), and get its $\mathcal{P}$-neighbors by finding vertices that match with each edge of $\mathcal{P}$ in a batch manner (lines 4-9). Finally, we compute $\mathbf{B}_k$ from $G_{\mathcal{P}}$ (lines 10-12).

LEMMA 1. *The total time cost of* `HomBCore` *is* $\mathcal{O}(n_1 \cdot d_{1,2} + n_1 \sum_{i=1}^{l} n_i \cdot d_{i,i+1})$ *time.*

PROOF. Please see appendices of the technical report [75]. □

3.1.2 An Advanced Algorithm

The major limitation of `HomBCore` is that it has to build an induced homogeneous graph $G_{\mathcal{P}}$ for all the vertices with the target type. This, however, is costly and unnecessary, because (1) not all the vertices with target type are $\mathcal{P}$-connected to q; and (2) it finds all the $\mathcal{P}$-neighbors for each vertex with target type. To tackle these two issues, we propose two labelling strategies, namely *batch search with labelling* and *depth-first search with labelling*.

Batch search with labelling (BSL). The BSL strategy is developed for efficiently finding all the vertices that are $\mathcal{P}$-connected to q. It is based on the batch search in `HomBCore`, but with labelling. For example, in Figure 1(a), let q=a_1 and $\mathcal{P}$=(APA). By using BSL, we will find five authors $\{a_1, \cdots, a_5\}$. Notice that author a_6 is excluded since it is not $\mathcal{P}$-connected to a_1.

Detailed steps of BSL are presented in Algorithm 2 (lines 1-11). Specifically, we first find q's $\mathcal{P}$-neighbor set using batch search.

Algorithm 2: The algorithm: `FastBCore`.

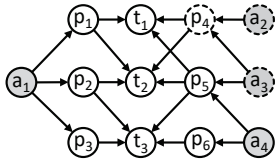
```
   Input: G, q, P, k;
   Output: B_k;
1  S ← ∅, X ← {q}, Q ← ∅, Ψ[ ] ← ∅;
2  while |X|>0 do
3      for i ← 1 to l do
4          Y ← ∅;
5          for each vertex v ∈ X do
6              for each neighbor u of v do
7                  if (v,u) matches with i-th edge of P then
8                      if (v,u) does not have a label i then
9                          Y.add(u) and attach a label i to (v,u);
10         X ← Y;
11     X ← X\S, S ← S ∪ X;
12 for each vertex v ∈ S do
13     Ψ[v] ← find up to k path instances of P which start from v;
14     if |Ψ[v]|<k then Q.add(v);
15 while |Q|>0 do
16     v ← Q.poll();
17     S ← S\{v}, U ← {u|u is P-connected to v by a path in Ψ[v]};
18     for each vertex u ∈ U do
19         if v is P-connected to u by a path instance p ∈ Ψ[u] then
20             remove p from Ψ[u];
21             if |Ψ[u]|<k then
22                 p' ← find a new path instance starting from u;
23                 if p' exists then Ψ[u] ← Ψ[u] ∪ {p'};
24                 else Q.add(u);
25 return a set {v|v ∈ S ∧ v is P-connected to q} found by BSL;
```

During this process, whenever we find an edge (v, u) matched with i-th edge of $\mathcal{P}$, if it does not have a label i, we attach it a label i and add u to Y; if it has a label i, we skip it directly. Note that initially, edges do not have labels. After finding Y, we find new vertices that are $\mathcal{P}$-neighbors of vertices in Y, in the next iteration. This process repeats until all the vertices are labelled. Clearly, each edge is accessed in constant times as we use labelling. As a result, finding a set S of vertices that are $\mathcal{P}$-connected to q takes liner time cost, which is bounded by $\mathcal{O}(\sum_{i=1}^{l} n_i \cdot d_{i,i+1})$.

Depth-first search with labelling (DSL). Recall that by definition, a $\mathbf{B}_k$ only requires that each of its vertices has least k $\mathcal{P}$-neighbors. Meanwhile, as shown in existing CS studies [15, 16, 35, 60], k is often not very large. Motivated by this observation, we propose to dynamically maintain up to k $\mathcal{P}$-neighbors for each vertex. Specifically, we first find up to k $\mathcal{P}$-neighbors for each vertex, and then iteratively remove vertices that do not satisfy the constraint of k. Since the removal of a vertex v will remove a $\mathcal{P}$-neighbor of v's $\mathcal{P}$-neighbor vertices, so we need to incrementally supply new $\mathcal{P}$-neighbors for v's $\mathcal{P}$-neighbors. To find these $\mathcal{P}$-neighbors incrementally, we propose the DSL strategy.

In specific, after finding a path p using depth-first search, we check each vertex of p and label it as "visited", if all its neighbors have been considered before. For example, consider an HIN in Figure 4 with v=a_1 and $\mathcal{P}$=$(APTPA)$. After finding paths $a_1 \rightarrow p_1 \rightarrow t_1 \rightarrow p_4 \rightarrow a_2$ and $a_1 \rightarrow p_1 \rightarrow t_1 \rightarrow p_4 \rightarrow a_3$, we will label a_2, a_3, and p_4 as visited. These labelled vertices will not be considered when finding the remaining paths. This ensures that each edge will be visited constance times. Thus, for each vertex, enumerating all its path instances takes $\mathcal{O}(d_{1,2} + \sum_{i=2}^{l} n_i \cdot d_{i,i+1})$ time.

Based on the BSL and DSL strategies above, we propose an advanced algorithm, denoted by `FastBCore`, shown in Algorithm 2. First, we use the BSL strategy to find a set S of all the vertices that are $\mathcal{P}$-connected to q (lines 1-11). Then, it finds up to k path instances for each vertex and collects vertices that do not have k paths into a queue Q (lines 12-14). Then, it iteratively removes a vertex



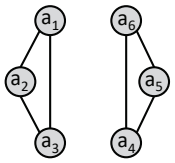


Figure 4: DFS with labelling. **Figure 5: Link relationships.**

v from Q, checks each $\mathcal{P}$-neighbor u, removes the path from u to v, tries to find a new path starting from u and ending at a vertex in S, and if it exists, then adds it to $\Psi[u]$; otherwise, adds v to Q (lines 15-24). After the loop, each vertex in S has k $\mathcal{P}$-neighbors.

Notice that since only k $\mathcal{P}$-neighbors are found for each vertex, we cannot find the maximal set of vertices that are $\mathcal{P}$-connected by using these $\mathcal{P}$-neighbors. For example, in Figure 1(a), let q=a_1, k=2, and $\mathcal{P}$=$(APTPA)$. After finding two $\mathcal{P}$-neighbors for each vertex, we may get two disconnected components as shown Figure 5, where each edge denotes a path instance. To remedy this issue, we reuse the BSL strategy to find the maximal set $\mathbf{B}_k$ of vertices that are in S and $\mathcal{P}$-connected to q, during which all the vertices with the target type are restricted to be from S.

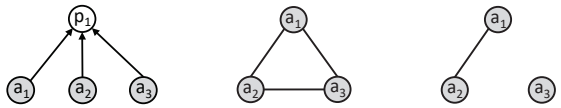
LEMMA 2. *The total time cost of* `FastBCore` *is* $\mathcal{O}(n_1 \cdot d_{1,2} + n_1 \sum_{i=2}^{l} n_i \cdot d_{i,i+1})$.

PROOF. Please see appendices of the technical report [75]. □

We remark that in practice, since the value of k is often not very large, `FastBCore` performs much faster than `HomBCore`.

3.2 Algorithms for Edge-disjoint (k, $\mathcal{P}$)-cores

Recall that $\mathbf{B}_k$ can be computed from the induced homogeneous graph $G_{\mathcal{P}}$. A natural question comes: Can we obtain $\mathbf{E}_k$ by computing k-core from $G_{\mathcal{P}}$? To get $G_{\mathcal{P}}$, a simple method is to adopt the one in `HomBCore` (i.e., if a vertex u is a $\mathcal{P}$-neighbor of a vertex v, then add an edge between them). Apparently, this method does not consider edge-disjoint paths and will result in incorrect $\mathbf{E}_k$.



(a) An example HIN (b) Method1 (c) Method2

Figure 6: Attempts of computing $\mathbf{E}_k$ from homogenous graphs.

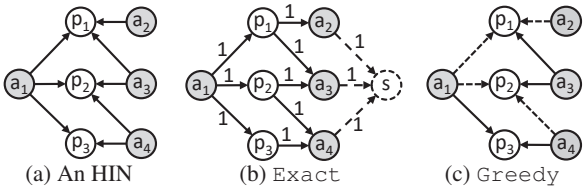
Another method is to build a homogenous graph in a greedy manner. Specifically, it first finds a path instance p of $\mathcal{P}$ which links two vertices (say u and v) in the HIN G, then removes all the edges of p from G, and finally adds an edge between u and v. The three steps above are repeated until there is no path instance of $\mathcal{P}$. This method, however, does not work either. Let $\Psi[v]$ denote the maximum set of edge-disjoint paths starting from v. The reasons are two-fold: First, $\Psi[v]$ may not be unique. Second, the link relationship may be *unsymmetric*, because for a specific path linking u and v, it may appear in $\Psi[v]$, but not in $\Psi[u]$. Example 2 illustrates these two methods where "Method1" and "Method2" denote them respectively. Therefore, computing $\mathbf{E}_k$ is more challenging than computing $\mathbf{B}_k$.

EXAMPLE 2. *Figure 6(a) shows an HIN G with $\mathcal{P}$=(APA). Clearly, a_1, a_2, and a_3 form an $\mathbf{E}_1$. By Method1, we get a graph in Figure 6(b), which is a 2-core. By Method2, we first find a path instance p="$a_1 \rightarrow p_1 \rightarrow a_2$", then move edges of p, and finally add an edge between a_1 and a_2. Since there is no other path instance of p in G, we get a graph in Figure 6(c), where a_3 is in 0-core. Thus, $\mathbf{E}_k$ cannot be computed from these induced homogeneous graphs.*

Next, we present the algorithms for computing $\beta(v, S)$, which is the e-degree of a vertex v regarding a set of vertices S with the type $\psi(q)$. Based on them, we propose two efficient query algorithms, where the first one peels vertices one by one, while the second one removes vertices in a batch manner.

3.2.1 Algorithms for Computing $\beta(v, S)$

We first introduce an exact algorithm, called `Exact`, based on the max-flow algorithm. Specifically, we first build a multipartite graph with $(l+1)$ partitions, also called a $(l+1)$-partite graph, and then get a flow network by using it. In the multipartite graph, the i-th partition contains all the i-th vertices in path instances of $\mathcal{P}$, and the edges from vertices in the i-th partition to vertices in the $(i+1)$-th partition are the i-th edges in all path instances of $\mathcal{P}$. To build the flow network, we first obtain all path instances starting from v and ending at vertices in S. Then, we build a multipartite graph. Next, we let v be the source vertex (which is in the first partition), and create a sink vertex s and link each vertex of the $(l+1)$-th partition to s. Finally, we add a capacity of 1 for each edge. We denote the above flow network construction method by `EBuilder`.



(a) An HIN (b) `Exact` (c) `Greedy`

Figure 7: Illustrating algorithms of computing $\beta(v, S)$.

Theorem 2 states that $\beta(v, S)$ equals to the capacity of the maximum flow of F. We illustrate this in Example 3.

THEOREM 2. *Given a vertex v and a flow network F=(V_F, E_F) built by* `EBuilder`, *$\beta(v, S)$ equals to the capacity of maximum flow from the source vertex to the sink vertex in F.*

PROOF. Please see appendices of the technical report [75]. □

EXAMPLE 3. *In Figure 7(a), let v=a_1, $\mathcal{P}$= (APA), and S=$\{a_1, a_2, a_3, a_4\}$. The flow network of a_1 is depicted in Figure 7(b). The capacity of the maximum flow is 3, so $\beta(v, S)$=3.*

By Theorem 2, we can use any existing max-flow algorithm to compute $\beta(v, S)$. In the literature, there are two well-known methods, namely Ford–Fulkerson method [12] and Orlin's method [51], where the former one is the most classic method and the latter one is the most recent method. Their time complexities are $\mathcal{O}(f \cdot |E_F|)$ and $\mathcal{O}(|V_F| \cdot |E_F|)$ respectively, where f is the capacity of maximum flow in the flow network F=(V_F, E_F). In our case, since $\beta(v, S)$ is at most $|S|$, we have f=$\beta(v, S) \leq |S| < |V_F|$. Thus, the Ford–Fulkerson method is faster than the Orlin's method on our flow network, and we adopt it to compute $\beta(v, S)$. The Ford–Fulkerson method computes the maximum flow by finding all the augmenting paths, where an augmenting path is a directed path staring from the source vertex and ending at the sink vertex.

The major limitation of `Exact` is its high computational cost, especially when $\beta(v, S)$ is very large. To improve efficiency, we propose an approximation algorithm, denoted by `Greedy`, as it works in a greedy and incremental manner. Specifically, given a vertex v, it first uses DFS to find a path instance p of $\mathcal{P}$, which starts with v and ends at a vertex $v' \in S$, and then removes all edges of p and v' from G. There two steps are repeated until there is not any path instance starting from v. Lemma 3 states that `Greedy` theoretically guarantees that the number of returned edge-disjoint path instances is at least $\frac{\beta(v,S)}{l}$. We illustrate it by Example 4.

LEMMA 3. *Given an HIN, a vertex v, and a set S of vertices with target type,* `Greedy` *achieves an approximation ratio of* $\frac{1}{l}$.

PROOF. Let $\Psi[v]$ be a maximal set of edge-disjoint path instances, which start with v and end at vertices in S. For any path instance p identified by `Greedy`, its edges appear in at most l path instances of $\Psi[v]$. This implies that after removing edges of p, it takes away at most l path instances from $\Psi[v]$. Hence, `Greedy` can find at least $\frac{\beta(v,S)}{l}$ path instances and the lemma holds. □

EXAMPLE 4. *Reconsider Example 3, where $\beta(a_1, S)$=3. Using* `Greedy`*, we may only find two path instances, i.e., $a_1 \rightarrow p_1 \rightarrow a_2$ and $a_1 \rightarrow p_2 \rightarrow a_4$ marked in dashed lines in Figure 7(c). Thus, the actual approximation ratio is $\frac{2}{3}$, which is larger than $\frac{1}{l}=\frac{1}{2}$.*

Since after finding a path instance all its edges are removed, the time cost of `Greedy` is linear to the size of the sub-HIN, which is induced by all the path instances starting with v. As a result, its time complexity could be bounded by $\mathcal{O}(|E_F|)$, where $|E_F|$ is the number of edges in the flow network F=(V_F, E_F) built by `EBuilder`. As discussed above, `Exact` has a time complexity of $\mathcal{O}(f \cdot |E_F|)$, where f could be up to $\frac{|E_F|}{l}$ in the worst case. Therefore, `Greedy` runs much faster than `Exact`.

Recall that the Ford–Fulkerson method [12] computes the maximum flow by iteratively finding augmenting paths. In our flow network, by concatenating each edge-disjoint path p with the sink vertex s, we can obtain an augmenting path from the source vertex to the sink vertex. Thus, to compute the exact $\beta(v, S)$, we can first run `Greedy` to get a set of edge-disjoint path instances, then mark them as augmenting paths, and finally find remaining augmenting paths using `Exact`. Note that an augmenting path is not necessary to correspond to a specific path instance of $\mathcal{P}$.

3.2.2 A Lazy Peeling-Based Algorithm

Inspired by the peeling paradigm of k-core computation [5], we propose an algorithm of computing $\mathbf{E}_k$ by iteratively removing vertices whose $\beta(v, S)$ values are less than k, where initially S contains all the vertices with the target type. After removing a vertex v, we need to decrease $\beta(u, S)$, where u is a $\mathcal{P}$-neighbor of v. However, the step of decreasing $\beta(u, S)$ is non-trivial, because it may remain unchanged. For example, consider the HIN in Figure 6(a) with v=a_1 and S=$\{a_1, a_2, a_3\}$. Initially, we have $\beta(a_1, S)$=1. After removing a_2 or a_3, we still have $\beta(a_1, S)$=1. To tackle this issue, a naive method is to recompute $\beta(u, S)$ for each $\mathcal{P}$-neighbor u of v by `Exact`. This, however, is very costly since the number of $\mathcal{P}$-neighbors is large and running `Exact` is also expensive. To alleviate this issue, we propose a *lazy peeling* strategy to postpone running `Exact` as late as possible, by relying on a key observation — after removing v, $\beta(u, S)$ decreases by at most 1.

Specifically, initially for each vertex $v \in S$, we compute an approximate $\beta(v, S)$ using `Greedy`. Then, we maintain a queue for keeping vertices whose current $\beta(v, S)$ values are less than k. Whenever we dequeue a vertex v, we compute the exact value of $\beta(v, S)$ using `Exact`. After that, if $\beta(v, S)<k$, we remove it and decrease $\beta(u, S)$ by 1 for each $\mathcal{P}$-neighbor u directly. If the updated $\beta(u, S)<k$, we add it into the queue. In other words, we do not update $\beta(u, S)$ precisely, but maintain a lower bound of $\beta(v, S)$. This will postpone running `Exact` until $\beta(u, S)<k$.

Based on discussions above, we develop an algorithm, denoted by `LazyECore`, shown in Algorithm 3. First, by Theorem 1, since $\mathbf{E}_k \subseteq \mathbf{B}_k$, it computes $\mathbf{B}_k$ (line 1). Then, it initializes a queue Q and an array $b[\,]$ for keeping $\beta(v, S)$ (line 2). Next, it computes $\beta(v, S)$ using `Greedy` for each vertex $v \in S$ and collects v if $\beta(v, S)<k$ (lines 3-5). In the loop (lines 6-18), it removes vertices one by one. Specifically, it first dequeues a vertex v. If $b[v]<\frac{k}{l}$, v can be removed directly by Corollary 1 (lines 9-10); otherwise,

Algorithm 3: The algorithm: `LazyECore`.

```
   Input: G, q, P, k;
   Output: E_k;
1  S ← run FastBCore to obtain the set B_k;
2  Q ← ∅, b[ ] ← ∅;
3  for each vertex v ∈ S do
4      b[v] ←Greedy(v, S);
5      if b[v]<k then Q.add(v);
6  while |Q|>0 do
7      v ← Q.poll();
8      η ←false;                    ▷ a variable indicating whether to delete v
9      if b[v]< k/l then
10         η ← true;
11     else
12         b[v] ←Exact(v, S);
13         if b[v]<k then η ← true;
14     if η is true then
15         S ← S\{v}, U ← {u|u ∈ S and u is a P-neighbor of v};
16         for each vertex u ∈ U do
17             b[u] ← b[u] − 1;          ▷ decrease by 1 directly
18             if b[u]<k then Q.add(u);
19 return a set {v|v ∈ S ∧ v is P-connected to q} found by BSL;
```

it invokes `Exact` (lines 11-13). If v can be deleted, it updates S, decreases $\beta(u, S)$ by 1 for each $\mathcal{P}$-neighbor u, and adds u into Q if $b[u]<k$ (lines 14-18). Finally, it returns $\mathbf{E}_k$ (line 19).

COROLLARY 1. *Given an HIN, a vertex v, and a set S of vertices with target type, if* `Greedy` *cannot find up to $\frac{k}{l}$ path instances, then we have $\beta(v, S)<k$.*

PROOF. The conclusion directly follows Lemma 3. □

LEMMA 4. `LazyECore` *completes in $\mathcal{O}(\sigma_1(n_1 \cdot d_{1,2} + n_1 \cdot \sum_{i=2}^{l} n_i \cdot d_{i,i+1}))$ time, where σ_1 ($\sigma_1<{n_1}^2$) is the total times of invoking* `Exact`.

PROOF. Please see appendices of the technical report [75]. □

3.2.3 A Batch Peeling-Based Algorithm

In this section, we develop another algorithm, which borrows the idea of `FastBCore` and removes vertices whose core numbers are less than k in a batch manner. We denote it by `BatchECore`. Recall that `FastBCore` sequentially removes vertices that are not in $\mathbf{B}_k$, during which we maintain a set of k path instances for each vertex dynamically. Since `Greedy` is able to find edge-disjoint path instances incrementally, we can also dynamically maintain a set of k edge-disjoint paths for each vertex.

Besides, instead of peeling vertices one by one, we propose a *batch peeling* strategy. Specifically, whenever we find a set of vertices that are not in $\mathbf{E}_k$, we remove all its vertices and update the values of $\beta(v, S)$ for their $\mathcal{P}$-neighbors in a collective manner. Compared to peeling one by one, it saves much computational cost, since `Exact` is invoked frequently for vertices whose $\beta(v, S)$ values are close to k. Note that the batch peeling cannot be used in `LazyECore`, since it cannot determine whether to remove a vertex until `Exact` is invoked. For `HomBCore`, the batch peeling cannot improve efficiency either, because after removing a vertex v, it enumerates v's $\mathcal{P}$-neighbors and supplies new path instances if needed, so we do not use batch peeling in `HomBCore`.

Algorithm 4 presents `BatchECore`. First, it collects all the vertices that are $\mathcal{P}$-connected to q by BSL, and initializes a set T and an array $\Psi[\,]$ (lines 1-2). Then, for each vertex $v \in S$, it finds a set $\Psi[v]$ of up to k edge-disjoint paths by `Greedy` (lines 3-4). By Corollary 1, if $|\Psi[v]|<\frac{k}{l}$, it puts v in T as it can be removed (lines 5-6); if $|\Psi[v]|<k$, it invokes `Exact` and puts it into T if it can be

Algorithm 4: The algorithm: `BatchECore`.

```
   Input: G, q, P, k;
   Output: E_k;
1  S ← find vertices that are P-connected to q by BSL;
2  T ← ∅, Ψ[ ] ← ∅;
3  for each vertex v ∈ S do
4      Ψ[v] ← find up to k edge-disjoint path instances by Greedy;
5      if |Ψ[v]|<k/l then
6          T.add(v);
7      else if |Ψ[v]|<k then
8          Ψ[v] ←Exact(v, S, Ψ[v]);
9          if |Ψ[v]|<k then T.add(v);
10 while |T|>0 do
11     T' ← ∅;  ▷ a set keeping vertices to be removed in next iteration
12     U←{u|u ∈ S\T and u is a P-neighbor of v where v ∈ T};
13     for each vertex u ∈ U do
14         for each path p ∈ Ψ[u] do
15             if p has an ending vertex in T then Ψ[u] ← Ψ[u]\{p};
16         supply new path instances using Greedy until |Ψ[u]|=k;
17         if |Ψ[u]|<k then
18             Ψ[u] ←Exact(v, S, Ψ[u]);
19             if |Ψ[u]|<k then T'.add(u);
20     S ← S\T, T ← T';
21 return a set {v|v ∈ S ∧ v is P-connected to q} found by BSL;
```

removed (lines 7-9). Next, in the loop (lines 10-20), it removes all the vertices in T, supplies new path instances for their $\mathcal{P}$-neighbors if needed, and finds a new set T' of vertices that can be removed in the next iteration. The batch peeling is repeated until no vertex can be removed. Finally, it returns $\mathbf{E}_k$ (line 21).

LEMMA 5. `BatchECore` *completes in* $\mathcal{O}(\sigma_2(n_1 \cdot d_{1,2} + n_1 \cdot \sum_{i=2}^{l} n_i \cdot d_{i,i+1}))$ *time, where* σ_2 ($\sigma_2 < {n_1}^2$) *is the total times of invoking* `Exact`.

PROOF. Please see appendices of the technical report [75]. □

3.3 Algorithms for Vertex-disjoint (k, $\mathcal{P}$)-cores

In this section, we focus on computing $\mathbf{V}_k$. Similar to $\mathbf{E}_k$, we cannot compute $\mathbf{V}_k$ directly from the induced homogeneous graphs, so we have to iteratively remove vertices that do not have sufficient vertex-disjoint paths. In the following, we first discuss how to compute $\gamma(v, S)$, and then show that the algorithms of computing $\mathbf{E}_k$ can be easily adapted for computing $\mathbf{V}_k$.

To compute $\gamma(v, S)$, we extend the exact algorithm `Exact` and approximation algorithm `Greedy`. For `Exact`, we build a new flow network by slightly modifying the flow network in `Exact`. In specific, after building a flow network F=(V_F, E_F) by `EBuilder`, for each intermediate vertex $g \in F$ between source and sink vertices, we split it as two vertices, say g^+ and g^-, such that the in-edges of g are connected to g^+, g^- is connected to the out-edges of g, and g^+ is connected to g^- where its capacity is set to 1. We denote this flow network construction method by `VBuilder`. It is easy to observe that $\gamma(v, S)$ equals to the capacity of the maximum flow in the modified network. We illustrate this by Example 5.

EXAMPLE 5. *Reconsider Example 3, where the flow network built by* `EBuilder` *is in Figure 7(b). By* `VBuilder`*, we can get a new flow network shown in Figure 8 and have* $\gamma(a_1, S)$=*3.*

Recall that `Greedy` iteratively finds path instances of $\mathcal{P}$ and after finding a path instance p, it removes all the *edges* of p. To compute $\gamma(v, S)$, we can follow the same steps, but replace the step of removing all the edges as removing all the *vertices*. Clearly, the adapted algorithm still achieves an approximation ratio of $\frac{1}{l}$.

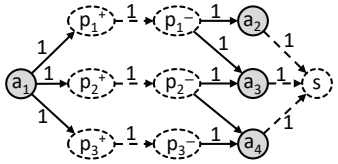


Figure 8: A new flow network built for computing $\gamma(v, S)$.

To compute $\mathbf{V}_k$, we extend `LazyECore` and `BatchECore` by simply replacing the step of computing $\beta(v, S)$ as computing $\gamma(v, S)$ using the algorithms above. We denote the extended algorithms by `LazyVCore` and `BatchVCore` respectively. It is easy to see that their time complexities are at most c ($c \leq 4$) times larger than those of `LazyECore` and `BatchECore` respectively, since the flow networks built by `VBuilder` are at most twice larger than those of `EBuilder` and `Exact` takes $\mathcal{O}(|E_F|^2/l)$ time.

4. INDEX-BASED ALGORITHMS

Although online algorithms above are fast, they may be inefficient when the queries are executed frequently. We further improve the efficiency by developing a novel space-efficient index, which allows the query to be completed in optimal query time cost. It relies on two key observations: First, the meta-paths frequently used in practice are often with limited lengths [48,58,62,64], since long meta-paths result in weak relationships [62]. Second, not all the meta-paths are meaningful. For example, by the schema of DBLP, we may obtain a meta-path $\mathcal{P}$=(VPV), but in practice a paper often can only be published in a single venue, so there are few or on path instances of $\mathcal{P}$. Since the number of meaningful meta-paths is often limited, we can pre-compute their cores and organize them into a compact structure, called `CoreIndex`. Note that these meta-paths can be obtained from domain experts or discovery algorithms [48]. In the following, we first give an overview of `CoreIndex` and then present the index construction algorithm.

4.1 Index Overview

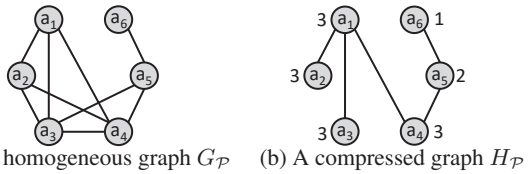
Given a set Λ of meta-paths and a specific core model, a simple method of building the index is to precompute and keep all the (k, $\mathcal{P}$)-cores offline. However, it is very costly due to its high space cost, i.e., $\mathcal{O}(\sum_{i=1}^{|\Lambda|} {t_i}^2)$, where t_i is the number of vertices with the type linked by $\mathcal{P}_i$. To alleviate this issue, we propose a space-efficient index structure, called `CoreIndex`. It organizes all the cores in a graph by carefully considering the *nested relationship* and $\mathcal{P}$*-connected relationship*, where the former one means that for any (k+1, $\mathcal{P}$)-core, there exists a (k, $\mathcal{P}$)-core containing it by Proposition 1, and latter one means that for a specific pair of k and $\mathcal{P}$, multiple (k, $\mathcal{P}$)-cores that are not $\mathcal{P}$-connected may exist.

Now we illustrate how `CoreIndex` organizes all the cores for a meta-path $\mathcal{P}$. Let S be the set of vertices with target type. We build a compressed labelled undirected graph $H_\mathcal{P}$ such that:

- for each node [1] $v \in H_\mathcal{P}$, it corresponds to a vertex $v' \in S$;
- for each node $v \in H_\mathcal{P}$, it has an associated value $\tau(v)$, which denotes the core number of its corresponding vertex $v' \in S$;
- for each pair of nodes u, $v \in H_\mathcal{P}$, if $\tau(u) \geq \tau(v)$, then there is only one path of nodes linking them in $H_\mathcal{P}$, where the associated value of each node in the path is at least $\tau(v)$.

Clearly, each connected component of $H_\mathcal{P}$ can be considered as a tree structure, so it takes linear space for a meta-path, and the overall index is space efficient as stated by Lemma 6. To answer a CSH query, we can simply find a set $\mathcal{C}$ of nodes, which are connected with q by nodes with core numbers of k or more, which takes the optimal time cost, i.e., $\mathcal{O}(|\mathcal{C}|)$. Example 6 illustrates this.

[1] We use "node" to mean "node of `CoreIndex`" in this paper.



(a) A homogeneous graph $G_{\mathcal{P}}$ (b) A compressed graph $H_{\mathcal{P}}$

Figure 9: Illustrating the `CoreIndex`.

LEMMA 6. *Given an HIN G=(V, E) and a set Λ of meta-paths, `CoreIndex` takes $\mathcal{O}(\sum_{i=1}^{|\Lambda|} t_i)$ space, where t_i is the total number of vertices with the type linked by $\mathcal{P}_i$.*

PROOF. The lemma directly follows the observation. □

EXAMPLE 6. *Consider the basic (k, $\mathcal{P}$)-core. Suppose the induced homogeneous graph $G_{\mathcal{P}}$ for vertices with the target type is depicted in Figure 9(a). Then, we can build $H_{\mathcal{P}}$ as shown in Figure 9(b). Let q=a_1 and k=2. We obtain a community $\{a_1, \cdots, a_5\}$.*

4.2 Index Construction

To build `CoreIndex`, we create a graph $H_{\mathcal{P}}$ for each meta-path $\mathcal{P}\in\Lambda$ and then concatenate the adjacency list of $H_{\mathcal{P}}$. Algorithm 5 presents the steps. We first compute the core number of each vertex, also called *core decomposition*, and get a $list$ of vertices (which are sorted in descending order of core numbers) (lines 3-4). Then, for each vertex v', we create a node v. After that, we find a set U of $\mathcal{P}$-neighbors of v', and for each vertex u' of U, if there is no path from node v to node u, we create an edge (v, u) (lines 4-9). Note that to check connectivity between nodes, we use the standard union-find data structure [1, 13], where MAKESET creates a node, FIND finds the root of a node, and UNION makes two nodes have the same root.

The index construction method above relies on a key step of core decomposition (line 3). To do this, we extend online algorithms in Section 3. For a specific meta-path $\mathcal{P}$ under a core model, we incrementally compute all the (k, $\mathcal{P}$)-cores, where k increases from 0 to its maximum value. Note that once we have computed all the (k, $\mathcal{P}$)-cores, we compute (k+1, $\mathcal{P}$)-cores from these (k, $\mathcal{P}$)-cores by exploiting their nested relationships. Detailed pseudocodes are presented in the technical report [75].

Algorithm 5: Index construction algorithm.

```
   Input: G, Λ, a core model Γ;
   Output: The CoreIndex ϒ;
1  ϒ ← ∅;
2  for each meta-path P ∈ Λ do
3      list ← perform core decomposition for P under the model Γ;
4      for i ← |list| to 1 do
5          let v' be list[i], create a node v for v', invoke MAKESET(v);
6          U←{u'|u' is a P-neighbor of v', and τ(u)≥τ(v)};
7          for each vertex u' ∈ U do
8              r(v) ← FIND(v), r(u) ←FIND(u);
9              if r(v) ≠ r(u) then UNION(v, u), create an edge (v, u);
10     merge the adjacency list of the graph H_P into ϒ;
11 return ϒ;
```

LEMMA 7. *Given a set Λ of meta-paths and a core model, Algorithm 5 takes $\mathcal{O}(\sum_{i=1}^{|\Lambda|}(\Delta_i + t_i \cdot d_i \cdot \alpha(t_i))$ time, where Δ_i is the cost of core decomposition for $\mathcal{P}_i$, t_i is the number of vertices with the type linked by $\mathcal{P}_i$, d_i is the maximum number of $\mathcal{P}$-neighbors for vertices with i-th vertex type, and $\alpha(t_i)$ is the inverse Ackermann function (less than 5 for all practical values of t_i).*

PROOF. Given a meta-path $\mathcal{P}_i$, invoking each of FIND, UNION, and MAKESET functions takes at most $\mathcal{O}(\alpha(t_i))$ time [1]. For each pair of $\mathcal{P}_i$-connected vertices, we only invoke FIND and UNION constant times, so it takes $\mathcal{O}(t_i \cdot d_i \cdot \alpha(t_i))$ and Lemma 7 holds. □

5. EXPERIMENTS

We now present the experimental results. Section 5.1 discusses the setup. We discuss the results in Sections 5.2 and 5.3.

5.1 Setup

Table 5: Datasets used in our experiments.

Dataset	Vertices	Edges	Vertex types	Edge types	Meta-paths
Foursquare	43,199	405,476	5	4	20
DBLP	682,819	1,951,209	4	3	12
IMDB	4,467,806	7,597,591	4	3	12
DBpedia	5,900,558	17,961,887	413	637	1,000
Freebase	14,420,276	53,306,405	55	389	1,000

Datasets. We use five real datasets: Foursquare [2], DBLP [3], IMDB [4], DBpedia [5], and Freebase [6]. Their statistics such as the numbers of vertices, edges, vertex types, and edge types are reported in Table 5. The first three datasets are with simple star-schemas (the schemas of Foursquare and IMDB are shown in appendices of the technical report [75]), while the rest two datasets are with rich schemas. Foursquare contains check-in records in US, which has five types of vertices (venues, cities, venue categories, users, and dates). DBLP includes publication records in computer science areas, and the vertex types are authors, papers, venues, and topics. IMDB contains the movie rating records since 2000, and it has four types of vertices (actors, directors, writers, and movies). DBpedia contains the data extracted from wikipedia infoboxes using the mapping-based extraction (object properties only). Freebase contains all the entities and relations in the music domain.

Queries. For each dataset, we collect a set of meta-paths and its size is reported in Table 5. Note that in line with existing works [38, 62], we focus on meta-paths with lengths at most four. For the first three datasets, we collect all the possible meta-paths; for the rest two datasets, we use the top-1000 meta-paths with the highest frequencies. We generate 200 queries for each dataset. To generate a query, we randomly select a meta-path and then select a vertex with core number of 6 or more, which ensures that there is a meaningful community containing the query vertex, similar to previous studies [24]. By default, we set the value of k to 6. In the results reported in the following, each data point is the average result for these 200 queries. We implement all the algorithms in Java, and run experiments on a machine having an Intel(R) Xeon(R) 3.40GHz CPU and 32GB of memory, with Ubuntu installed.

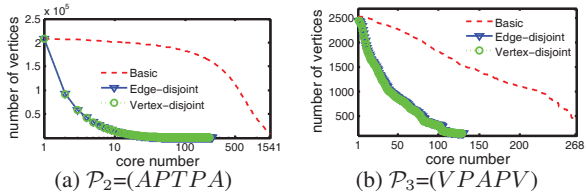
5.2 Effectiveness Evaluation

5.2.1 Core Analysis

To analyze the three kinds of proposed (k, $\mathcal{P}$)-cores, we examine the size distribution of (k, $\mathcal{P}$)-cores, where k ranges from 0 to its maximum value. Due to the space limitation, we only show results of two meta-paths, i.e., $\mathcal{P}_2$ and $\mathcal{P}_3$ (see Figure 2 and Table 2), on the DBLP network in Figure 10, where each data point (x, y) means that the corresponding (x, $\mathcal{P}$)-core has y vertices.

From Figure 10, we see that the size distributions of $\mathbf{B}_k$ are different from those of $\mathbf{E}_k$ and $\mathbf{V}_k$, while the size distributions of $\mathbf{E}_k$ and $\mathbf{V}_k$ are very similar. The reason is that the disjoint core models impose stronger cohesiveness constraints on path instances, making the e-degrees and v-degrees smaller than b-degrees. Meanwhile, for $\mathcal{P}_2$, the maximum core number of $\mathbf{B}_k$ is over 6 times larger

[2]https://sites.google.com/site/yangdingqi/home/foursquare-dataset
[3]http://dblp.uni-trier.de/xml/
[4]https://www.imdb.com/interfaces/
[5]https://wiki.dbpedia.org/Datasets
[6]http://freebase-easy.cs.uni-freiburg.de/dump/



(a) $\mathcal{P}_2$=$(APTPA)$ (b) $\mathcal{P}_3$=$(VPAPV)$

Figure 10: Core size distributions.

than those of $\mathbf{E}_k$ and $\mathbf{V}_k$, while for $\mathcal{P}_3$, their maximum core numbers tend to be much closer. This is because for $\mathcal{P}_2$, although each author has a high b-degree, her e-degree and v-degree are small due to the limited number of research topics; for $\mathcal{P}_3$, the number of venues is smaller than the numbers of paper and authors, and each paper is often published in a single venue, making the differences between b-degrees and e-degrees small.

5.2.2 *Community Quality Analysis*

We analyze the quality of communities from following aspects:

1 Examination of $\mathbf{B}_k$. We first analyze the e-degree and v-degree distributions of vertices in communities of $\mathbf{B}_k$. Specifically, for each dataset, we run the 200 CS queries using `FastBCore` (k=6), and for each community, we count the percentages of vertices, whose e-degrees and v-degrees are 1, 2, . . . , 20 respectively. Due to the space limitation, we only report the average percentage values on two datasets in Figures 11(a)-(b). Clearly, the percentages of vertices whose e-degrees and v-degrees are 1 and 2 are very high. For example, the percentage of vertices with e-degrees of 1 on IMDB is over 30%. Thus, although each community member is $\mathcal{P}$-connected to at least k other members, it is weakly engaged in the community, because it can be isolated from the community after removing only one edge. Moreover, the total percentages of vertices whose e-degrees and v-degrees are less than 6 are over 44% on all datasets. In contrast, the communities of $\mathbf{E}_k$ and $\mathbf{V}_k$ do not suffer from such issues, so they achieve stronger cohesiveness.

2 Closeness of communities. To measure the closeness of communities, a commonly-used metric is the diameter [37], or the largest shortest distance between any pair of vertices in the subgraph of the community. To adapt it for communities in HINs, we redefine the "distance" as path-constrained distance, i.e., *$\mathcal{P}$-distance*; that is, the $\mathcal{P}$-distance between two vertices linked by an instance of the meta-path $\mathcal{P}$ is 1. The average diameters for communities of each core model are depicted in Figure 12. Clearly, the communities of $\mathbf{E}_k$ and $\mathbf{V}_k$ have smaller diameters than those of $\mathbf{B}_k$ on all the datasets, which means that their vertices tend to have closer relationships.

3 Density of link relationships. Conventionally, the density of a graph is defined as the number of edges over the number of vertices [74, 76]. To adapt it for communities in HINs, we redefine it as the number of vertex pairs that are $\mathcal{P}$-connected over the number of vertices (here, all the vertices are with the target type). The average densities for communities of each core model are depicted in Figure 13. We observe that the communities based on the basic $(k, \mathcal{P})$-core have the lowest densities, while communities of vertex-disjoint core models have the highest densities. Thus, vertices in $\mathbf{E}_k$ and $\mathbf{V}_k$ tend to be more densely connected to each other.

4 Similarity of community members. We have measured the similarity of community members by PathSim [62]. Specifically, we first find communities of $\mathbf{B}_k$, $\mathbf{E}_k$, and $\mathbf{V}_k$, and then compute the PathSim value for each pair of vertices in these communities. Figure 14 shows the average PathSim values on three datasets. Clearly, communities of $\mathbf{E}_k$ and $\mathbf{V}_k$ achieve higher similarity values than those of $\mathbf{B}_k$, so their members are more similar to each other.

5 F_1-scores. In line with existing CS studies [35, 36], we test CSH queries on a small DBLP dataset (denoted by S-DBLP) with ground-truth communities. Specifically, we build S-DBLP by using publications of major conferences in four research areas (i.e., database, security, graphics, and communications) in 2015 and 2016, with 500 authors, 15 conferences, 7909 papers, and 496 topics. We classify authors and conferences into four communities according to the areas they belong to, respectively. After that, for each author and conference, we query its communities using the three core models with meta-paths $\mathcal{P}_2$ and $\mathcal{P}_3$ (see Section 1) respectively, and then compute the average F_1-score values [35], which is reported in Table 6. For both meta-paths, we observe that $\mathbf{V}_k$ achieves the highest F_1-score values while $\mathbf{B}_k$ has the smallest F_1-score values. Thus, $\mathbf{V}_k$ and $\mathbf{E}_k$ can better find the ground-truth communities.

Table 6: F_1-score values on S-DBLP dataset.

k	$\mathcal{P}_2$=$(APTPA)$			$\mathcal{P}_3$=$(VPAPV)$		
	$\mathbf{B}_k$	$\mathbf{E}_k$	$\mathbf{V}_k$	$\mathbf{B}_k$	$\mathbf{E}_k$	$\mathbf{V}_k$
3	0.542	0.544	0.544	0.474	0.474	0.474
4	0.542	0.544	0.544	0.474	0.474	0.533
5	0.542	0.543	0.543	0.554	1.0	1.0

6 A Case Study. We perform two CSH queries using $\mathbf{B}_k$ and $\mathbf{E}_k$ on S-DBLP. In the first query, q=Prof. Xuemin Lin (a researcher in the area of graph database), $\mathcal{P}$=(APA), and k=5. As shown in Table 2, the two communities contain six researchers who collaborated intensively, but the first one has 13 additional authors. With an in-depth investigation, we find that these authors were Ph.D. students and master students, and they co-authored only one or two papers with others. In the second query, q=SIGMOD conference, $\mathcal{P}$=$(VPAPV)$, and k=5. From Table 2, we see that the community of $\mathbf{B}_k$ contains six conferences in database area and five conferences in security areas, while the community of $\mathbf{E}_k$ only consists of conferences in the database area, which are highly related to the query conference. Therefore, we conclude that in terms of community cohesiveness, $\mathbf{E}_k$ is more cohesive than $\mathbf{B}_k$.

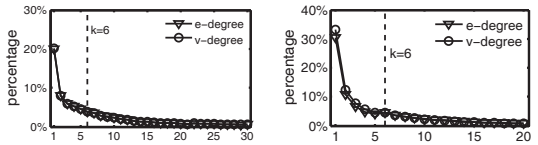
In addition, we have developed a baseline for finding communities with both *high cohesiveness* and *high similarity*. For the cohesiveness, it uses $\mathbf{B}_k$ and for similarity, it maximizes the PathSim [62] values for the vertex pairs that are $\mathcal{P}$-connected within the community. However, our experiments show that the communities of $\mathbf{E}_k$ and $\mathbf{V}_k$ achieve higher quality than its communities. For details, please refer to the appendices of the technical report [75].

5.3 Efficiency Evaluation

1 Online algorithms. The efficiency results of online algorithms by varying k are reported in Figure 15. As shown in Figures 15(a)-15(e), `FastBCore` is consistently faster than `HomBCore`, since for each vertex with the target type, `HomBCore` finds all its $\mathcal{P}$-neighbors, while `FastBCore` only finds a small number of them. Meanwhile, as k becomes larger, the running time of `FastBCore` increases since a larger k means finding more $\mathcal{P}$-neighbors. The running time of `HomBCore` remains almost stable since the main overhead comes from building the homogeneous graph.

From Figures 15(f)-15(j), we see that `BatchECore` is consistently faster than `LazyECore`. This is because the total times of invoking `Exact` in `BatchECore` is often much smaller than that in `LazyECore`, i.e., σ_2<σ_1. For example, on Foursquare, σ_1 is averagely over an order of magnitude larger than σ_2. Meanwhile, as the value of k grows, `BatchECore`'s running time increases slightly while `LazyECore`'s running time decreases slightly. This is because as k becomes larger, `BatchECore` needs to find more path instances for vertices with the target type, while `LazyVECore` also has to find more path instances, but less cost on running `Exact` of computing $\beta(v, S)$, making the overall time become smaller. Similarly, for $\mathbf{V}_k$, we can observe such trends.

In addition, it is easy to observe that querying $\mathbf{B}_k$ takes the least time cost while computing $\mathbf{V}_k$ is the most time consuming one. The main reason is that computing k vertex-disjoint paths is more



(a) DBLP (b) IMDB

Figure 11: E-degree and v-degree distributions in $\mathbf{B}_k$.

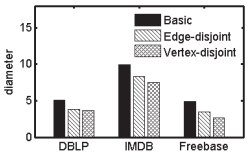


Figure 12: Diameter.

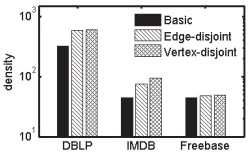


Figure 13: Density.

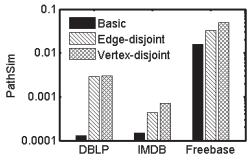
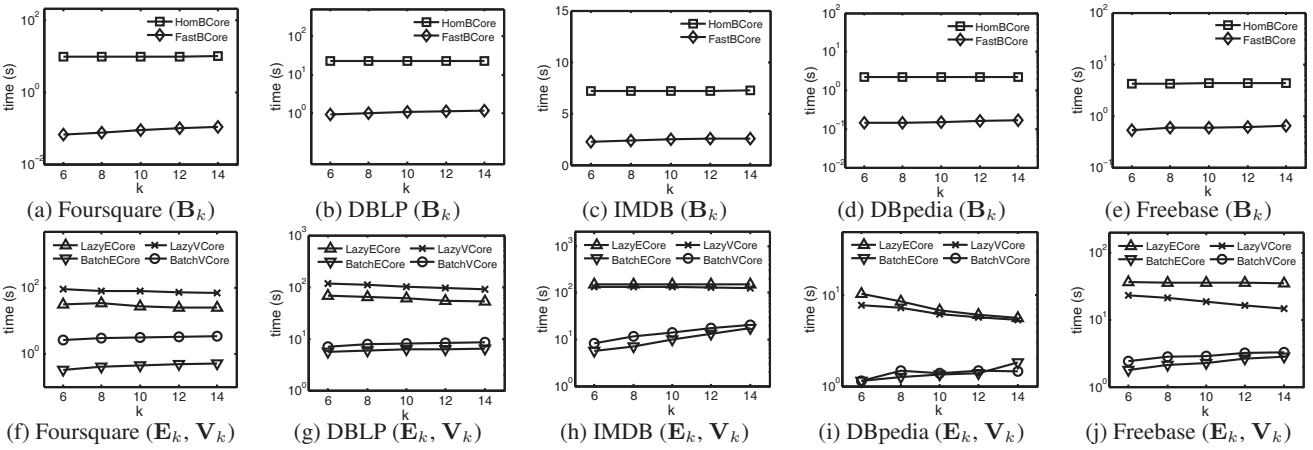


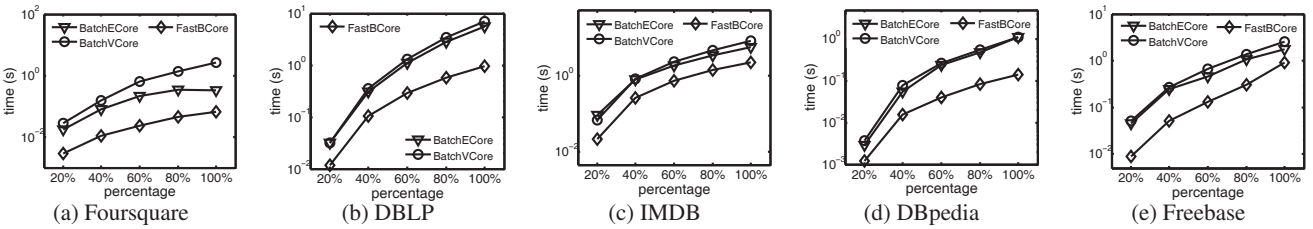
Figure 14: PathSim.



(a) Foursquare ($\mathbf{B}_k$) (b) DBLP ($\mathbf{B}_k$) (c) IMDB ($\mathbf{B}_k$) (d) DBpedia ($\mathbf{B}_k$) (e) Freebase ($\mathbf{B}_k$)

(f) Foursquare ($\mathbf{E}_k$, $\mathbf{V}_k$) (g) DBLP ($\mathbf{E}_k$, $\mathbf{V}_k$) (h) IMDB ($\mathbf{E}_k$, $\mathbf{V}_k$) (i) DBpedia ($\mathbf{E}_k$, $\mathbf{V}_k$) (j) Freebase ($\mathbf{E}_k$, $\mathbf{V}_k$)

Figure 15: Efficiency results of online query algorithms.



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

Figure 16: Scalability test for online query algorithms.

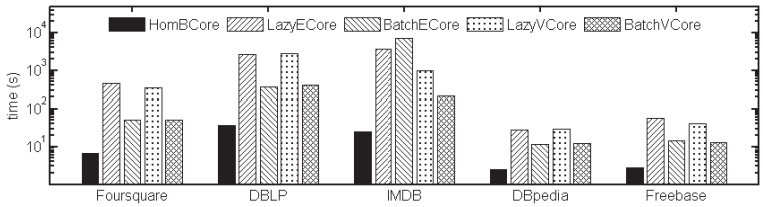


Figure 17: Efficiency of core decomposition.

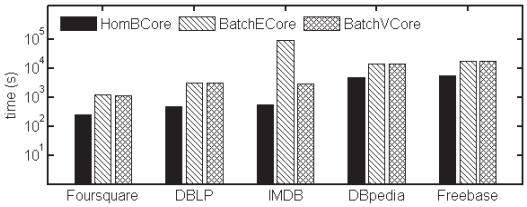
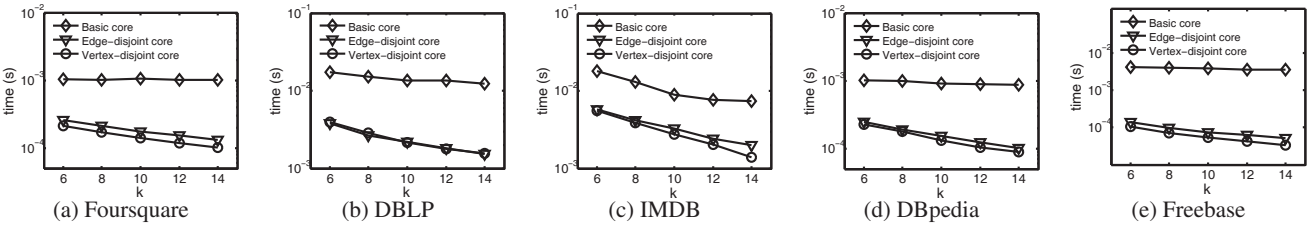


Figure 18: Efficiency of index construction.



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

Figure 19: Efficiency results of index-based queries.

expensive than finding k paths that do not need to satisfy such a constraint.

2 Scalability test. For each dataset, we randomly select 20%, 40%, 60%, 80%, and 100% vertices and obtain five subgraphs induced by these vertices respectively. Then, we run online CSH queries using the advanced algorithm for each core model, and report the efficiency results in Figure 16. We can see that generally, they scale well with the number of vertices. Moreover, their performance trends are similar to those discussed before.

3 Core decomposition. As discussed in Section 4.2, each online query algorithm can be extended for core decomposition. However, for the basic core model, both `FastBCore` and `HomBCore` need to find all $\mathcal{P}$-neighbors of each vertex with target type, but `FastBCore` needs extra cost to maintain k $\mathcal{P}$-neighbors dynamically, so `HomBCore` is a better option and we skip `FastBCore`.

For each dataset, we perform core decomposition using these five algorithms and report the average time cost of core decomposition for each meta-path in Figure 17. Clearly, decomposing $\mathbf{B}_k$ takes the least time cost since its core model is simpler than others. `BatchECore` is consistently faster than `LazyBCore` for decomposing $\mathbf{E}_k$, and similarly `BatchVCore` is faster than `LazyVCore`. Another interesting observation is that on almost all the datasets, $\mathbf{V}_k$ can be decomposed more efficiently than $\mathbf{E}_k$ (using their faster algorithms, i.e., `BatchECore` and `BatchVCore`). This is mainly because both of them decompose cores incrementally from k=0, but the maximum core number of $\mathbf{V}_k$ is smaller than that of $\mathbf{E}_k$, so `BatchVCore` takes less time cost on core decomposition.

4 Index construction. To build `CoreIndex`, we use the faster core decomposition algorithms for the three core models, namely `HomBCore`, `BatchECore`, and `BatchVCore`. For each dataset, we consider all the meta-paths in Table 5, and report the efficiency results in Figure 18. Generally, the time cost of index construction is in line with that of core decomposition; that is, building `CoreIndex` for $\mathbf{B}_k$ takes the least time cost, and the time cost of $\mathbf{V}_k$ is less than that of $\mathbf{E}_k$. The reason is that given the core number of each vertex, building the compressed graph of `CoreIndex` takes almost linear time cost, as discussed in Section 4.2.

5 Index-based queries. Figure 19 shows the efficiency results of index-based queries. Clearly, the index-based query algorithms are around 3 to 5 orders of magnitude faster than online query algorithms, because they take optimal query time cost. Besides, querying $\mathbf{V}_k$ is the fastest, and querying $\mathbf{E}_k$ is faster than querying $\mathbf{B}_k$, since $|\mathbf{V}_k| \leq |\mathbf{E}_k| \leq |\mathbf{V}_k|$ by Theorem 1.

6. RELATED WORK

The research on graphs has received much attention [17, 26, 29, 30, 43, 45–47, 52, 53, 55, 67, 72], and the problems of network community retrieval can generally be classified into community detection (CD) and community search (CS).

Community detection (CD). Earlier solutions [31, 49] employ link-based analysis to detect these communities. However, they mainly focus on homogeneous graphs. Some recent works [58, 59, 61, 63–65, 81] focus on generating clusters/communities in HINs. They can generally be classified into two groups according to vertex types in the communities. The first group [8, 59, 61, 65] focuses on detecting clusters, each of which contains objects with multiple types. The second group [63,64,81] aims to generate clusters of objects with a specific type. In [63], Sun et al. proposed an algorithm to generate clusters of a specific type of objects; in [64], a user-guided algorithm is developed to cluster objects of a target type; in [81], a social influence-based clustering algorithm is presented. Our work is more related to the second group as we find communities where vertices are of the same type. The main differences between this group and ours are three-fold. First, its solutions are generally costly, as they often detect all the communities from an entire HIN. Second, it is not clear how to adapt them for personalized CS. Third, the communities are difficult to interpret since there are various kinds of relationships among objects in the community.

Community search (CS). CS aims to query densely connected subgraphs containing a specific vertex in an "online" manner [15, 16, 26, 35, 37, 42, 60]. To measure the structure cohesiveness of a community, people often use some metrics [26], and the *minimum degree* metric is the most frequently used one; it requires that each vertex's degree is at least k within the community, which is also used in k-core [5, 6, 28, 57]. For example, in [60], Sozio et al. proposed to find the connected k-core containing the query vertex as the community; in [15], Cui et al. developed an efficient local search algorithm. Besides, some works [10, 21–25, 27, 41, 42, 68] also use the minimum degree metric to search communities from attributed graphs, where vertices are associated with attributes such as keywords [24] and locations [27, 71, 73]. Another group of CS works is based on the k-truss model [11, 80]. For example, in [35, 37], the k-truss-based community search is studied; in [9, 36], Huang et al. and Chen et al. studied CS using k-truss on attributed graphs. In addition, other well-known cohesiveness metrics, including k-clique [16, 78] and k-edge connected component [7, 33, 34], have also been used for CS. A comprehensive survey can be found in [26]. However, all these works focus on homogeneous graphs, and it is not clear how to adapt them for CS over HINs, calling for new solutions of CS over HINs. To the best of our knowledge, this paper is the first work of CS over HINs.

Disjoint paths. The disjoint paths have been widely used in graph clustering. In [3,56], a graph clustering algorithm was developed, which first computes the numbers of vertex- or edge-disjoint paths between pairs of vertices which are used to measure the similarities between vertex pairs, and then clusters the graph using these similarity values. In [66], another clustering algorithm is developed based on the concepts of k-component and k-block, which are the maximal subgraphs such that each pair of vertices among these subgraphs is joined by at least k edge- and vertex-disjoint paths, respectively. In [79], Zhang et al. developed a clustering algorithm which measures the similarity between vertices using disjoint paths. In [19], the authors partitioned a graph into edge-disjoint cycles and paths. However, all these works focus on homogeneous graphs, so they cannot be directly used for CS over HINs.

7. CONCLUSION

In this paper, we study the CSH problem, which aims to search a community for a query vertex in an HIN. To model the cohesiveness of a community with vertices of the same type, we adopt the well-known concept of meta-path and propose three kinds of core models by incorporating the meta-path. For each core model, we develop efficient online query algorithms. Moreover, we develop a compact index structure to further boost the query efficiency. Our experimental results show that the proposed solutions are effective and efficient for searching communities over large HINs.

In the future, we will try to use other cohesiveness metrics (e.g., k-truss [35, 36, 77] and k-clique [4]) for CS on HINs and bipartite graphs [44, 69, 70]. We will also study how to search communities with vertices of multiple types (e.g., a community contains both authors and topics in the DBLP network).

Acknowledgments. We would like to thank Prof. Maria E. Orlowska for her insightful suggestions on improving the paper. Wenjie Zhang was supported by PS53783, DP200101116 and DP180103096. Xuemin Lin was supported by NSFC61232006, 2018YFB1003504, U1636215, DP200101338, DP180103096, and DP170101628. Xin Cao was supported by ARC DE190100663.

8. REFERENCES

[1] https://en.wikipedia.org/wiki/Disjoint-set_data_structure.

[2] A. Amelio and C. Pizzuti. Overlapping community discovery methods: A survey. In *Social Networks: Analysis and Case Studies*, pages 105–125. Springer, 2014.

[3] H. Balakrishnan and N. Deo. Discovering communities in complex networks. In *Proceedings of the 44st Annual Southeast Regional Conference*, pages 280–285, 2006.

[4] X. Bao and L. Wang. A clique-based approach for co-location pattern mining. *Information Sciences*, 490:244–264, 2019.

[5] V. Batagelj and M. Zaversnik. An o(m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049*, 2003.

[6] F. Bonchi, A. Khan, and L. Severini. Distance-generalized core decomposition. In *SIGMOD*, pages 1006–1023, 2019.

[7] L. Chang, X. Lin, L. Qin, J. X. Yu, and W. Zhang. Index-based optimal algorithms for computing steiner components with maximum connectivity. In *SIGMOD*, pages 459–474. ACM, 2015.

[8] L. Chen, Y. Gao, Y. Zhang, C. S. Jensen, and B. Zheng. Efficient and incremental clustering algorithms on star-schema heterogeneous graphs. In *ICDE*, pages 256–267. IEEE, 2019.

[9] L. Chen, C. Liu, R. Zhou, J. Li, X. Yang, and B. Wang. Maximum co-located community search in large scale social networks. *PVLDB*, 11(9):1233–1246, 2018.

[10] Y. Chen, Y. Fang, R. Cheng, Y. Li, X. Chen, and J. Zhang. Exploring communities in large profiled graphs. *TKDE*, 31(8):1624–1629, 2018.

[11] J. Cohen. Trusses: Cohesive subgraphs for social network analysis. *National security agency technical report*, 16:3–1, 2008.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Section 26.2: The ford-fulkerson method. *Introduction to algorithms*, pages 651–664, 2001.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2009.

[14] M. Coscia, F. Giannotti, and D. Pedreschi. A classification for community discovery methods in complex networks. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 4(5):512–546, 2011.

[15] W. Cui, Y. Xiao, H. Wang, and W. Wang. Local search of communities in large graphs. In *SIGMOD*, pages 991–1002. ACM, 2014.

[16] W. Cui et al. Online search of overlapping communities. In *SIGMOD*, pages 277–288, 2013.

[17] B. Du, T. Xinyao, Z. Wang, L. Zhang, and D. Tao. Robust graph-based semisupervised learning for noisy labeled data via maximum correntropy criterion. *IEEE transactions on cybernetics*, 49(4):1440–1453, 2018.

[18] J. T. Dudley, T. Deshpande, and A. J. Butte. Exploiting drug–disease relationships for computational drug repositioning. *Briefings in bioinformatics*, 12(4):303–311, 2011.

[19] H. Enomoto. Graph partition problems into cycles and paths. *Discrete Mathematics*, 233(1):93 – 101, 2001.

[20] M. A. Erce et al. Interactions affected by arginine methylation in the yeast protein–protein interaction network. *Molecular & Cellular Proteomics*, 12(11):3184–3198, 2013.

[21] Y. Fang and R. Cheng. On attributed community search. In *International Workshop on Mobility Analytics for Spatio-temporal and Social Data of PVLDB 2017*, pages 1–21. Springer, 2017.

[22] Y. Fang, R. Cheng, Y. Chen, S. Luo, and J. Hu. Effective and efficient attributed community search. *The VLDB Journal*, 26(6):803–828, 2017.

[23] Y. Fang, R. Cheng, X. Li, S. Luo, and J. Hu. Effective community search over large spatial graphs. *PVLDB*, 10(6):709–720, 2017.

[24] Y. Fang, R. Cheng, S. Luo, and J. Hu. Effective community search for large attributed graphs. In *PVLDB*, pages 1233–1244, 2016.

[25] Y. Fang, R. Cheng, S. Luo, J. Hu, and K. Huang. C-explorer: Browsing communities in large graphs. *PVLDB*, 10(12):1885–1888, 2017.

[26] Y. Fang, X. Huang, L. Qin, Y. Zhang, W. Zhang, R. Cheng, and X. Lin. A survey of community search over big graphs. *The VLDB Journal*, 2019.

[27] Y. Fang, Z. Wang, R. Cheng, X. Li, S. Luo, J. Hu, and X. Chen. On spatial-aware community search. *TKDE*, 31(4):783–798, 2019.

[28] Y. Fang, Z. Wang, R. Cheng, H. Wang, and J. Hu. Effective and efficient community search over large directed graphs. *TKDE*, 31(11):2093–2107, 2019.

[29] Y. Fang, K. Yu, R. Cheng, L. V. Lakshmanan, and X. Lin. Efficient algorithms for densest subgraph discovery. *PVLDB*, 12(11):1719–1732, 2019.

[30] Y. Fang, H. Zhang, Y. Ye, and X. Li. Detecting hot topics from twitter: A multiview approach. *Journal of Information Science*, 40(5):578–593, 2014.

[31] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

[32] C. Giatsidis, D. M. Thilikos, and M. Vazirgiannis. D-cores: Measuring collaboration of directed graphs based on degeneracy. In *IEEE International Conference on Data Mining*, pages 201–210, Dec 2011.

[33] J. Hu, X. Wu, R. Cheng, S. Luo, and Y. Fang. Querying minimal steiner maximum-connected subgraphs in large graphs. In *CIKM*, pages 1241–1250. ACM, 2016.

[34] J. Hu, X. Wu, R. Cheng, S. Luo, and Y. Fang. On minimal steiner maximum-connected subgraph queries. *TKDE*, 29(11):2455–2469, 2017.

[35] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. In *SIGMOD*, pages 1311–1322, 2014.

[36] X. Huang and L. V. S. Lakshmanan. Attribute-driven community search. *PVLDB*, 10(9):949–960, May 2017.

[37] X. Huang, L. V. S. Lakshmanan, J. X. Yu, and H. Cheng. Approximate closest community search in networks. *PVLDB*, 9(4):276–287, 2015.

[38] Z. Huang, Y. Zheng, R. Cheng, Y. Sun, N. Mamoulis, and X. Li. Meta structure: Computing relevance in large heterogeneous information networks. In *KDD*, pages 1595–1604. ACM, 2016.

[39] B. S. Khan and M. A. Niazi. Network community detection: A review and visual survey. *arXiv preprint arXiv:1708.00977*, 2017.

[40] J. Leskovec, K. J. Lang, and M. Mahoney. Empirical comparison of algorithms for network community detection. In *WWW*, pages 631–640. ACM, 2010.

[41] R.-H. Li, L. Qin, F. Ye, J. X. Yu, X. Xiao, N. Xiao, and Z. Zheng. Skyline community search in multi-valued networks. In *SIGMOD*, pages 457–472. ACM, 2018.

[42] R.-H. Li, L. Qin, J. X. Yu, and R. Mao. Influential

community search in large networks. *PVLDB*, 8(5):509–520, 2015.

[43] Z. Li, Y. Fang, Q. Liu, J. Cheng, R. Cheng, and J. C. Lui. Walking in the cloud: Parallel simrank at scale. *PVLDB*, 9(1):24–35, 2015.

[44] B. Liu, L. Yuan, X. Lin, L. Qin, W. Zhang, and J. Zhou. Efficient (α, β)-core computation: An index-based approach. In *The World Wide Web Conference*, pages 1130–1141, 2019.

[45] F. Luo, B. Du, L. Zhang, L. Zhang, and D. Tao. Feature learning using spatial-spectral hypergraph discriminant analysis for hyperspectral image. *IEEE transactions on cybernetics*, 49(7):2406–2419, 2018.

[46] C. Ma, R. Cheng, L. V. Lakshmanan, T. Grubenmann, Y. Fang, and X. Li. Linc: a motif counting algorithm for uncertain graphs. *PVLDB*, 13(2):155–168, 2019.

[47] C. Ma, Y. Fang, R. Cheng, L. V. Lakshmanan, W. Zhang, and X. Lin. Efficient algorithms for densest subgraph discovery on large directed graphs. In *SIGMOD*. ACM, 2020.

[48] C. Meng, R. Cheng, S. Maniu, P. Senellart, and W. Zhang. Discovering meta-paths in large heterogeneous information networks. In *WWW*, pages 754–764, 2015.

[49] M. E. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026–113, 2004.

[50] T. Opsahl. Triadic closure in two-mode networks: Redefining the global and local clustering coefficients. *Social Networks*, 35(2):159–167, 2013.

[51] J. B. Orlin. Max flows in o (nm) time, or better. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 765–774. ACM, 2013.

[52] Y. Peng, Y. Zhang, X. Lin, W. Zhang, L. Qin, and J. Zhou. Towards bridging theory and practice: hop-constrained st simple path enumeration. *PVLDB*, 13(4):463–476, 2019.

[53] Y. Peng, Y. Zhang, W. Zhang, X. Lin, and L. Qin. Efficient probabilistic k-core computation on uncertain graphs. In *ICDE*, pages 1192–1203. IEEE, 2018.

[54] P. Pesantez-Cabrera and A. Kalyanaraman. Efficient detection of communities in biological bipartite networks. *IEEE/ACM transactions on computational biology and bioinformatics*, 16(1):258–271, 2017.

[55] X. Qiu, W. Cen, Z. Qian, Y. Peng, Y. Zhang, X. Lin, and J. Zhou. Real-time constrained cycle detection in large dynamic graphs. *PVLDB*, 11(12):1876–1888, 2018.

[56] J. Scott. *Social Network Analysis: A Handbook*. Sage Publications, 2000.

[57] S. B. Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983.

[58] C. Shi, Y. Li, J. Zhang, Y. Sun, and P. S. Yu. A survey of heterogeneous information network analysis. *IEEE Trans. on Knowl. and Data Eng.*, 29(1):17–37, Jan. 2017.

[59] C. Shi, R. Wang, Y. Li, P. S. Yu, and B. Wu. Ranking-based clustering on general heterogeneous information networks by network projection. In *CIKM*, pages 699–708. ACM, 2014.

[60] M. Sozio and A. Gionis. The community-search problem and how to plan a successful cocktail party. In *KDD*, pages 939–948, 2010.

[61] Y. Sun, C. C. Aggarwal, and J. Han. Relation strength-aware clustering of heterogeneous information networks with incomplete attributes. *PVLDB*, 5(5):394–405, Jan. 2012.

[62] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *PVLDB*, 4(11):992–1003, 2011.

[63] Y. Sun, J. Han, P. Zhao, Z. Yin, H. Cheng, and T. Wu. Rankclus: Integrating clustering with ranking for heterogeneous information network analysis. In *EDBT*, pages 565–576. ACM, 2009.

[64] Y. Sun, B. Norick, J. Han, X. Yan, P. S. Yu, and X. Yu. Integrating meta-path selection with user-guided object clustering in heterogeneous information networks. In *KDD*, pages 1348–1356. ACM, 2012.

[65] Y. Sun, Y. Yu, and J. Han. Ranking-based clustering of heterogeneous information networks with star network schema. In *KDD*, pages 797–806. ACM, 2009.

[66] S. M. Van Dongen. Graph clustering by flow simulation. *PhD thesis, University of Utrecht*, 2000.

[67] G. Wan, B. Du, S. Pan, and J. Wu. Adaptive knowledge subgraph ensemble for robust and trustworthy knowledge graph completion. *World Wide Web*, pages 1–20, 2019.

[68] K. Wang, X. Cao, X. Lin, W. Zhang, and L. Qin. Efficient computing of radius-bounded k-cores. In *ICDE*, pages 233–244. IEEE, 2018.

[69] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang. Vertex priority based butterfly counting for large-scale bipartite networks. *PVLDB*, 12(10):1139–1152, 2019.

[70] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang. Efficient bitruss decomposition for large-scale bipartite graphs. In *ICDE*. IEEE, 2020.

[71] L. Wang, X. Bao, H. Chen, and L. Cao. Effective lossless condensed representation and discovery of spatial co-location patterns. *Information Sciences*, 436:197–213, 2018.

[72] L. Wang, X. Bao, and L. Zhou. Redundancy reduction for prevalent co-location patterns. *TKDE*, 30(1):142–155, 2017.

[73] L. Wang, X. Bao, L. Zhou, and H. Chen. Mining maximal sub-prevalent co-location patterns. *World Wide Web*, 22(5):1971–1997, 2019.

[74] Y. Wu, R. Jin, J. Li, and X. Zhang. Robust local community detection: on free rider effect and its elimination. *PVLDB*, 8(7):798–809, 2015.

[75] Y. Fang, Y. Yang, W. Zhang, X. Lin, X. Cao. Effective and efficient community search over large heterogeneous information networks (technical report). `http://www.cse.unsw.edu.au/~z3525370/csh.pdf`.

[76] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *International Conference on Data Mining*, pages 745–754, 2012.

[77] Y. Yang, Y. Fang, X. Lin, and W. Zhang. Effective and efficient truss computation over large heterogeneous information networks. In *ICDE*. IEEE, 2020.

[78] L. Yuan, L. Qin, W. Zhang, L. Chang, and J. Yang. Index-based densest clique percolation community search in networks. *IEEE Transactions on Knowledge and Data Engineering*, 30(5):922–935, May 2018.

[79] B. Zhang, T. Nie, D. Shen, Y. Kou, G. Yu, and Z. Zhou. A graph clustering algorithm for citation networks. In *APWeb*, pages 414–418. Springer, 2016.

[80] Y. Zhang and J. X. Yu. Unboundedness and efficiency of truss maintenance in evolving graphs. In *SIGMOD*, pages 1024–1041, 2019.

[81] Y. Zhou and L. Liu. Social influence based clustering of heterogeneous information networks. In *KDD*, pages 338–346. ACM, 2013.