Effective Community Search over Large Star-Schema Heterogeneous Information Networks

Yangqin Jiang
The Chinese University of Hong Kong, Shenzhen
jiangyangqin@cuhk.edu.cn

Yixiang Fang*
The Chinese University of Hong Kong, Shenzhen
fangyixiang@cuhk.edu.cn

Chenhao Ma
The University of Hong Kong
chma2@cs.hku.hk

Xin Cao
University of New South Wales
xin.cao@unsw.edu.au

Chunshan Li
Harbin Institute of Technology
lics@hit.edu.cn

ABSTRACT

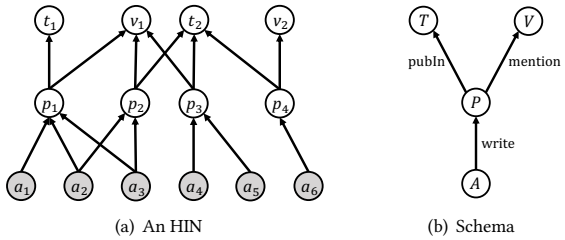
Community search (CS) enables personalized community discovery and has found a wide spectrum of emerging applications such as setting up social events and friend recommendation. While CS has been extensively studied for conventional homogeneous networks, the problem for heterogeneous information networks (HINs) has received attention only recently. However, existing studies suffer from several limitations, e.g., they either require users to specify a meta-path or relational constraints, which pose great challenges to users who are not familiar with HINs. To address these limitations, in this paper, we systematically study the problem of CS over large star-schema HINs without asking users to specify these constraints; that is, given a set Q of query vertices with the same type, find the most-likely community from a star-schema HIN containing Q, in which all the vertices are with the same type and close relationships. To capture the close relationships among vertices of the community, we employ the meta-path-based core model, and maximize the number of shared meta-paths such that each of them results in a cohesive core containing Q. To enable efficient CS, we first develop online algorithms via exploiting the anti-monotonicity property of shared meta-paths. We further boost the efficiency by proposing a novel index and an efficient index-based algorithm with elegant pruning techniques. Extensive experiments on four real large star-schema HINs show that our solutions are effective and efficient for searching communities, and the index-based algorithm is much faster than the online algorithms.

PVLDB Reference Format:
Yangqin Jiang, Yixiang Fang, Chenhao Ma, Xin Cao, and Chunshan Li. Effective Community Search over Large Star-Schema Heterogeneous Information Networks. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/ZzMeei/CS-StarSchemaHIN.

*Yixiang Fang is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX



(a) An HIN (b) Schema

Figure 1: An example HIN with a star-schema.

1 INTRODUCTION

Heterogeneous information networks (HINs) are networks with multiple typed objects and multiple typed links denoting different semantic relations. A representative type of HINs is the star-schema HIN [34, 37, 40, 50, 57], such as the bibliographic network [34, 40], IMDB movie network [37, 50], Foursquare check-in network [14], and patent network [57]. The star-schema often has a base-type and several attribute-types, serving as the center node and tail nodes respectively. Correspondingly, the vertices of base-type in the HIN play the roles of hub vertices and connect vertices of attribute-types. Figure 1(a) illustrates a star-schema HIN of the DBLP network, describing the relationships among objects of different types, i.e., author (A), paper (P), venue (V), and topic (T), and its schema is depicted in Figure 1(b), where P is the base-type and others are the attribute-types. In specific, the HIN consists of six authors (i.e., $a_1, \cdots, a_6$), four papers (i.e., $p_1, \cdots, p_4$), two venues (i.e., v_1 and v_2) and two topics (i.e., t_1 and t_2). The directed lines denote their semantic relationships. For example, the author a_3 has *written* a paper p_2, which *mentions* the topic t_2, *published* in the venue v_1.

In this paper, we study the problem of CS over star-schema HINs. Given a star-schema HIN $\mathcal{H}$ and a set of vertices Q with the same type, we aim to find a community, or a set of vertices, which are with the same type of vertices in Q and have close relationships, from $\mathcal{H}$ containing Q. The CS over HINs can be used in various real applications, including event organization, friend recommendation, and biological data analysis [11, 33]. For example, if a group of organizers want to hold an academic workshop, they can search the community with close relationships from the DBLP network, and then invite its members to join the workshop.

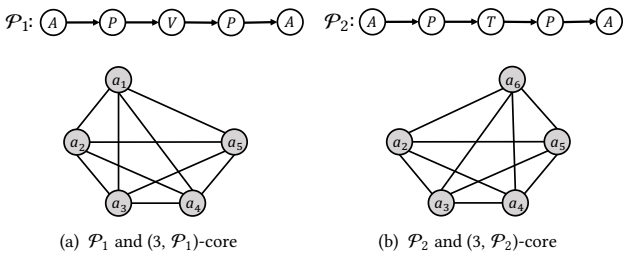
Prior works. Network community retrieval has been studied for decades. Existing works on community retrieval can be roughly

classified into *community detection* (CD) and *community search* (CS). Generally, CD algorithms aim to identify all communities for a graph [15, 27–29, 31, 41–43, 56]. These studies are not "query-based", i.e., they are not customized for a query request (e.g., a user-specified query vertex and some parameters). Moreover, for large graphs, they are often costly to detect all the communities. To tackle these issues, the query-based CS approaches (e.g., [8, 13, 20, 38]) have received much interest. While CS has been extensively studied for homogeneous networks, the problem for HINs [14, 24] has received attention only recently, and these works often require users to specify constraints like a meta-path [14] or relational constraints [24]. This poses great challenges to users, who just realize that some vertices are in the same community, but not familiar with the HIN schema or their semantic relationships.

CSSH problem. To overcome the limitations of existing CS on HINs, in this paper, we propose to find the community containing a set of query vertices Q from a star-schema HIN without specifying constraints above, such that all the vertices of the community are with the same type and close relationships. We also call it **CS** over **S**tar-schema **H**INs, or CSSH problem. To achieve the goal above, we face two key questions: (1) How to model the relationships of vertices with the same type in a community? (2) How to identify the most-likely community containing Q? For the first question, the existing work [14] shows that the $(k, \mathcal{P})$-core, or the maximal set of vertices in which each vertex has at least k neighbors linked by the path instances of meta-path $\mathcal{P}$, is effective for modeling the relationships of vertices with the same type in a community, because a meta-path with limited length between two vertices well reveals their specific semantic relationships (e.g., $\mathcal{P}$=(APA) reveals the co-authorship between two authors).

The second question is not easy to answer, because there may exist many meta-paths connecting the query vertices Q and various combinations of these meta-paths also exist, making it hard to figure out the proper meta-path(s) that can well explain the hidden relationships among the query vertices. For example, in Figure 1(a), let Q=$\{a_3, a_4\}$ and $k = 3$ (i.e., each vertex has at least 3 neighbors). Clearly, authors a_3 and a_4 can be connected via two different meta-paths $\mathcal{P}_1$=($APVPA$) and $\mathcal{P}_2$=($APTPA$), and there are two different communities containing Q, which are $(3, \mathcal{P}_1)$-core and $(3, \mathcal{P}_2)$-core, denoted by $C_1 = \{a_1, a_2, a_3, a_4, a_5\}$ and $C_2 = \{a_2, a_3, a_4, a_5, a_6\}$ respectively. Figure 2 shows the induced homogeneous graphs of these two communities, where each edge indicates a path instance of the corresponding meta-path. Since the vertices' relationships in each community can be explained by the corresponding meta-path, it is not easy to claim whether $(3, \mathcal{P}_1)$-core or $(3, \mathcal{P}_2)$-core is more likely to contain the query vertices than the other one.

Inspired by existing community retrieval works [9, 26, 54], we observe that for a set Q of query vertices, the community with more shared meta-paths intuitively can better capture the close relationships among them. In the example above, the vertex set $C_3 = \{a_2, a_3, a_4, a_5\}$ forms a $(3, \mathcal{P}_1)$-core and a $(3, \mathcal{P}_2)$-core at the same time, so it is more likely to contain the query vertices than $(3, \mathcal{P}_1)$-core and $(3, \mathcal{P}_2)$-core. Thus, it is better to find the community with the maximum shared meta-paths such that each of them results in a $(k, \mathcal{P})$-core containing the query vertices. Nevertheless, simply maximizing the number of shared meta-paths may not result in community with close relationships since the semantic meanings



(a) $\mathcal{P}_1$ and $(3, \mathcal{P}_1)$-core (b) $\mathcal{P}_2$ and $(3, \mathcal{P}_2)$-core

Figure 2: Induced homogeneous graphs of two communities.

of different meta-paths have nested relationships. In Figure 1, for example, consider the meta-paths $\mathcal{P} = (APA)$ and $\mathcal{P}_1 = (APVPA)$. The former implies that two authors have co-authored a paper, and the latter one shows that two authors have published papers in the same venue. Obviously, in the DBLP network, if two authors meet the relationships of $\mathcal{P}$, then they also naturally meet the relationships of $\mathcal{P}_1$, since two authors who have co-authored a paper must have papers published in the same venue. This also implies that the semantic meanings of $\mathcal{P}$ and $\mathcal{P}_1$ have nested relationships, so there is no sense to simply maximize the number of shared meta-paths for the community containing Q.

Fortunately, the meta-paths with nested semantic relationships can be easily detected from the schema of HIN. Therefore, by carefully considering the above issues in our CSSH problem formulation, *we aim to find a set of vertices by maximizing the number of shared meta-paths without nested semantic relationships, such that for each meta-path $\mathcal{P}$, they form a $(k, \mathcal{P})$-core containing Q*. This ensures that the returned community can well capture the rich semantic relationships carried by the query vertices.

Our technical contributions. To model the set of $(k, \mathcal{P})$-cores for meta-paths without nested semantic relationships, we propose a novel concept, called (k, Ψ)-Non-nested Meta-path Core, or (k, Ψ)-NMC, where Ψ is a set of meta-paths without nested semantic relationships and for each meta-path $\mathcal{P} \in \Psi$, there exists a $(k, \mathcal{P})$-core containing Q. A naive method of solving our problem is to enumerate the combinations of all possible meta-paths, then verify the existence of (k, Ψ)-NMC for each combination, and finally return the (k, Ψ)-NMC with the maximum size of Ψ. However, this method needs to verify $2^{|\mathcal{X}|} - 1$ possible (k, Ψ)-NMCs, where $\mathcal{X}$ is the set of all meta-paths linking vertex types of Q, thus it will be very costly specially when $\mathcal{X}$ is very large.

We observe the anti-monotonicity property, which states that given a set Ψ of meta-paths, if it results in a (k, Ψ)-NMC containing Q, then for every subset Ψ' of Ψ, there exists a (k, Ψ')-NMC containing Q. We use this intuition to propose an online algorithm by reducing the number of candidates verified. However, this algorithm needs to build many homogeneous graphs induced by different meta-paths, leading to expensive cost w.r.t. time and space. We further boost the efficiency by developing a faster online algorithm which tries to save the cost of building homogeneous graphs.

Although the online algorithms perform well on many HINs, they are still inefficient to process large-scale HINs with complicated schemas. We observe that there is much repeated computation among different queries, which can be avoided if some prepossessing is made. In light of this, we design a novel index, which stores

the core numbers of vertices w.r.t. different meta-paths and exploits an efficient compression strategy to reduce the space cost. Moreover, much unnecessary verification of candidates can be pruned by exploiting our index and the set of vertices for candidate verification can also be minimized. Using the techniques above, we design a fast index-based algorithm with elegant pruning techniques.

In addition, we have performed extensive experiments on four real large star-schema HINs. We first evaluate the effectiveness of our CS query by analyzing the returned communities from different angles and showing a case study. We then thoroughly evaluate the efficiency of our algorithms, and the experimental results show that they are efficient to answer the queries. In particular, our index-based query algorithm is over two orders of magnitude faster than the basic online query algorithm.

Outline. We formulate our CSSH problem in Section 2. In Sections 3 and 4, we show online algorithms and index-based algorithm respectively. We report the experimental results in Section 5. We review the related works in Section 6 and conclude in Section 7.

2 PROBLEM DEFINITION

2.1 Preliminaries

We first present the basic concepts about HIN. Frequently used notations are summarized in Table 1.

Definition 1. ***HIN [23, 40].*** *An HIN is a directed graph $\mathcal{H} = (V, E)$ 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 $\psi(v) \in \mathcal{A}$, each edge $e \in E$ belongs to an edge type (also called relation) $\phi(e) \in \mathcal{R}$, and $|\mathcal{A}| + |\mathcal{R}| > 2$.*

Definition 2. ***HIN schema [23, 40].*** *Given an HIN $\mathcal{H} = (V, E)$ with mappings $\psi : V \rightarrow \mathcal{A}$ and $\phi : E \rightarrow \mathcal{R}$, its schema $T_{\mathcal{H}}$ is a directed graph defined over vertex types $\mathcal{A}$ and edge types (as relations) $\mathcal{R}$, i.e., $T_{\mathcal{H}} = (\mathcal{A}, \mathcal{R})$.*

The HIN schema describes all allowable edge types between vertex types, where each edge type can denote one-to-one, one-to-many, or many-to-many relationships. Note that if there is an edge type R from vertex type A to vertex type B, the inverse edge type R^{-1} naturally exists from B to A. In this paper, we focus on the HIN with a star-schema, which has a base-type and several attribute-types, serving as the center node and tail nodes respectively. A vertex of base-type plays the role of hub vertex in the HIN, as it connects a set of vertices whose types cover all the attribute-types.

Definition 3. ***Meta-path [40].*** *A meta-path $\mathcal{P}$ is a path defined on an HIN schema $T_{\mathcal{H}} = (\mathcal{A}, \mathcal{R})$, and is denoted in the form $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \cdots \xrightarrow{R_L} A_{L+1}$, or $(A_1 A_2 \cdots A_{L+1})$, where $L = |\mathcal{P}|$ is the length of $\mathcal{P}$, $A_i \in \mathcal{A}$, and $R_i \in \mathcal{R} (1 \leqslant i \leqslant L)$.*

In the HIN, 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 $\phi(e_i){=}R_i$. 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 $T_{\mathcal{H}}$, and denote it by $\mathcal{P}^{-1}$. We say $\mathcal{P}$ is *symmetric*, if it is the same with $\mathcal{P}^{-1}$ [14]. For a symmetric meta-path $\mathcal{P} = (A_1 A_2 \cdots A_{n+1} \cdots A_2 A_1)$, we use $\mathcal{P}_{half} = (A_1 A_2 \cdots A_{n+1})$ to denote the *half* meta-path of $\mathcal{P}$, where L is the length of $\mathcal{P}$ and $n = L/2$. Note that since we focus on finding communities of vertices with the same type, all the meta-paths mentioned in this paper are symmetric.

Table 1: Notations and meanings.

Notation	Meaning
$\mathcal{H} = (V, E)$	An HIN with vertex set V and edge set E
$\mathcal{P}$	A symmetric meta-path defined on the schema of $\mathcal{H}$
$\mathcal{H}_{\mathcal{P}}$	A homogeneous graph induced by a meta-path $\mathcal{P}$ on $\mathcal{H}$
$d_{\mathcal{P}}(v, \mathbf{B}_{k,\mathcal{P}})$	$\mathcal{P}$-degree, or the number of $\mathcal{P}$-neighbors of v in S
Ψ	A set of meta-paths without nested semantic relationships
$\mathbf{B}_{k,\mathcal{P}}$	A basic $(k, \mathcal{P})$-core [14]
(k, Ψ)-NMC	(k, Ψ)-non-nested meta-path core, abbreviated as Ψ-NMC
Q	A set of query vertices

In line with [14], we say that a vertex u is a $\mathcal{P}$*-neighbor* of a vertex v, if they are connected by an instance of $\mathcal{P}$, and two vertices u and v are $\mathcal{P}$*-connected*, if there exists a chain of vertices from u to v such that any two adjacent vertices in the chain are the $\mathcal{P}$-neighbor of each other. Given a vertex v and a set S of vertices with the same type, we define $d_{\mathcal{P}}(v, S)$, called $\mathcal{P}$*-degree*, as the number of $\mathcal{P}$-neighbors of v within the set S. Based on the concept of $\mathcal{P}$-degree, Fang et al. [14] introduced the $(k, \mathcal{P})$-core model:

Definition 4. ***(k, $\mathcal{P}$)-core [14].*** *Given an HIN $\mathcal{H}$ and an integer k, a $(k, \mathcal{P})$-core of $\mathcal{H}$ is a maximal set $\mathbf{B}_{k,\mathcal{P}}$ of $\mathcal{P}$-connected vertices, s.t. $\forall v \in \mathbf{B}_{k,\mathcal{P}}, d_{\mathcal{P}}(v, \mathbf{B}_{k,\mathcal{P}}) \geqslant k$, where vertices of $\mathbf{B}_{k,\mathcal{P}}$ are with the type linked by $\mathcal{P}$.*

Definition 5. ***Core number.*** *Given an HIN $\mathcal{H}$, a vertex $v \in V$ and a meta-path $\mathcal{P}$, its core number, denoted by $core_{\mathcal{P}}[v]$, is the largest k such that there exists a $(k, \mathcal{P})$-core, $\mathbf{B}_{k,\mathcal{P}}$, containing v.*

For example, in Figure 1, let $k = 2$ and $\mathcal{P} = (APA)$. Then, we can find a $\mathbf{B}_{2,\mathcal{P}} = \{a_1, a_2, a_3\}$, since each vertex of $\mathbf{B}_{2,\mathcal{P}}$ has at least 2 $\mathcal{P}$-neighbors in $\mathbf{B}_{2,\mathcal{P}}$. Besides, the core numbers of all vertices in $\mathbf{B}_{2,\mathcal{P}}$ are 2, since there is no $\mathbf{B}_{3,\mathcal{P}}$. We would like to remark that in [14], Fang et al. introduced two variants of the basic $(k, \mathcal{P})$-core model above, and our solutions can also be extended for them, so we omit the detailed discussions for lack of space.

For each meta-path $\mathcal{P}$, we can also induce a homogeneous graph from $\mathcal{H}$, called $\mathcal{P}$-graph, which is denoted by $\mathcal{H}_{\mathcal{P}}$. Essentially, the $(k, \mathcal{P})$-core is the set of vertices in a connected k-core of $\mathcal{H}_{\mathcal{P}}$.

Definition 6. ***$\mathcal{P}$-Graph [49].*** *Given an HIN $\mathcal{H}$ and a meta-path $\mathcal{P}$, the $\mathcal{P}$-graph is a homogeneous graph $\mathcal{H}_{\mathcal{P}}$, such that it contains all the vertices with the type linked by $\mathcal{P}$ and for every vertex, it has an edge linked to each of its $\mathcal{P}$-neighbors.*

2.2 Problem definition

To formulate the community model, we propose the concepts of nested meta-path and (k, Ψ)-NMC.

Definition 7. ***Nested meta-path.*** *Given a star-schema HIN and a symmetric meta-path $\mathcal{P}$ whose half meta-path is $\mathcal{P}_{half} = (A_1 A_2 \cdots A_i \cdots A_j)$, the nested meta-path of $\mathcal{P}$ is a symmetric meta-path $\mathcal{P}'$ whose half meta-path is $\mathcal{P}'_{half} = (A_1 A_2 \cdots A_i)$, where $j \geq i$. We use $\mathcal{P}' \sqsubseteq \mathcal{P}$ to denote that $\mathcal{P}'$ is nested in $\mathcal{P}$.*

In the star-schema HIN, the nested meta-paths of a meta-path can be easily detected by traversing its schema network. Besides, in

the star-schema HIN, for any two vertices, if there is a path instance of $\mathcal{P}'$ between them, then naturally there also exists a path instance of $\mathcal{P}$ between them. This also implies that there exists a nested relationship between the two cores of two meta-paths with nested relationships, where the nested core is structurally more compact, as stated by Lemma 1. Example 1 illustrates this.

Lemma 1. *Given a star-schema HIN $\mathcal{H}$, a positive integer k, two meta-paths $\mathcal{P}$ and $\mathcal{P}'$ with $\mathcal{P}' \sqsubseteq \mathcal{P}$, if there exists a $(k, \mathcal{P}')$-core $\mathbf{B}_{k,\mathcal{P}'}$, then there must exist a $(k, \mathcal{P})$-core $\mathbf{B}_{k,\mathcal{P}}$, s.t. $\mathbf{B}_{k,\mathcal{P}'} \subseteq \mathbf{B}_{k,\mathcal{P}}$.*

Proof. If there exists a $\mathbf{B}_{k,\mathcal{P}'}$, then each vertex v of $\mathbf{B}_{k,\mathcal{P}'}$ has at least k $\mathcal{P}'$-neighbors, which further implies that it has at least k $\mathcal{P}$-neighbors. Hence, there must exist a $\mathbf{B}_{k,\mathcal{P}}$, s.t. $\mathbf{B}_{k,\mathcal{P}'} \subseteq \mathbf{B}_{k,\mathcal{P}}$. □

Example 1. *Reconsider the HIN in Figure 1. Since the meta-path $\mathcal{P}$=(APA) is a nested meta-path of $\mathcal{P}_2$=(APTPA), the $(k, \mathcal{P})$-core is nested in the $(k, \mathcal{P}_2)$-core. E.g., the $(2, \mathcal{P})$-core contains vertices $\{a_1, a_2, a_3\}$ and $(2, \mathcal{P}_2)$-core contains vertices $\{a_1, a_2, a_3, a_4, a_5, a_6\}$, so the former one is nested in the latter one and it is more compact.*

Definition 8. ***Non-nested meta-path core.*** *Given an HIN $\mathcal{H}$, an integer k, and a set Ψ of meta-paths without nested meta-paths (i.e., $\forall \mathcal{P}, \mathcal{P}' \in \Psi$, $\mathcal{P} \not\sqsubseteq \mathcal{P}'$ and $\mathcal{P}' \not\sqsubseteq \mathcal{P}$), a (k, Ψ)-Non-nested Meta-path Core, or (k, Ψ)-NMC, of $\mathcal{H}$ is a maximal set of vertices, such that $\forall \mathcal{P} \in \Psi$, the vertices of (k, Ψ)-NMC form a $(k, \mathcal{P})$-core.*

Example 2. *In Figure 1, let $\Psi = \{\mathcal{P}_1, \mathcal{P}_2\}$ with $\mathcal{P}_1$=(APVPA) and $\mathcal{P}_2$=(APTPA). Then, there exists a (3, Ψ)-NMC = $\{a_2, a_3, a_4, a_5\}$, since these vertices form a (3, $\mathcal{P}_1$)-core and also a (3, $\mathcal{P}_2$)-core.*

Intuitively, if Ψ=$\{\mathcal{P}, \mathcal{P}'\}$ and $\mathcal{P}' \sqsubseteq \mathcal{P}$, then Ψ can be simplified as Ψ'=$\{\mathcal{P}\}$, since the (k, Ψ)-NMC is the same with (k, Ψ')-NMC. This also implies that there exists some redundant semantic relationships in (k, Ψ)-NMC and thus (k, Ψ')-NMC is preferred. Next, we formally introduce our CSSH problem.

Problem 1. *Given a star-schema HIN $\mathcal{H}$, a set of query vertices Q with the same type, and a positive integer k, return a (k, Ψ)-NMC such that it contains Q and Ψ satisfies the following properties:*

*1. **Set non-nestedness:** There does not exist any (k, Ψ')-NMC such that Ψ' is formed by nested meta-paths of meta-paths in Ψ;*

*2. **Size maximality:** On the premise that Property 1 is satisfied, the size of Ψ is maximized.*

For simplicity, we call the type of vertices Q *target type*. The first property ensures that there is no other community whose semantic relationships carried are nested in the relationships revealed by meta-paths of Ψ. Intuitively, given two meta-path sets Ψ=$\{\mathcal{P}\}$ and Ψ'=$\{\mathcal{P}'\}$ with $\mathcal{P}' \sqsubseteq \mathcal{P}$, if both (k, Ψ)-NMC and (k, Ψ')-NMC exist, then (k, Ψ')-NMC should be preferred for modeling the community, because the nested meta-path is shorter and the corresponding core is more compact. As pointed out by many recent studies on meta-paths [14, 40], long meta-paths often indicate weak relationships, so shorter meta-paths with limited lengths are often more meaningful and frequently used in practice. In line with these works, we can set a limited length (e.g., L=4) for the meta-paths used. And CSSH problem is fixed-parameter tractable (FPT) with respect to parameter L. The proof of the hardness of CSSH problem is in the appendix of the technical report [25]. The second property guarantees that the semantic relationships carried by the (k, Ψ)-NMC are rich. Without any users' given meta-paths, by maximizing the number of shared meta-paths, the semantic relationships among the query vertices can be well captured.

Example 3. *In Figure 1 with $\mathcal{P}$=(APA), $\mathcal{P}_1$=(APVPA), and $\mathcal{P}_2$=(APTPA), if Q=(a_2, a_3) and k=3, then we will return the (3, Ψ)-NMC = $\{a_2, a_3, a_4, a_5\}$ with Ψ=$\{\mathcal{P}_1, \mathcal{P}_2\}$. This is because for $\mathcal{P}_1$ and $\mathcal{P}_2$, which are not nested with each other, we have $\mathbf{B}_{3,\mathcal{P}_1}$=$\{a_1, a_2, a_3, a_4, a_5\}$ and $\mathbf{B}_{3,\mathcal{P}_2}$=$\{a_2, a_3, a_4, a_5, a_6\}$, so we take the intersection of two cores to maximize the size of Ψ. Similarly, if Q=(a_2, a_3) and k=2, we will return (2, Ψ)-NMC =$\{a_1, a_2, a_3\}$ with Ψ=$\{\mathcal{P}\}$.*

3 ONLINE ALGORITHMS

For simplicity, in the context without ambiguity, we write $\mathbf{B}_{k,\mathcal{P}}$ and (k, Ψ)-NMC as $\mathbf{B}_{\mathcal{P}}$ and Ψ-NMC, respectively. Recall that the CSSH problem aims to maximize the number of shared meta-paths that satisfies the property of non-nestedness. Thus, a straightforward method to solve this problem performs three steps. First, we generate the set of all possible meta-paths $\mathcal{X}$, which link the target type and have the limited length, from the HIN schema, and enumerate all nonempty subsets of $\mathcal{X}$: $\Psi_1, \Psi_2, \cdots, \Psi_{2^{|\mathcal{X}|}-1}$. Second, for each subset Ψ_i $(1 \leqslant i \leqslant 2^{|\mathcal{X}|} - 1)$, we verify the existence of Ψ_i-NMC. Finally, we output the Ψ_i-NMC, where Ψ_i-NMC satisfies the properties of set non-nestedness and size maximality. One major drawback of this method is that we need to verify $2^{|\mathcal{X}|} - 1$ possible Ψ_i-NMCs. For a large set of meta-paths $\mathcal{X}$, the exponential computation cost renders the method impractical. To improve efficiency, we propose a novel two-step framework based on the anti-monotonicity property of the shared meta-paths.

3.1 A two-step framework

We begin with a lemma inspiring the design of our algorithms.

Lemma 2. *(Anti-monotonicity) Given a star-schema HIN $\mathcal{H}$, a set of vertices Q in $\mathcal{H}$, a positive integer k, and a set Ψ of meta-paths, if there exists a Ψ-NMC containing Q, then for any subset $\Psi' \subseteq \Psi$, there will also exist a Ψ'-NMC containing Q.*

Proof. Based on the definition of Ψ-NMC, $\forall \mathcal{P} \in \Psi$, vertices of Ψ-NMC can form a $\mathbf{B}_{\mathcal{P}}$ in $\mathcal{H}$, s.t. $\forall v \in \mathbf{B}_{\mathcal{P}}$, $d_{\mathcal{P}}(v, \mathbf{B}_{\mathcal{P}}) \geq k$. Consider a new meta-path set $\Psi' \subseteq \Psi$. For every $\mathcal{P} \in \Psi'$, we can easily conclude that $\mathcal{P} \in \Psi$ and the vertices of Ψ-NMC can form a $\mathbf{B}_{\mathcal{P}}$, s.t. $\forall v \in \mathbf{B}_{\mathcal{P}}$, $d_{\mathcal{P}}(v, \mathbf{B}_{\mathcal{P}}) \geq k$. Also, note that $Q \subseteq \Psi$-NMC. Therefore, there exists a Ψ'-NMC containing vertices in Q. □

The anti-monotonicity above allows us to stop examining all the supersets of Ψ, once we have verified that Ψ-NMC does not exist, thus saving much computational cost. Based on Lemma 2, we begin with examining the set, $\mathcal{S}_1$, of size-1 candidate meta-path sets, each of which contains a single meta-path in $\mathcal{X}$. We then repeatedly execute the following two key steps, to retrieve the size-2 (size-3, $\cdots$) qualified meta-path subsets, until no qualified meta-path subsets are found:

• **Verification.** For each candidate meta-path set Ψ in $\mathcal{S}_c$ (initially $c = 1$), mark Ψ as a qualified set if Ψ-NMC exists.

• **Candidate generation.** For any two size-c qualified meta-path sets which only differ in one meta-path, union them as a

newly expanded candidate with size-$(c+1)$, and put it into set $\mathcal{S}_{c+1}$, if all its subsets are qualified, by Lemma 2.

Based on the above discussions, we present our two-step framework in Algorithm 1. We first invoke GenMetaPaths on the HIN schema $T_{\mathcal{H}}$ to get all the eligible meta-paths $\mathcal{X}$ with limited length L (line 1), and the details of GenMetaPaths are in the technical report [25]. Then, we initialize an integer c=1 and a hash map $\mathcal{M}_c$ to store the corresponding community of every Ψ (line 2). $\mathcal{S}_c$ is initialized, in which each meta-path set contains a meta-path in $\mathcal{X}$ (line 3). Next, the two key steps, **verification** and **candidate generation**, are repeated iteratively until we obtain the community (lines 4-10). In each iteration, we first verify each candidate meta-path set Ψ in $\mathcal{S}_c$ sequentially (lines 5-8). The verification can be completed by computing the Ψ-NMC (line 6), and we will elaborate its algorithms extensively later. If there exists a valid community, we put the community to the hash map $\mathcal{M}_c$ (line 7); otherwise, we will remove Ψ from $\mathcal{S}_c$ since it is an invalid candidate (line 8). After all candidates in $\mathcal{S}_c$ are examined, we use GenCan to generate new size-$(c+1)$ sets of meta-paths w.r.t. $\mathcal{S}_c$ (line 9). GenCan generates new candidates by combining two size-c qualified meta-path sets as a new size-$(c+1)$ candidate via Lemma 2, and its details are in our technical report [25]. Afterward, c is increased by 1. The whole loop will stop when there does not exist any valid meta-path set (line 11). Finally, for every valid meta-path set with size $c-1$, we simply get the corresponding community from $\mathcal{M}_{c-1}$ (lines 12-14).

Algorithm 1: Our two-step framework

```
   Input: H, Q, k, L
   Output: Ψ-NMC with the maximum |Ψ|
1  X ← GenMetaPaths(T_H, target type, L);
2  initialize c ← 1, M_c ← ∅;
3  initialize S_c ← {{P}|P ∈ X};
4  repeat
5      foreach Ψ ∈ S_c do
6          Ψ-NMC ← compute the Ψ-NMC containing Q from H;
7          if Ψ-NMC exists then M_c.put(Ψ, Ψ-NMC) ;
8          else S_c.remove(Ψ) ;
9      S_{c+1} ← GenCan(S_c);
10     c ← c+1;
11 until S_c = ∅;
12 foreach Ψ-NMC in M_{c-1} do
13     if Ψ satisfies property of set non-nestedness then
14         output Ψ-NMC;
```

Lemma 3. *Algorithm 1 takes $O(\sum_{i=1}^{c}\sum_{j=1}^{|\mathcal{M}_i|}\Delta_j)$ time, where c denotes the max size of Ψ, $\mathcal{M}_i$ denotes the hash map that stores the corresponding community of every Ψ and Δ_j denotes the time cost of Ψ-NMC computation, s.t.* HomNMC *(Algorithm 2) or* FastNMC *(Algorithm 3).*

Proof. In Algorithm 1, it needs $\sum_{i=1}^{c}|\mathcal{M}_i|$ times of Ψ-NMC computation. Hence, Lemma 3 holds. □

Example 4. *Consider the HIN in Figure 1 and let Q=(a_3, a_4), k=3. In Algorithm 1, we first use* GenMetaPaths *to generate all meta-paths $\mathcal{X}$ = {APA, APTPA, APVPA} (assume L=4). Then we initialize $\mathcal{S}_1$ with the size-1 candidates (i.e., Ψ_0 = {APA}, Ψ_1 = {APVPA}, Ψ_2 = {APTPA}) and $\forall\Psi \in \mathcal{S}_1$, verify if Ψ-NMC exists. Clearly, there are two Ψ-NMCs, i.e., Ψ_1-NMC = $(a_1, a_2, a_3, a_4, a_5)$ and Ψ_2-NMC = (a_2, a_3, a_4, a_5). After verifying all size-1 candidate sets, we use* GenCan *to generate size-2 candidates and we have a candidate Ψ_3 = {APVPA, APTPA}. Since* GenCan *cannot generate size-3 candidates and Ψ_3 satisfies the properties of non-nestedness and maximality, Ψ_3-NMC will be returned as the community.*

In our two-step framework, a key issue is how to efficiently verify the existence of Ψ-NMC for each meta-path set Ψ, i.e., line 5 of Algorithm 1. To tackle this issue, we first develop a naive algorithm and then a fast algorithm in the following two subsections.

3.2 A naive algorithm to compute Ψ-NMC

Here, we give a naive algorithm to compute the Ψ-NMC consists of two steps. First, we derive a $\mathcal{P}_i$-graph $\mathcal{H}_{\mathcal{P}_i}$ from $\mathcal{H}$ for each $\mathcal{P}_i \in \Psi$, so we get a set of homogeneous graphs $\{\mathcal{H}_{\mathcal{P}_i}|1 \le i \le |\Psi|\}$, which contain all the vertices with target type but have different sets of edges. Second, we compute the k-cores simultaneously on all homogeneous graphs by keeping peeling the vertices whose degrees are less than k in any $\mathcal{P}$-graph, until all the vertices have at least k neighbors in each graph. We call this algorithm HomNMC.

Algorithm 2 presents the details of HomNMC. Given Ψ, we first compute the set of induced homogeneous graphs $\{\mathcal{H}_{\mathcal{P}_i}|1 \le i \le |\Psi|\}$, w.r.t. the meta-paths in Ψ (lines 2-3). Then, we assign V as the intersection of each connected component containing Q in each $\mathcal{P}$-graph (line 5). Next, we keep peeling the vertex from V if its degree in any $\mathcal{P}$-graph is less than k (lines 6-7). We repeat the above process until V only contains one connected component (lines 4-8). Finally, we return V as the Ψ-NMC (line 9).

Algorithm 2: HomNMC (naive algorithm to get Ψ-NMC)

```
1 Function HomNMC(H, Q, k, Ψ):
2     foreach P_i ∈ Ψ do
3         H_{P_i} ← compute the P_i-graph from H;
4     repeat
5         V ← ⋂_{P_i∈Ψ} connected component in H_{P_i} containing Q;
6         while ∃u ∈ V, P_i ∈ Ψ, d_{P_i}(u, V) < k do
7             remove u from V and all P_i-graphs;
8     until V is connected on all P_i-graphs;
9     return V;
```

Lemma 4. *Algorithm 2 takes $O(((c+1)\cdot m + c\cdot n_1)\cdot|\Psi| + \sum_{i=1}^{|\Psi|}(n_1\cdot b_{1,2} + n_1\sum_{i=2}^{L} n_i\cdot b_{i,i+1}))$ time, where m denotes the maximum number of edges in all $\mathcal{H}_{\mathcal{P}_i}$ ($\mathcal{P}_i \in \Psi$), c is the number of iterations in the loop, n_i denotes the number of vertices with i-th vertex type in $\mathcal{P}$ and $b_{i,i+1}$ denotes the maximum number of vertices with $(i+1)$-th vertex type that are connected to a vertex with i-th vertex type in $\mathcal{P}$.*

Proof. In the worst case, we need to remove all vertices with the target type which takes $O(m\cdot|\Psi|)$ time. In every loop, for all $\mathcal{H}_{\mathcal{P}_i}$ ($\mathcal{P}_i \in \Psi$), it takes $O(n_1\cdot|\Psi|)$ time to compute the degree of each vertex and it takes $O(m\cdot|\Psi|)$ time to compute a connected component. Apart from this, at the beginning of the algorithm, it takes $O(\sum_{i=1}^{|\Psi|}(n_1\cdot b_{1,2} + n_1\sum_{i=2}^{L} n_i\cdot b_{i,i+1}))$ time to compute all $\mathcal{H}_{\mathcal{P}_i}$ ($\mathcal{P}_i \in \Psi$). Hence, Lemma 4 holds. □

3.3 A fast algorithm to compute Ψ-NMC

Although the time complexity of HomNMC is almost linear to the size of all homogeneous graphs $\{\mathcal{H}_{\mathcal{P}_i}\}$, it is still very costly because building these homogeneous graphs involves much time and space cost, dominating the overall cost. To improve the efficiency, we optimize HomNMC from the following two observations:

(1) We observe that given previously computed Ψ-NMCs with $|\Psi|$=c, we can quickly compute Ψ'-NMCs with $|\Psi'|$=c+1.
(2) Inspired by an algorithm of computing $(k, \mathcal{P})$-core [14], we observe that it is unnecessary to find all the $\mathcal{P}$-neighbors for each vertex with target type, as we only need to know whether a vertex has k $\mathcal{P}$-neighbors in (k, Ψ)-NMC.

To exploit the first observation, we show an interesting lemma.

Lemma 5. *Given a star-schema HIN $\mathcal{H}$ and two sets Ψ_1, Ψ_2 of meta-paths, if both Ψ_1-NMC and Ψ_2-NMC exist, then*

$$(\Psi_1 \cup \Psi_2)\text{-NMC} \subseteq \Psi_1\text{-NMC} \cap \Psi_2\text{-NMC}.$$

Proof. By definition of Ψ-NMC, $\forall \mathcal{P} \in \Psi_1$, vertices of Ψ_1-NMC can form a $\mathbf{B}_{\mathcal{P}}$ in $\mathcal{H}$, s.t. $\forall v \in \mathbf{B}_{\mathcal{P}}$, $d_{\mathcal{P}}(v, \mathbf{B}_{\mathcal{P}}) \geqslant k$. Similar statements also hold for Ψ_2-NMC and $(\Psi_1 \cup \Psi_2)$-NMC. Hence, it is easy to conclude that vertices of $(\Psi_1 \cup \Psi_2)$-NMC can form a Ψ_1-NMC, or a Ψ_2-NMC, so $(\Psi_1 \cup \Psi_2)$-NMC is the subset of the intersection of Ψ_1-NMC and Ψ_2-NMC, indicating that Lemma 5 holds. □

Lemma 5 implies that the vertex set of a Ψ'-NMC with $|\Psi'|$=c+1 is actually a subset of the intersection of a few Ψ-NMCs with $|\Psi|$=c. As a result, the homogeneous graphs for computing Ψ'-NMC can be built on these vertices, rather than all the vertices with the target type in the whole HIN as what HomNMC does.

Given a set of vertices by intersecting Ψ-NMCs, the second observation indicates that to compute a Ψ'-NMC, we actually do not have to build the whole homogeneous graphs; instead, we can just find up to k $\mathcal{P}$-neighbors for each vertex, during which it repeatedly deletes vertices with less than k $\mathcal{P}$-neighbors and in the meantime finds new $\mathcal{P}$-neighbors incrementally if needed. In summary, by utilizing the two observations above, we obtain a fast algorithm for computing the Ψ-NMC, denoted by FastNMC.

Algorithm 3 presents FastNMC. First, we locate the Ψ-NMC in a small vertex set V by intersecting Ψ_1-NMC and Ψ_2-NMC stored in $\mathcal{M}$ according to Lemma 5 (line 3). Next, we check whether the vertex set contains query vertices, and if it does not, then there does not exist a Ψ-NMC (line 4). Then for each $\mathcal{P} \in \Psi$, we find up to k $\mathcal{P}$-neighbors for every vertex in V (line 8). If we cannot find k $\mathcal{P}$-neighbors, we will add this vertex v to S for deletion (line 9). At the same time, we use the hash map $\mathcal{R}$ to record the neighbors of each vertex (Φ) w.r.t. different $\mathcal{P}$ (line 10). Now we get a set S of unqualified vertices which need to be deleted. Next, we first delete vertices in S and update $\mathcal{R}$ by calling function DeleteVertex (line 12), which removes each vertex v in S from V, incrementally supplies new $\mathcal{P}$-neighbors for v's $\mathcal{P}$-neighbors for all $\mathcal{P}$ in Ψ, and if we cannot find k $\mathcal{P}$-neighbors for some $\mathcal{P}$-neighbors of v, we put them into S. The details of DeleteVertex are in the appendix of the technical report [25]. Then, we get V' as the intersection of each connected component containing Q w.r.t. each $\mathcal{P}$ in Ψ (line 13). The vertices in $V \setminus V'$ are added into S and V is updated by V' (lines 14-15). The above process is repeated until V is a connected component w.r.t. all meta-paths in Ψ (line 16).

Algorithm 3: FastNMC (fast algorithm to get Ψ-NMC)

```
1  Function FastNMC(H, Q, k, Ψ, M):
2      S ← ∅, R ← an empty hash map;
3      V ← M[Ψ1] ∩ M[Ψ2] ;                          // Ψ1 ∪ Ψ2 = Ψ
4      if Q ⊈ V then return ∅ ;
5      foreach P ∈ Ψ do
6          Φ ← ∅;
7          foreach vertex v ∈ V do
8              Φ[v] ← find up to k P-neighbors of v;
9              if |Φ[v]| < k then S.add(v) ;
10         R.put(P, Φ);
11     repeat
12         DeleteVertex(R, S, V);
13         V' ← intersect components containing Q;
14         S ← V \ V';
15         V ← V';
16     until ∀P ∈ Ψ, V is a connected component w.r.t P;
17     return V;
```

Lemma 6. *Algorithm 3 takes $O(\sum_{i=1}^{|\Psi|}(n_1 \cdot b_{1,2} + n_1 \sum_{i=2}^{L} n_i \cdot b_{i,i+1}) + c \cdot \sum_{i=1}^{|\Psi|}(\sum_{i=1}^{L} n_i \cdot b_{i,i+1}))$ time, where c is the number of iterations in the loop.*

Proof. In the worst case, we need to delete all vertices with the target type, which costs $O(\sum_{i=1}^{|\Psi|}(n_1 \cdot b_{1,2} + n_1 \sum_{i=2}^{L} n_i \cdot b_{i,i+1}))$ time. In every iteration, for every $\mathcal{P}$ in Ψ, it costs $O(\sum_{i=1}^{L} n_i \cdot b_{i,i+1})$ to compute a connected component on the homogeneous graph. So the computation of all connected components costs $O(c \cdot \sum_{i=1}^{|\Psi|}(\sum_{i=1}^{L} n_i \cdot b_{i,i+1}))$ time. Hence, Lemma 6 holds. □

4 INDEX-BASED ALGORITHMS

Generally, the online algorithms perform well on many HINs, but they may be inefficient to process large-scale HINs. As we will show later, they may take over 0.5 minutes to answer a CS query on a million-scale HIN like DBLP, which is costly if the queries are frequently issued. To alleviate this issue, we propose a novel space-efficient index, relying on two key observations: First, the lengths of frequently used meta-paths in practice are often limited [30, 35, 40, 42], as long meta-paths result in weak relationships [40]. Second, there are much repeated computations among different queries when computing the cores. As the number of meaningful meta-paths is often limited, we can pre-compute their $(k, \mathcal{P})$-cores, and compactly organize them into a tree structure, called Core Meta-path tree or **CM-tree**. Based on the CM-tree, we propose a query algorithm with elegant pruning techniques. Next, we first give an overview of CM-tree, then present the index construction algorithm, and finally show the index-based query algorithm.

4.1 Index overview

The CM-tree index is built based on the key observations that the $(k, \mathcal{P})$-cores are nested with each other regarding to the parameters k and $\mathcal{P}$, respectively:

(1) For parameter $k \geq 1$, Fang et al. [14] showed that $\mathbf{B}_{k+1,\mathcal{P}} \subseteq \mathbf{B}_{k,\mathcal{P}}$, as stated in Lemma 7.

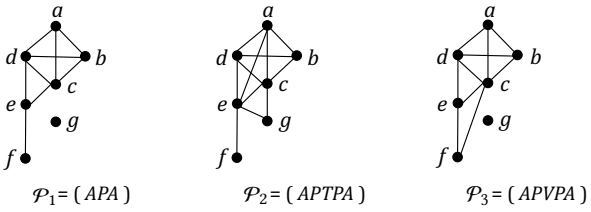
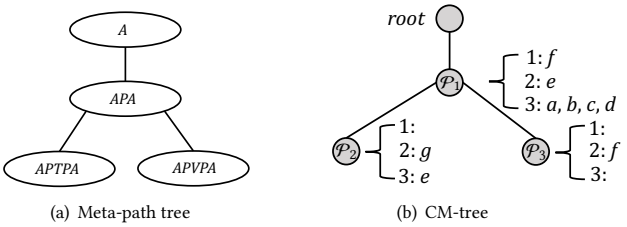


Figure 3: Three $\mathcal{P}$-graphs induced by three meta-paths.



(a) Meta-path tree (b) CM-tree

Figure 4: Meta-path tree and the corresponding CM-tree.

(2) For parameter $\mathcal{P}$, given two meta-paths $\mathcal{P}$ and $\mathcal{P}'$, if $\mathcal{P}' \sqsubseteq \mathcal{P}$, then $\mathbf{B}_{k,\mathcal{P}'} \subseteq \mathbf{B}_{k,\mathcal{P}}$, as stated by Lemma 1.

Lemma 7. *[14]. Given an HIN $\mathcal{H}$, a positive integer k and a meta-path $\mathcal{P}$, the $(k,\mathcal{P})$-cores are nested, i.e., for any $(k+1,\mathcal{P})$-core $\mathbf{B}_{k+1,\mathcal{P}} \neq \emptyset$, there must exist a $(k,\mathcal{P})$-core $\mathbf{B}_{k,\mathcal{P}}$ such that $\mathbf{B}_{k+1,\mathcal{P}} \subseteq \mathbf{B}_{k,\mathcal{P}}$.*

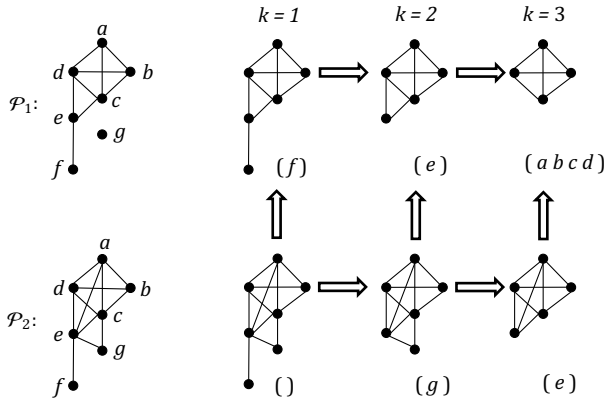
By carefully considering the nested properties, we can compactly organize all the cores in a tree index, by exploiting the following two compression methods:

• **Meta-path-based compression (MC):** Consider the tree in Figure 4(a), which organizes all the meta-paths linked the target type into a tree by using their nested relationships. Every node in the meta-path tree has a parent node and a list of child nodes (except for leaf nodes), where the meta-path in the parent node is the nested meta-path of those in the child nodes. According to Lemma 1, when k is fixed, the $(k,\mathcal{P}')$-core in the parent must be contained by the $(k,\mathcal{P})$-core in the child. Thus, to save space cost, for each child node, we only store the vertices that are not contained in its parent node.

• **K-core-based compression (KC):** Apart from MC, we can also make use of the nestedness property of cores w.r.t. different k values. According to Lemma 7, for a certain meta-path $\mathcal{P}$, $\mathbf{B}_{k+1,\mathcal{P}} \subseteq \mathbf{B}_{k,\mathcal{P}}$. Hence, for each meta-path, we can use a hash map to record mappings from core numbers to corresponding vertices, as the core number $core_{\mathcal{P}}(v)$ is the largest k such that $v \in \mathbf{B}_{k,\mathcal{P}}$.

Notice that if only MC is used, the index will contain redundant vertices for different k values. Similarly, if only KC is used, the index will be redundant for nested meta-paths. Fortunately, we can combine them, and thus obtain an effective compression method, called **meta-path and k-core-based compression (MKC)**.

Specifically, for each meta-path $\mathcal{P}$, we use $V_{\mathcal{P}}[k]$ to denote the set of vertices in all $(k,\mathcal{P})$-cores. Note that there may exist multiple $(k,\mathcal{P})$-cores in an HIN, as a core needs to be a connected component. Based on Lemma 1, we have $V_{\mathcal{P}'}[k] \subseteq V_{\mathcal{P}}[k]$ for $\mathcal{P}' \sqsubseteq \mathcal{P}$. Based



(a) Homogeneous graph (b) Compress example

Figure 5: An example of MKC compression.

on Lemma 7, we have $V_{\mathcal{P}}[k+1] \subseteq V_{\mathcal{P}}[k]$. Thus, it is obvious that

$$V_{\mathcal{P}'}[k+1] \subseteq V_{\mathcal{P}}[k+1] \subseteq V_{\mathcal{P}}[k], \tag{1}$$

and

$$V_{\mathcal{P}'}[k+1] \subseteq V_{\mathcal{P}'}[k] \subseteq V_{\mathcal{P}}[k]. \tag{2}$$

To apply MKC, we use the meta-path tree as the skeleton of our CM-tree. For each CM-tree node $\mathcal{T}_{\mathcal{P}}$, we store a hash map, whose keys are the core numbers and the values are the sets of vertices mapped to corresponding core numbers. By Eq. (1) and (2), we find that $V_{\mathcal{P}}[k]$ always contains $V_{\mathcal{P}}[k+1]$ and $V_{\mathcal{P}'}[k]$. To reduce the space cost, we let the set mapped to core number k in node $\mathcal{T}_{\mathcal{P}}$ be:

$$\mathcal{T}_{\mathcal{P}}.map[k] = V_{\mathcal{P}}[k] \setminus (V_{\mathcal{P}}[k+1] \cup V_{\mathcal{P}'}[k]). \tag{3}$$

We further illustrate MKC by Example 5 with Figures 4 and 5.

Example 5. *Figure 5(a) shows two $\mathcal{P}$-graphs induced by meta-paths $\mathcal{P}_1$ and $\mathcal{P}_2$, where $\mathcal{P}_1 \sqsubseteq \mathcal{P}_2$. Obviously, the $\mathcal{P}_1$-graph is the subgraph of $\mathcal{P}_2$-graph. Figure 5(b) presents the MKC compression process of these two graphs. Since the $(3,\mathcal{P}_1)$-core is the subset of other five cores, we put vertices of this core to $\mathcal{T}_{\mathcal{P}_1}.map[3]$ first. Then for both $(2,\mathcal{P}_1)$-core and $(3,\mathcal{P}_2)$-core, they contain all vertices in $(3,\mathcal{P}_1)$-core. For $\mathcal{T}_{\mathcal{P}_1}.map[2]$ and $\mathcal{T}_{\mathcal{P}_2}.map[3]$, we only store $\{e\}$ since e is the only vertex that is not in the $(3,\mathcal{P}_1)$-core but contained in the other two cores. Other CM-tree nodes can be built similarly.*

Besides the hash map, each CM-tree node $\mathcal{T}_{\mathcal{P}}$ also needs to store its parent and child list. To summarize, $\mathcal{T}_{\mathcal{P}}$ has three elements:

• *map*: it maps core numbers to corresponding sets of vertices;
• *parent*: the parent node of $\mathcal{T}_{\mathcal{P}}$;
• *childList*: a list of child nodes of $\mathcal{T}_{\mathcal{P}}$.

Figure 4(b) depicts the CM-tree index for all the three homogeneous graphs in Figure 3, and its skeleton is the meta-path tree in Figure 4(a). Based on the CM-tree, the following key operations used by our query algorithms (Section 4.3), can be performed efficiently.

• **Core finding.** Given a positive integer k and a meta-path $\mathcal{P}$, find a set of vertices that belong to $(k,\mathcal{P})$-cores.

• **Meta-path search.** Given a meta-path $\mathcal{P}$, find the nested meta-path of $\mathcal{P}$, or the list of meta-paths within which $\mathcal{P}$ is nested.

4.2 Index construction

We now discuss the index construction algorithm. Since the meta-path tree serves as the skeleton of the CM-tree, we first build a meta-path tree according to the nested relationships of meta-paths in the HIN schema. Then, we build all CM-tree nodes starting from the root of the meta-path tree following the depth-first search (DFS) order, as the hash map construction in the child node depends on the result of the parent node according to Eq. (3).

Algorithm 4 presents the CM-tree construction algorithm. We first obtain the set $\mathcal{X}$ of all valid meta-paths (line 3). Then we pick the shortest meta-path $\mathcal{P}$ from $\mathcal{X}$ (line 4), and build the meta-path tree starting from $\mathcal{P}$ by calling `BuildMetaPathTree` (line 6). Finally, we invoke `BuildCMnode` to build each node of the CM-tree following the DFS order (line 7).

Algorithm 4: CM-tree construction algorithm

```
1  Function BuildCMtree(H):
2      T ← ∅;
3      invoke GenMetaPaths to generate meta-path set X;
4      P ← the shortest meta-path in X;
5      build a tree node T_P;
6      BuildMetaPathTree(T, X, P);
7      BuildCMnode(T_P, H);
8      return T;
9  Function BuildMetaPathTree(T, X, P):
10     foreach P' ∈ X do
11         if P ⊑ P' and |P'| = |P| + 2 then
12             build a tree node T_P';
13             T_P.childList.add(T_P'), T_P'.parent ← T_P;
14             BuildMetaPathTree(T, X, P');
15 Function BuildCMnode(T_P, H):
16     induce H_P from H;
17     core_P[ ] ← k-core decomposition on H_P;
18     foreach v ∈ H_P do T_P.map[core_P[v]].add(v) ;
19     foreach k ∈ T_P.map.keyset() do
20         T_P.map[k] ← T_P.map[k] \ ⋃_{P'⊑P} T_P'.map[k];
21     foreach T_P' ∈ T_P.childList do BuildCMnode(T_P', H) ;
```

In `BuildMetaPathTree`, we build the meta-path tree by following the DFS order. In each invoking of `BuildMetaPathTree`, we traverse $\mathcal{X}$ to find meta-paths $\mathcal{P}'$ such that $\mathcal{P} \sqsubseteq \mathcal{P}'$ and $|\mathcal{P}'| = |\mathcal{P}| + 2$ (lines 10-14). For each $\mathcal{P}'$, we build a new index node $\mathcal{T}_{\mathcal{P}'}$ (line 12), add $\mathcal{T}_{\mathcal{P}'}$ to the child list of $\mathcal{T}_{\mathcal{P}}$, let $\mathcal{T}_{\mathcal{P}}$ be the parent node of $\mathcal{T}_{\mathcal{P}'}$ (line 13) and recursively build the child nodes of $\mathcal{T}_{\mathcal{P}}$ (line 14).

In `BuildCMnode`, we build each CM-tree node by following the DFS order from the root of the meta-path tree. For every node, we first derive the P-graph $\mathcal{H}_{\mathcal{P}}$ (line 16), and then do k-core decomposition using the linear algorithm [1] to get the core numbers of all vertices $core_{\mathcal{P}}[\,]$ (line 17). Next, we traverse all vertices in $\mathcal{H}_{\mathcal{P}}$ and add v to the hash map $\mathcal{T}_{\mathcal{P}}.map$ according to its core number $core_{\mathcal{P}}[v]$ (line 18). For each core number k in the keys of $\mathcal{T}_{\mathcal{P}}.map$, we also need to delete vertices that are in the union set of $\mathcal{T}_{\mathcal{P}'}.map[k]$, where $\mathcal{P}' \sqsubseteq \mathcal{P}$ to avoid redundancy (lines 19-20). After building $\mathcal{T}_{\mathcal{P}}$, we traverse the child list of the current node and call `BuildCMnode` to build child nodes recursively (line 21).

Lemma 8. *Given a star-schema HIN $\mathcal{H}$ and a set of meta-paths $\mathcal{X} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{|\mathcal{X}|}\}$, the CM-tree takes $O(\sum_{i=1}^{|\mathcal{X}|} t_i)$ space cost, where t_i is the total number of vertices with the type linked by $\mathcal{P}_i$.*

Proof. If only KC is used, then for each $\mathcal{P}_i$, $\mathcal{T}_{\mathcal{P}_i}$ stores t_i vertices. For MKC, according to Eq. (3), each $\mathcal{T}_{\mathcal{P}_i}$ stores less than t_i vertices when $\mathcal{T}_{\mathcal{P}_i}$ is not the root node of the index, so Lemma 8 holds. □

Lemma 9. *Algorithm 4 takes $O(\sum_{i=1}^{|\Psi|}(n_1 \cdot b_{1,2} + n_1 \sum_{i=2}^{L} n_i \cdot b_{i,i+1}))$ time.*

Proof. It takes $O(\sum_{i=1}^{|\Psi|}(n_1 \cdot b_{1,2} + n_1 \sum_{i=2}^{L} n_i \cdot b_{i,i+1}))$ time to compute all $\mathcal{H}_{\mathcal{P}_i}$ ($\mathcal{P}_i \in \Psi$). And computing a k-core from a homogeneous graph costs linear time [1, 38]. Hence, Lemma 9 holds. □

4.3 Index-based query algorithm

With CM-tree index, we propose a fast index-based query algorithm to solve the CSSH problem based on the fast algorithm of computing Ψ-NMC (`FastNMC` in Section 3.3). Specifically, we first introduce a novel concept of *nested meta-path set*. Then, we design pruning and early stop strategies to reduce the number of candidates to be verified. Further, to verify each meta-path set, we use the CM-tree to quickly locate the Ψ-NMC in a small subset of vertices. Next, we first introduce the concept of nested meta-path set.

Definition 9. ***Nested meta-path set.*** *Given a set of n meta-paths $\Psi = \{\mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_n\}$, the nested meta-path set of Ψ is another meta-path set $\Psi' = (\mathcal{P}_1', \mathcal{P}_2', \cdots, \mathcal{P}_n')$, s.t. $\forall \mathcal{P} \in \Psi$, there exists $\mathcal{P}' \in \Psi'$ such that $\mathcal{P}' \sqsubseteq \mathcal{P}$.*

For example, let $\mathcal{P}$=(APA), $\mathcal{P}_1$=($APVPA$), and $\mathcal{P}_2$=($APTPA$). Then, $\{\mathcal{P}\}$ is a nested meta-path set of both $\{\mathcal{P}_1\}$ and $\{\mathcal{P}_2\}$. Next, we show that the nested relationships also exists between a meta-path set and its nested meta-path set.

Lemma 10. *Given a star-schema HIN $\mathcal{H}$, a positive integer k, a meta-path set Ψ and its nested meta-path set Ψ', if there exists a Ψ'-NMC, there must exist a Ψ-NMC, s.t. Ψ'-NMC $\subseteq$ Ψ-NMC.*

Proof. If there exists a Ψ'-NMC, then for each $\mathcal{P}'$ in Ψ', vertices in the Ψ'-NMC can form a $\mathbf{B}_{\mathcal{P}'}$. By Lemma 1 and Definition 9, for each $\mathcal{P}$ in Ψ, there exists a meta-path $\mathcal{P}' \in \Psi'$ and $\mathcal{P}' \sqsubseteq \mathcal{P}$, such that the Ψ'-NMC (and also a $(k, \mathcal{P}')$-core) is contained by a $(k, \mathcal{P})$-core. Thus, there exists a Ψ-NMC s.t. Ψ'-NMC $\subseteq$ Ψ-NMC. □

• **Index-based pruning for candidates of meta-path sets.** Recall that in Algorithm 1, `GenCan` generates size-$(c+1)$ meta-path set candidates only based on valid size-c sets, but there are still too many size-$(c+1)$ candidates to be verified. We show that many of them can be pruned by exploiting Lemma 10. Specifically, given two meta-path sets Ψ and Ψ' s.t. Ψ' is a nested meta-path set of Ψ, if we have verified that Ψ'-NMC exists, then Ψ-NMC must exist as well by Lemma 10. Hence, after verifying the existence of Ψ'-NMC, we can get all meta-path sets Ψ that Ψ' is the nested meta-path set of Ψ and mark them as valid meta-path set candidates. In this way, we can reduce the number of candidates to be verified.

In order to generate all meta-path sets Ψ, of which Ψ' is the nested meta-path set, we design `GenValidCan` based on the CM-tree. Specifically, for each $\mathcal{P}' \in \Psi'$, we get all $\mathcal{P}$ s.t. $\mathcal{P}' \sqsubseteq \mathcal{P}$ by searching

the CM-tree. Then, we select a meta-path from $\{\mathcal{P}|\mathcal{P}' \sqsubseteq \mathcal{P}\}$ for each $\mathcal{P}'$ in Ψ', and combine them to get a meta-path set. The detail of GenValidCan are in the appendix of technical report [25].

• **Optimal candidate verification order.** By Lemma 10, some candidate meta-path sets can be pruned after their nested sets are verified. As a result, the order to verify candidates affects the number of candidates to be verified. To find the optimal order for reducing the time cost, we introduce Lemma 11.

Lemma 11. *Given several meta-path set candidates (Ψ_1, Ψ_2, $\cdots$, Ψ_n), each of them has the same size, verifying each candidate set Ψ_i in ascending order by the length sum of all meta-paths in Ψ_i minimizes the number of candidates that need to be verified.*

Proof. Based on the definition of nested meta-path set, for a meta-path set Ψ and its nested meta-path set Ψ', the length sum of meta-paths in Ψ' is less than that of Ψ, since $|\Psi|=|\Psi'|$ and for any meta-path $\mathcal{P}$, it has the same or smaller length than its nested meta-path. Then we conclude that in a list of same size candidate sets of meta-paths, for any two candidate sets Ψ_i and Ψ_j, if Ψ_i is the nested meta-path set of Ψ_j, the length sum of meta-paths in Ψ_i must be smaller than the length sum of meta-paths in Ψ_j. Moreover, by Lemma 10, if Ψ_i is a valid meta-path set, then Ψ_j is a valid meta-path set without being verified. Hence, Lemma 11 holds. □

• **Early stop strategy.** We further boost the efficiency by introducing an early stop condition in Lemma 12.

Lemma 12. *Given all meta-path sets with the same size Ψ_1, Ψ_2, $\cdots$, Ψ_n, if there exists a Ψ_i-NMC that Ψ_i ($1 \le i \le n$) is the nested meta-path set of all Ψ_j ($1 \le j \le n$ and $j \ne i$), then Ψ_i-NMC is the community we need.*

Proof. According to the property of *set non-nestedness* of meta-paths, if Ψ-NMC is the result community, then there does not exist a Ψ'-NMC s.t.$\Psi' = (\Psi \cup \{\mathcal{P}'\}) \setminus \{\mathcal{P}\}$ where $\mathcal{P}' \sqsubseteq \mathcal{P}$. Thus, if there exists a Ψ_i-NMC that Ψ_i is the nested meta-path set of all Ψ_j ($1 \le j \le n$ and $j \ne i$), then even if Ψ_j-NMC exists, it is not the result community because of **set non-nestedness**. For the meta-path sets with larger size generated from Ψ_i, there does not exist a result community for the same reason. Hence, Lemma 12 holds. □

• **Index-based Ψ-NMC computation.** To verify each candidate meta-path set Ψ, we need to compute the Ψ-NMC. With CM-tree, we can locate the core in a small set by intersecting the $(k, \mathcal{P})$-cores where $\mathcal{P} \in \Psi$, according to the following corollary.

Corollary 4.1. *Given a star-schema HIN $\mathcal{H}$, a positive integer k, and a set Ψ of meta-paths, we have*

$$\Psi\text{-}NMC \subseteq \bigcap_{\mathcal{P}\in\Psi} \mathbf{B}_{\mathcal{P}}. \tag{4}$$

Proof. The conclusion directly follows Lemma 5. □

In summary, we propose an index-based query algorithm, following the two-step framework (verification and candidate generation). We first generate the meta-path set $\mathcal{X}$ (line 1) and initialize c=1, a hash map $\mathcal{M}_c$ to store the corresponding community of every Ψ (line 2). $\mathcal{S}_c$ is initialized, in which each meta-path set contains a meta-path in $\mathcal{X}$ (line 3). Then in the loop, we sort every $\Psi \in \mathcal{S}_c$ in ascending order by the length sum of all meta-paths in Ψ (line 5), based on Lemma 11. Next, we initialize a set $\mathcal{S}_c^*$ to store the valid size-c meta-path sets (line 6). For each $\Psi \in \mathcal{S}_c$, we verify the existence of Ψ-NMC by IndexNMC (line 8). Specifically, for each $\mathcal{P} \in \Psi$, IndexNMC gets a $\mathbf{B}_{\mathcal{P}}$ with the help of CM-tree, computes the intersection set of all $\mathbf{B}_{\mathcal{P}}$, and calls function FastNMC (Algorithm 3) to compute Ψ-NMC. The details of IndexNMC are in the appendix of the technical report [25]. If Ψ-NMC exists, we use $\mathcal{M}_c$ to record Ψ and its corresponding community (line 10), and call function GenValidCan to generate a set of valid meta-path sets $\mathcal{Y}$ based on Lemma 10 (line 11) to reduce the number of $\mathbf{B}_\Psi$ that needs to be computed. If Ψ satisfies Lemma 12, then Ψ-NMC is the only result community and we finish (line 12). To reduce unnecessary verification of candidates, we add all meta-path sets in $\mathcal{Y}$ into $\mathcal{S}_c^*$ and remove them from $\mathcal{S}_c$ (lines 13-14). After all candidates in $\mathcal{S}_c$ are checked, we generate new size-(c+1) meta-path sets via GenCan (line 15), and c is increased by 1. The whole loop is repeated until there is no new candidate to be verified in $\mathcal{S}_c$ (line 17). Finally, we output all Ψ-NMCs in $\mathcal{M}_{c-1}$ (line 18) as the communities.

Algorithm 5: Index-based query algorithm

```
   Input: H, T, Q, k
   Output: Ψ-NMC with the maximum |Ψ|
1  invoke GenMetaPaths to generate meta-paths set X;
2  initialize c ← 1, M_c ← ∅;
3  initialize S_c ← {{P}|P ∈ X};
4  repeat
5      sort all Ψ in S_c in ascending order of Σ_{P∈Ψ} |P|;
6      S_c^* ← ∅;
7      foreach Ψ ∈ S_c do
8          Ψ-NMC ← IndexNMC(T, k, Ψ, Q);
9          if Ψ-NMC exists then
10             M_c.put(Ψ, B_Ψ);
11             Y ← GenValidCan(Ψ, T);
12             if Y = S_c then output Ψ-NMC, return;
13             S_c^* ← S_c^* ∪ Y;
14             S_c ← S_c \ Y;
15     S_{c+1} ← GenCan(S_c^*);
16     c ← c + 1;
17 until S_c = ∅;
18 foreach Ψ-NMC ∈ M_{c-1} do output Ψ-NMC ;
```

Lemma 13. *Algorithm 5 takes $O(\sum_{i=1}^{c}\sum_{j=1}^{|\mathcal{M}_i|}(|\Psi_j| \cdot (\frac{1}{2}\cdot L \cdot n_1 + \sum_{k=1}^{L} n_k \cdot b_{k,k+1}) + \Delta_j))$ time, where Δ_j denotes the time cost of FastNMC (Algorithm 3).*

Proof. In Algorithm 5, it calls $\sum_{i=1}^{c}|\mathcal{M}_i|$ times IndexNMC and IndexNMC cost $O(|\Psi_j| \cdot (\frac{1}{2}\cdot L \cdot n_1 + \sum_{k=1}^{L} n_k \cdot b_{k,k+1}) + \Delta_j)$ time for each Ψ_j. The details of IndexNMC are in the appendix of the technical report [25]. Hence, Lemma 13 holds. □

5 EXPERIMENTS

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

Table 2: Datasets used in our experiments.

Dataset	Vertices	Edges	Vertex types	Edge types	Meta-paths
PubMed	14,256	33,556	4	3	12
IMDB	854,616	3,898,144	4	3	12
DBLP	2,056,444	6,607,065	4	3	11
Foursquare	4,472,122	10,200,000	4	3	8

5.1 Setup

Datasets. We use four real star-schema HINs: *PubMed*[1] [46], *IMDB*[2], *DBLP*[3] [44], and *Foursquare*[4] [47, 48]. Their statistics, including the numbers of vertices, edges, vertex types, and edge types, are reported in Table 2. *PubMed* is a network of genes, diseases, chemicals, and species, constructed from PubMed[5]. *IMDB* contains the movie rating records since 2000, and it has four types of vertices (authors, directors, writers and movies). *DBLP* includes publication records in computer science areas, and the vertex types are authors, papers, venues and topics. *Foursquare* contains the check-in records in US, which has four types of vertices, including records, users, venues, and categories.

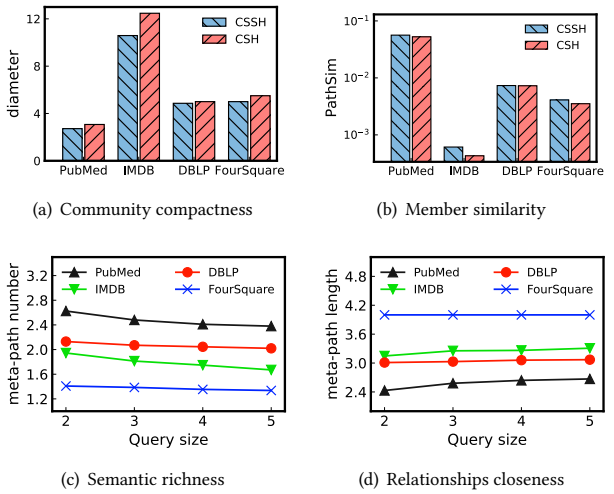
Queries. For each dataset, we collect a set of meta-paths and its size is reported in Table 2. Note that in line with existing works [14, 23, 40], we collect all the meta-paths with lengths at most four. We generate 200 queries for each dataset. To generate a query, we randomly select a meta-path and then select several vertices with core numbers of 6 or more following [14], where the number of selected vertices $|Q|$ varies from 2 to 5 with 2 as the default value, which ensures that for every vertex in a query, there exists a meaningful community containing it. By default, we set the value of k to 6 [14]. In the following reported results, each data point is the average result for these 200 queries unless otherwise specified. We implement all the algorithms in Java and run experiments on a machine having an Intel(R) Xeon(R) Gold 6226R 2.90GHz CPU and 256GB of memory, with Ubuntu installed.

5.2 Effectiveness evaluation

• **Community compactness.** To measure the community compactness of communities, a commonly-used metric is the diameter [14, 22]. To adapt it for communities in HINs, we use $\mathcal{P}$-distance [14], which is the minimum number of path instances of $\mathcal{P}$ for linking two vertices (e.g., the $\mathcal{P}$-distance between two vertices linked by an instance of $\mathcal{P}$ is 1). We also compare CSSH query with CSH query [14]. Specifically, we first run each CSSH query and get the community with a set Ψ of shared meta-paths. Then, for each $\mathcal{P} \in \Psi$, we run the CSH query to get the community with the same values of Q and k.

We depict the average diameters for communities of CSSH and CSH queries on four datasets in Figure 6(a). Clearly, the communities of CSSH queries have smaller diameters than those of CSH queries, so our Ψ-NMC-based communities are more structurally compact and their vertices tend to have closer relationships.

[1]https://github.com/yangji9181/HNE
[2]https://www.imdb.com/interfaces/
[3]https://www.aminer.cn/citation
[4]https://sites.google.com/site/yangdingqi/home/foursquare-dataset
[5]https://www.ncbi.nlm.nih.gov/pubmed/



(a) Community compactness (b) Member similarity

(c) Semantic richness (d) Relationships closeness

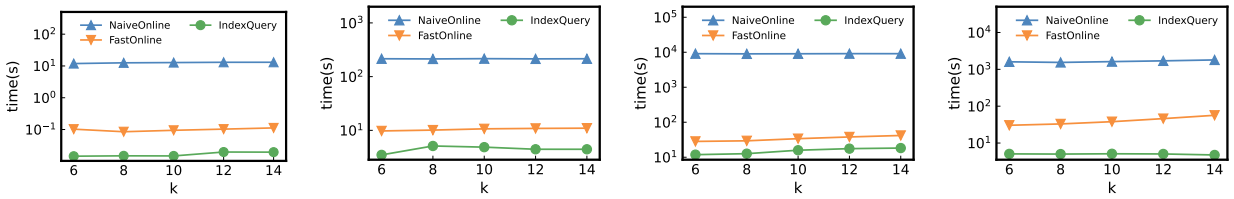
Figure 6: Results of effectiveness evaluation.

• **Similarity of community members.** We measure the similarity of community members by PathSim [40]. Specifically, we first find communities by CSH and CSSH queries, where the settings of CSSH queries are similar to those in the experiment above, and then compute the PathSim value for each pair of vertices in these communities, respectively. Figure 6(b) shows the average PathSim values on four datasets. Clearly, communities based on our Ψ-NMC achieve higher similarity values than those based on $(k, \mathcal{P})$-core in the CSH queries, so our CSSH query is better to capture the similarity among community members.

• **Semantic richness.** To measure the semantic richness of the community, we count the number of meta-paths that the community shares, by varying the size of the query vertex set (i.e., $|Q|$). Notice that in this experiment, the number of meta-paths that the community shares is not simply the size of Ψ; instead, for a community Ψ-NMC, we count the number of all meta-paths in Ψ, and also the meta-paths whose nested meta-paths are in Ψ. For example, on DBLP, if Ψ=$\{APA\}$, then the number of meta-paths we count is 3, since $\{APA\}$ is nested in $\{APVPA\}$ and $\{APTPA\}$. We report the average results in Figure 6(c). Generally, as $|Q|$ becomes larger, the number of shared meta-paths decreases, because a larger $|Q|$ means that fewer $(k, \mathcal{P})$-cores can contain its vertices.

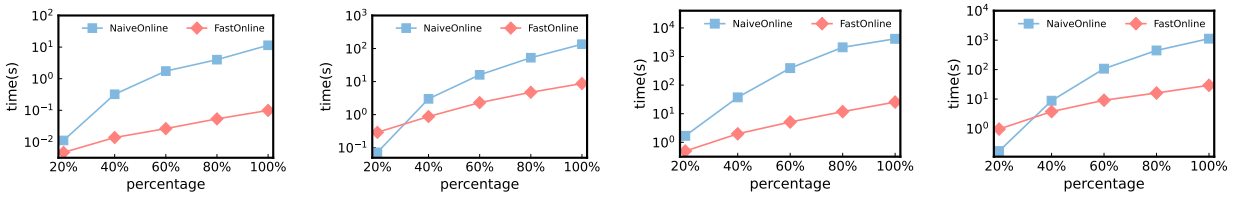
• **Relationships closeness.** To measure the relationships closeness of members of the community, we compute the average length of meta-paths in the meta-path set Ψ, by varying $|Q|$. We report the average results in Figure 6(d). Clearly, as $|Q|$ becomes larger, the average length of meta-paths increases, meaning that the relationships closeness becomes weak. This is because for a larger $|Q|$, we need longer meta-paths to form Ψ-NMCs containing Q.

• **A case study.** We run a CSSH query and two CSH queries [14] on a small DBLP network with 50,663 vertices and 88,986 edges (randomly extracted from the original network). In the CSSH query, we set query vertex set Q = {Jiawei Han, Jeffrey Xu Yu, Yizhou Sun} and k=4. We obtain a community where $\Psi = \{\mathcal{P}_1, \mathcal{P}_2\}$, where $\mathcal{P}_1 = (APVPA)$ and $\mathcal{P}_2 = (APTPA)$. We draw it in Figure 9, where a



(a) PubMed (b) IMDB (c) DBLP (d) FourSquare

Figure 7: Efficiency results of three query algorithms.



(a) PubMed (b) IMDB (c) DBLP (d) FourSquare

Figure 8: Scalability test for online query algorithms.

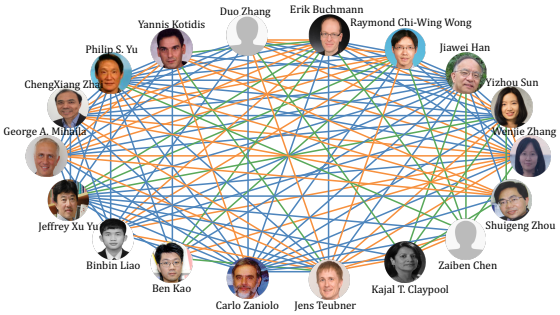


Figure 9: A community with Q={Jiawei Han, Jeffrey Xu Yu, Yizhou Sun} and k=4, where Ψ={$(APVPA)$, $(APTPA)$}.

Table 3: Statistics of a case study on a small DBLP network.

Community	Members	PathSim	Diameter
Ψ-NMC	18	0.153	4
$(k, \mathcal{P}_1)$-core	96	0.0830	5
$(k, \mathcal{P}_2)$-core	2,118	0.0142	7

yellow edge and a green edge mean two vertices linked by instances of $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively, and a blue edge means two vertices linked by instances of both $\mathcal{P}_1$ and $\mathcal{P}_2$. In two CSH queries, we use the same query vertex set and specify the query meta-path as $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively. Since their returned communities are much larger, we do not draw them. Instead, we report the statistics of three communities in Table 3. Clearly, our CSSH community has the smallest community size as it needs to share the semantic relationships of both $\mathcal{P}_1$ and $\mathcal{P}_2$. Besides, our CSSH community has the largest value of PathSim and smallest diameter, so it is more compact and has higher community member similarity.

5.3 Efficiency evaluation

We now evaluate efficiency of both online and index-based query algorithms, which are denoted by `NaiveOnline`, `FastOnline`, and `IndexQuery`. In particular, the naive online algorithm `NaiveOnline` and fast online algorithm `FastOnline` use `HomNMC` (Algorithm 2) and `FastNMC` (Algorithm 3) to compute Ψ-NMC respectively.

• **Online and index-based query algorithms.** We show the efficiency results of query algorithms by varying k in Figure 7. Clearly, `FastOnline` is up to two orders of magnitude faster than `NaiveOnline`, since for each vertex with the target type, `HomNMC` finds all its $\mathcal{P}$-neighbors, while `FastNMC` only finds a small number of them. Meanwhile, as k becomes larger, the running time of `FastOnline` increases since a larger k means finding more $\mathcal{P}$-neighbors, while the running time of `NaiveOnline` remains almost stable as the main overhead comes from building the homogeneous graphs. In addition, the index-based query algorithms are more than two orders of magnitude faster than `NaiveOnline` and up to one order of magnitude faster than `FastOnline`, indicating that our CM-tree is effective for facilitating the queries.

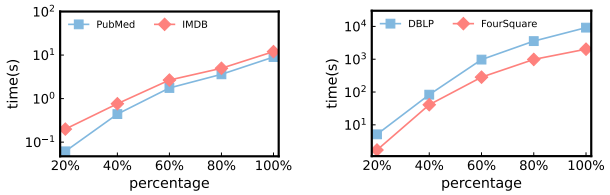
• **Scalability test.** For each dataset, we randomly select 20%, 40%, 60%, 80% and 100% of its vertices and obtain five subgraphs induced by these vertices respectively. Then, we run CSSH queries using two online query algorithms and report the average efficiency results in Figure 8. Generally, their time cost scales linearly with the number of vertices in the graph, showing good scalability.

• **Index space cost analysis.** Recall that we propose three index compression strategies (i.e., MC, KC and MKC). To measure their compression effectiveness, we adapt our CM-tree construction algorithm by using three strategies respectively, and then count the total number of vertices stored in all index nodes. Table 4 shows the results on all datasets, where L is the maximum length of all

Table 4: Results of index compression analysis.

Dataset	L (length)	MKC	KC	MC
PubMed	2	5,348	5,348	331,706
	4	15,712	16,044	3,661,457
IMDB	2	54,881	54,881	87,880
	4	132,287	164,643	3,393,559
DBLP	2	785,104	785,104	3,891,749
	4	2,348,595	2,355,312	7,647,951,943
FourSquare	2	129,039	129,039	129,039
	4	382,357	387,117	588,368,051

meta-paths. Clearly, MKC is the most effective one for reducing the space cost of the CM-tree.



(a) PubMed and IMDB (b) DBLP and FourSquare

Figure 10: Efficiency results of index construction.

• **Index construction time analysis.** To evaluate the index construction algorithm, for each dataset, we randomly select 20%, 40%, 60%, 80% and 100% of its vertices and obtain five subgraphs induced by these vertices respectively. Afterwards, for each dataset, we set the max length of meta-path L=4 and randomly select a target type, and report the efficiency results in Figure 10. We can observe that the time cost is almost linear to the size of HINs, and thus our index construction algorithm scales well on large HINs.

6 RELATED WORK

The problems of network community retrieval contain two main streams: community detection (CD) and community search (CS).

Community detection (CD). Earlier solutions [15, 31] mainly employ link-based analysis to detect these communities. However, most of them focus on homogeneous graphs, where vertices are of the same type [12, 13, 32]. Some recent works [35, 36, 39, 41–43, 56] focus on generating clusters/communities in HINs, which can roughly be grouped into two classes according to vertex types in the communities. The first class [4, 36, 39, 43] focuses on detecting clusters, each of which contains objects with multiple types, while the second class [41, 42, 56] aims to generate clusters of objects with a specific type. In [41], Sun et al. proposed an algorithm to generate clusters of a specific type of objects; in [42], a user-guided algorithm is developed to cluster objects of a target type.

Community search (CS). CS aims to query densely connected subgraphs containing a specific vertex or a set of vertices [8, 13, 14, 38, 49]. To measure the structure cohesiveness of a community, people often use some cohesive subgraph models [13], like k-core [1, 2], k-truss [6, 53], k-clique [7, 51] and k-edge connected component [3, 18, 19]. The k-core [1, 2], which requires each vertex having at least k neighbors within the community, is the most frequently used one. For example, in [8], Cui et al. designed a local search CS algorithm; in [12], Fang et al. used k-core for CS on attributed graphs. Another group of CS works is based on the k-truss [6, 53]. For example, in [20, 22], the k-truss-based community search is studied; in [5, 21], Huang et al. studied CS using k-truss on attributed graphs. A recent survey of CS can be found in [13].

While CS has been extensively studied, most of existing works focus on conventional homogeneous networks and little attention has been paid to the problem for HINs. Recently, researchers have attempted to study CS over HINs [10, 14, 16, 24, 45, 49, 55]. In [14], Fang et al. introduced a novel core model on HIN, which focuses on a specific type of vertices and requires that each vertex is linked to at least k other vertices with the same type via meta-paths. In [24], Jian et al. proposed the relational constraint that allows the user to specify fine-grained connection requirements between vertices and studied the problem of relational community search over HINs, where the community involves nodes of multiple types. However, all these existing studies suffer from several limitations, e.g., they either require users to specify a meta-path or relational constraints, which pose great challenges to users who are not familiar with HINs. Thus, how to effectively perform CS over HINs without specifying these constraints is still an open question. To solve this problem, in this paper we propose to study CS over the HIN with star-schema, which is a representative type of HINs, without asking users to specify these constraints, and also develop efficient solutions.

7 CONCLUSIONS

In this paper, we study the problem of Community Search over Star-schema HINs (or CSSH problem in short), which aims to search the most-likely community containing a set of query vertices Q from a star-schema HIN, without specifying the parameters like meta-paths and relational constraints. To model the community that can well capture the rich semantic relationships carried by query vertices Q, we find the set of vertices under the meta-path-based core model, by maximizing the set of shared meta-paths satisfying the property of non-nestedness. We first develop efficient online query algorithms. We further boost the query efficiency by designing a novel compact index structure and an index-based query algorithm. Our experimental results on four real large star-schema HINs show that the proposed solutions are effective and efficient for searching communities over large HINs. In the future, we will study how to efficiently maintain the index since many HINs are dynamic, and also attempt to use other HIN cohesive subgraph models (e.g., clique-based models [17, 52]) to formulate community models over other kinds of HINs, which are more general.

ACKNOWLEDGMENTS

Chunshan Li was supported by the National Key Research and Development Program of China (No.2018YFB1700400), NSFC under Grant 61902090, and the Major Scientific and Technological Innovation Project of Shandong Province of China (2021ZLGX05, 2020CXGC010705). Yixiang Fang was supported by NSFC under Grant 62102341, and Basic and Applied Basic Research Fund in Guangdong Province under Grant 2022A1515010166. Xin Cao was supported by ARC DE190100663.

REFERENCES

[1] Vladimir Batagelj and Matjaz Zaversnik. 2003. An O (m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049* (2003).

[2] Francesco Bonchi, Arijit Khan, and Lorenzo Severini. 2019. Distance-generalized core decomposition. In *Proceedings of the 2019 International Conference on Management of Data*. 1006–1023.

[3] Lijun Chang, Xuemin Lin, Lu Qin, Jeffrey Xu Yu, and Wenjie Zhang. 2015. Index-based optimal algorithms for computing steiner components with maximum connectivity. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 459–474.

[4] Lu Chen, Yunjun Gao, Yuanliang Zhang, Christian S Jensen, and Bolong Zheng. 2019. Efficient and incremental clustering algorithms on star-schema heterogeneous graphs. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 256–267.

[5] Lu Chen, Chengfei Liu, Rui Zhou, Jianxin Li, Xiaochun Yang, and Bin Wang. 2018. Maximum co-located community search in large scale social networks. *Proceedings of the VLDB Endowment* 11, 10 (2018), 1233–1246.

[6] Jonathan Cohen. 2008. Trusses: Cohesive subgraphs for social network analysis. *National security agency technical report* 16, 3.1 (2008).

[7] Wanyun Cui, Yanghua Xiao, Haixun Wang, Yiqi Lu, and Wei Wang. 2013. Online search of overlapping communities. In *Proceedings of the 2013 ACM SIGMOD international conference on Management of data*. 277–288.

[8] Wanyun Cui, Yanghua Xiao, Haixun Wang, and Wei Wang. 2014. Local search of communities in large graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 991–1002.

[9] Yuxiao Dong, Nitesh V. Chawla, and Ananthram Swami. 2017. Metapath2vec: Scalable Representation Learning for Heterogeneous Networks *(KDD '17)*. Association for Computing Machinery, New York, NY, USA, 135–144. https://doi.org/10.1145/3097983.3098036

[10] Zheng Dong, Xin Huang, Guorui Yuan, Hengshu Zhu, and Hui Xiong. 2021. Butterfly-core community search over labeled graphs. *arXiv preprint arXiv:2105.08628* (2021).

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

[12] Yixiang Fang, Reynold Cheng, Siqiang Luo, and Jiafeng Hu. 2016. Effective community search for large attributed graphs. *Proceedings of the VLDB Endowment* 9, 12 (2016), 1233–1244.

[13] Yixiang Fang, Xin Huang, Lu Qin, Ying Zhang, Wenjie Zhang, Reynold Cheng, and Xuemin Lin. 2020. A survey of community search over big graphs. *The VLDB Journal* 29, 1 (2020), 353–392.

[14] Yixiang Fang, Yixing Yang, Wenjie Zhang, Xuemin Lin, and Xin Cao. 2020. Effective and efficient community search over large heterogeneous information networks. *Proceedings of the VLDB Endowment* 13, 6 (2020), 854–867.

[15] Santo Fortunato. 2010. Community detection in graphs. *Physics reports* 486, 3-5 (2010), 75–174.

[16] Edoardo Galimberti, Francesco Bonchi, and Francesco Gullo. 2017. Core decomposition and densest subgraph in multilayer networks. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 1807–1816.

[17] Jiafeng Hu, Reynold Cheng, Kevin Chen-Chuan Chang, Aravind Sankar, Yixiang Fang, and Brian YH Lam. 2019. Discovering maximal motif cliques in large heterogeneous information networks. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 746–757.

[18] Jiafeng Hu, Xiaowei Wu, Reynold Cheng, Siqiang Luo, and Yixiang Fang. 2016. Querying minimal steiner maximum-connected subgraphs in large graphs. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. 1241–1250.

[19] Jiafeng Hu, Xiaowei Wu, Reynold Cheng, Siqiang Luo, and Yixiang Fang. 2017. On minimal steiner maximum-connected subgraph queries. *IEEE Transactions on Knowledge and Data Engineering* 29, 11 (2017), 2455–2469.

[20] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying k-truss community in large and dynamic graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 1311–1322.

[21] Xin Huang and Laks VS Lakshmanan. 2017. Attribute-driven community search. *Proceedings of the VLDB Endowment* 10, 9 (2017), 949–960.

[22] Xin Huang, Laks VS Lakshmanan, Jeffrey Xu Yu, and Hong Cheng. 2015. Approximate closest community search in networks. *arXiv preprint arXiv:1505.05956* (2015).

[23] Zhipeng Huang, Yudian Zheng, Reynold Cheng, Yizhou Sun, Nikos Mamoulis, and Xiang Li. 2016. Meta structure: Computing relevance in large heterogeneous information networks. In *Proceedings of the 22nd ACM SIGKDD International conference on knowledge discovery and data mining*. 1595–1604.

[24] Xun Jian, Yue Wang, and Lei Chen. 2020. Effective and efficient relational community detection and search in large dynamic heterogeneous information networks. *Proceedings of the VLDB Endowment* 13, 10 (2020), 1723–1736.

[25] Yangqin Jiang, Yixiang Fang, Chenhao Ma, Xin Cao, and Chunshan Li. 2022. Effective community search over large star-schema heterogeneous information networks (technical report). *https://github.com/ZzMeei/CS-StarSchemaHIN/blob/main/main.pdf* (2022).

[26] Linhao Luo, Yixiang Fang, Xin Cao, Xiaofeng Zhang, and Wenjie Zhang. 2021. *Detecting Communities from Heterogeneous Graphs: A Context Path-Based Graph Neural Network Model.* Association for Computing Machinery, New York, NY, USA, 1170–1180. https://doi-org.eproxy.lib.hku.hk/10.1145/3459637.3482250

[27] Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks VS Lakshmanan, and Xiaolin Han. 2022. A Convex-Programming Approach for Efficient Directed Densest Subgraph Discovery. In *SIGMOD*.

[28] Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks VS Lakshmanan, Wenjie Zhang, and Xuemin Lin. 2020. Efficient algorithms for densest subgraph discovery on large directed graphs. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1051–1066.

[29] Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks VS Lakshmanan, Wenjie Zhang, and Xuemin Lin. 2021. On Directed Densest Subgraph Discovery. *TODS* 46, 4 (2021), 1–45.

[30] Changping Meng, Reynold Cheng, Silviu Maniu, Pierre Senellart, and Wangda Zhang. 2015. Discovering meta-paths in large heterogeneous information networks. In *Proceedings of the 24th International Conference on World Wide Web*. 754–764.

[31] Mark EJ Newman and Michelle Girvan. 2004. Finding and evaluating community structure in networks. *Physical review E* 69, 2 (2004), 026113.

[32] You Peng, Song Bian, Rui Li, Sibo Wang, and Jeffrey Xu Yu. 2022. Finding Top-r Influential Communities under Aggregation Function. In *ICDE*. IEEE.

[33] Paola Pesántez-Cabrera and Ananth Kalyanaraman. 2017. Efficient detection of communities in biological bipartite networks. *IEEE/ACM transactions on computational biology and bioinformatics* 16, 1 (2017), 258–271.

[34] Chuan Shi, Xiangnan Kong, Philip S Yu, Sihong Xie, and Bin Wu. 2012. Relevance search in heterogeneous networks. In *Proceedings of the 15th international conference on extending database technology*. 180–191.

[35] Chuan Shi, Yitong Li, Jiawei Zhang, Yizhou Sun, and S Yu Philip. 2016. A survey of heterogeneous information network analysis. *IEEE Transactions on Knowledge and Data Engineering* 29, 1 (2016), 17–37.

[36] Chuan Shi, Ran Wang, Yitong Li, Philip S Yu, and Bin Wu. 2014. Ranking-based clustering on general heterogeneous information networks by network projection. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. 699–708.

[37] Chuan Shi, Chong Zhou, Xiangnan Kong, Philip S Yu, Gang Liu, and Bai Wang. 2012. Heterecom: a semantic-based recommendation system in heterogeneous networks. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1552–1555.

[38] Mauro Sozio and Aristides Gionis. 2010. The community-search problem and how to plan a successful cocktail party. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. 939–948.

[39] Yizhou Sun, Charu C Aggarwal, and Jiawei Han. 2012. Relation strength-aware clustering of heterogeneous information networks with incomplete attributes. *arXiv preprint arXiv:1201.6563* (2012).

[40] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. 2011. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment* 4, 11 (2011), 992–1003.

[41] Yizhou Sun, Jiawei Han, Peixiang Zhao, Zhijun Yin, Hong Cheng, and Tianyi Wu. 2009. Rankclus: integrating clustering with ranking for heterogeneous information network analysis. In *Proceedings of the 12th international conference on extending database technology: advances in database technology*. 565–576.

[42] Yizhou Sun, Brandon Norick, Jiawei Han, Xifeng Yan, Philip S Yu, and Xiao Yu. 2013. Pathselclus: Integrating meta-path selection with user-guided object clustering in heterogeneous information networks. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 7, 3 (2013), 1–23.

[43] Yizhou Sun, Yintao Yu, and Jiawei Han. 2009. Ranking-based clustering of heterogeneous information networks with star network schema. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 797–806.

[44] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. ArnetMiner: Extraction and Mining of Academic Social Networks. In *KDD'08*. 990–998.

[45] Ruby W Wang and Y Ye Fred. 2019. Simplifying Weighted Heterogeneous networks by extracting h-Structure via s-Degree. *Scientific reports* 9, 1 (2019), 1–8.

[46] Carl Yang, Yuxin Xiao, Yu Zhang, Yizhou Sun, and Jiawei Han. 2020. Heterogeneous Network Representation Learning: A Unified Framework with Survey and Benchmark. *TKDE* (2020).

[47] Dingqi Yang, Daqing Zhang, Longbiao Chen, and Bingqing Qu. 2015. NationTelescope: Monitoring and visualizing large-scale collective behavior in LBSNs. *Journal of Network and Computer Applications* 55 (2015), 170–180.

[48] Dingqi Yang, Daqing Zhang, and Bingqing Qu. 2015. Participatory cultural mapping based on collective behavior in location based social networks. *ACM Transactions on Intelligent Systems and Technology* (2015). in press.

[49] Yixing Yang, Yixiang Fang, Xuemin Lin, and Wenjie Zhang. 2020. Effective and efficient truss computation over large heterogeneous information networks. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 901–912.

[50] Xiao Yu, Xiang Ren, Yizhou Sun, Bradley Sturt, Urvashi Khandelwal, Quanquan Gu, Brandon Norick, and Jiawei Han. 2013. Recommendation in heterogeneous information networks with implicit user feedback. In *Proceedings of the 7th ACM conference on Recommender systems*. 347–350.

[51] Long Yuan, Lu Qin, Wenjie Zhang, Lijun Chang, and Jianye Yang. 2017. Index-based densest clique percolation community search in networks. *IEEE Transactions on Knowledge and Data Engineering* 30, 5 (2017), 922–935.

[52] Zhirong Yuan, You Peng, Peng Cheng, Li Han, Xuemin Lin, Lei Chen, and Wenjie Zhang. 2022. Efficient k-clique Listing with Set Intersection Speedup. In *ICDE*. IEEE.

[53] Yikai Zhang and Jeffrey Xu Yu. 2019. Unboundedness and efficiency of truss maintenance in evolving graphs. In *Proceedings of the 2019 International Conference on Management of Data*. 1024–1041.

[54] Yaping Zheng, Shiyi Chen, Xinni Zhang, Xiaofeng Zhang, Xiaofei Yang, and Di Wang. 2020. Heterogeneous-Temporal Graph Convolutional Networks: Make the Community Detection Much Better. arXiv:1909.10248 [cs.LG]

[55] Alexander Zhou, Yue Wang, and Lei Chen. 2020. Finding large diverse communities on networks: the edge maximum k*-partite clique. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2576–2589.

[56] Yang Zhou and Ling Liu. 2013. Social influence based clustering of heterogeneous information networks. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 338–346.

[57] Honglei Zhuang, Jing Zhang, George Brova, Jie Tang, Hasan Cam, Xifeng Yan, and Jiawei Han. 2014. Mining query-based subnetwork outliers in heterogeneous information networks. In *2014 IEEE International Conference on Data Mining*. IEEE, 1127–1132.

A PROOFS

Proof. **The hardness of CSSH problem.** There exists a solution (which we discuss in Section 3) running in $f(L) \cdot n^{O(1)}$ time, where L denotes the maximal length of meta-paths, n denotes the number of edges in HINs, and f is a function of L which is independent of n. Hence, CSSH problem is fixed-parameter tractable (FPT) with respect to parameter L. □

B META-PATHS GENERATION

In this subsection, we introduce how to generate meta-paths with the length limit L from the HIN schema $T_{\mathcal{H}}$. The meta-paths are generated via a breadth-first search on the HIN schema. As we only consider symmetric meta-paths, the breadth-first search focuses on generating the corresponding half meta-paths.

Algorithm 6 presents the details. `GenMetaPaths` (Algorithm 6) takes the HIN schema $T_{\mathcal{H}}$, the length limit L, and the target vertex type A as input. `GenMetaPaths` first initializes two empty sets $\mathcal{X}$ for recording all meta-paths and S for storing meta-paths to be extended (line 2). We generate the half meta-paths according to the length from 1 to $\frac{L}{2}$ iteratively (lines 3-10). In each iteration, we use S' to store the meta-paths with length equal to i temporarily (line 4). Meanwhile, S stores the meta-paths with length equal to $i-1$. To generate the length-i meta-paths, the algorithm iterates over each meta-path in S (line 5), obtains the last vertex A_i of the meta-path (line 6), concatenates the meta-path with each neighbor of A_i (line 8), and inserts the new meta-path into S' (line 9). Then, all length-i meta-paths in S' are inserted into $\mathcal{X}$, and S is updated to S' (line 10). After finding all half meta-paths, we get symmetric meta-paths from half meta-paths in $\mathcal{X}$ (line 11). Finally, the meta-path set $\mathcal{X}$ is returned as the output.

Algorithm 6: Generate all meta-paths $\mathcal{X}$

```
1  Function GenMetaPaths(T_H, L, A):
2      X ← ∅, S ← {A};
3      for i ← 1 to L/2 do
4          S' ← ∅;
5          foreach P_half ∈ S do
6              A_i ← the last node of P_half;
7              foreach neighbor A_{i+1} of A_i do
8                  P'_half ← P_half ‖ A_{i+1};
9                  insert P'_half into S';
10         insert all meta-paths in S' to X, S ← S';
11     X ← get symmetric meta-paths from X;
12     return X;
```

C CANDIDATE GENERATION

In this subsection, we introduce how to generate candidate meta-path sets based on Lemma 2.

Algorithm 7 presents the details. `GenCan` (Algorithm 7) takes a set $\mathcal{S}_c$ of qualified size-c meta-path set as input. Algorithm 7 generates new size-$(c+1)$ candidate meta-path sets incrementally by linking each pair of size-c meta-path sets. We first initialize $\mathcal{S}_{c+1}$ as an empty set (line 2). Then for each pair, Ψ_i and Ψ_j, of meta-path sets in $\mathcal{S}_c$, we sort their meta-paths. If they differ only at the last meta-path, then we generate a new set of meta-paths $\Psi' = \Psi_i \cup \Psi_j$, by a union operation (lines 3-6). According to Lemma 2, if any subset of Ψ' does not appear in $\mathcal{S}_c$, we prune Ψ'. Meanwhile, if there exists a pair of nested meta-paths in Ψ', we prune Ψ' as well. Otherwise, we regard it as a candidate and add it into $\mathcal{S}_{c+1}$ (lines 7-8). Finally, we return $\mathcal{S}_{c+1}$ as the results (line 9).

Algorithm 7: Generate candidate meta-path sets

```
1  Function GenCan(S_c):
2      S_{c+1} ← ∅;
3      foreach Ψ_i ∈ S_c do
4          foreach Ψ_j ∈ S_c do
5              if Ψ_i and Ψ_j differs at the last meta-path then
6                  initialize Ψ' = Ψ_i ∪ Ψ_j;
7                  if there does not exist a pair of nested meta-paths in
                     Ψ' and Ψ' cannot be pruned by Lemma 2 then
8                      S_{c+1}.add(Ψ')
9      return S_{c+1};
```

D VERTEX DELETION

In this subsection, we introduce how to delete vertices in S and update $\mathcal{R}$.

Algorithm 8 presents the details. `DeleteVertex` (Algorithm 8) takes a k neighbors map $\mathcal{R}$, a set of vertices S that need to be deleted and a set of vertices V that the HIN shares as input. Algorithm 8 can get a set of vertices V that for every $\mathcal{P}$ in Ψ, all vertices in the homogeneous graph induced by $\mathcal{P}$ from HIN $\mathcal{H}$ and restricted by V have more than k neighbors. Generally, we first remove the vertices in S from V, because they do not satisfy the constraint of k (lines 3-4). Since for every $\mathcal{P}$ in Ψ, 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 (lines 6-14). If we can not find enough $\mathcal{P}$-neighbors for v's $\mathcal{P}$-neighbors, we put them into S as vertices that need to be deleted (line 15). The whole process will stop when there are no vertices in S (line 2) or the vertex in Q is added into S (line 5).

Lemma 14. *Algorithm 8 takes* $O(\sum_{i=1}^{|\Psi|}(n_1 \cdot b_{1,2} + n_1 \sum_{i=2}^{L} n_i \cdot b_{i,i+1}))$ *time.*

Proof. In the worst case, we need to delete all vertices with the target type. Since before we delete a vertex, we must enumerate all the path instances of $\mathcal{P}$ starting from it, so this will take $O(\sum_{i=1}^{|\Psi|}(n_1 \cdot b_{1,2} + n_1 \sum_{i=2}^{L} n_i \cdot b_{i,i+1}))$ time. Hence, Lemma 14 holds. □

E INDEX-BASED VALID CANDIDATES GENERATION

In this subsection, we introduce how to generate a set of valid candidate meta-path sets $\mathcal{Y}$ based on Lemma 10.

Algorithm 9 presents the details. `GenValidCan` (Algorithm 9) takes valid meta-path set Ψ and CM-tree $\mathcal{T}$ as input. Algorithm 9 returns a set of valid meta-path sets $\mathcal{Y}$ that for every Ψ' in $\mathcal{Y}$, there exists a Ψ'-NMC. Generally, for each $\mathcal{P}$ in Ψ, we use a hash map $\mathcal{R}$

Algorithm 8: FastNMC (fast algorithm to get Ψ-NMC)

```
1  Function DeleteVertex(R, S, V):
2      while S ≠ ∅ do
3          v ← get a vertex from S and remove it from S;
4          remove v from V;
5          if v ∈ Q then return do not exist Ψ-NMC;
6          foreach P ∈ Ψ do
7              Φ ← R.get(P);
8              U ← {u|u is P-connected to v by a path in Φ[v]};
9              foreach vertex u ∈ U do
10                 if v is P-connected to u by a path instance
                       p ∈ Φ[u] then
11                     remove p from Φ[u];
12                     if |Φ[u]| < k then
13                         p' ← find a new path instance starting
                               from u;
14                         if p' exists then Φ[u].add(p') ;
15                         else S.add(u) ;
```

to store all $\mathcal{P}^{'}$ that $\mathcal{P}$ is the nested meta-path of $\mathcal{P}^{'}$ by searching CM-tree (lines 2-8). Then we call function GenCombination to generate the set of valid meta-path sets $\mathcal{Y}$ (line 9).

In function GenCombination, first we choose a $\mathcal{P}$ in Ψ and select a $\mathcal{P}^{'}$ that $\mathcal{P} \sqsubseteq \mathcal{P}^{'}$ (lines 12-16). If we have not selected $|\Psi|$ meta-paths, then we continue to choose next $\mathcal{P}$ in Ψ (line 19). Else we use these meta-paths to combine a $|\Psi|$-size meta-path set $\Psi^{'}$ and add it to $\mathcal{Y}$ (lines 17-18).

Algorithm 9: Generate valid candidates based on CM-tree

```
1  Function GenValidCan(Ψ, T):
2      Y ← ∅, visit[] ← false;
3      initialize an empty hash map R;
4      foreach P ∈ Ψ do
5          R.add(P, ∅), initialize a queue S ← {P};
6          while S ≠ ∅ do
7              P' ← S.pop(), R.get(P).add(P');
8              foreach P_i ∈ T_{P'}.childList do S.add(P_i) ;
9      GenCombination(Ψ, ∅, Y, visit, R);
10     return Y;
11 Function GenCombination(Ψ, Ψ', Y, visit, R):
12     foreach P ∈ Ψ do
13         if visit[P] = false then
14             visit[P] ← true;
15             foreach Ψ' ∈ M.get(P) do
16                 Ψ'.add(P');
17                 if |Ψ| = |Ψ'| then
18                     Y.add(Ψ'), remove P' from Ψ';
19                 else GenCombination(Ψ, Ψ', Y, visit, R) ;
20             visit[P] ← false;
```

F INDEX-BASED (k, Ψ)-NMC COMPUTATION

In this subsection, we introduce how to compute Ψ-NMC based on CM-tree.

Algorithm 10 presents the detail. IndexNMC (Algorithm 10) takes MC-tree $\mathcal{T}$, k, meta-path set Ψ and query vertex set Q as input. Generally, the algorithm includes two steps. First, for every $\mathcal{P}$ in Ψ, we call function IndexSingleCore to get $\mathbf{B}_{\mathcal{P}}$ (lines 2-3). Second, we compute the intersection set of all $\mathbf{B}_{\mathcal{P}}$s and use Algorithm 3 to compute Ψ-NMC (lines 4-5).

In function IndexSingleCore, we firstly initialize an empty set V to store vertices with core number greater than or equal to k (line 8). Then for $\mathcal{P}$ and each its nested meta-path $\mathcal{P}^{'}$, we add all vertices in $\mathcal{T}_{\mathcal{P}}.map.[k^{'}]$ to V, where $k^{'}$ is greater or equal to k (lines 9-11). After that, if query vertex set Q is not a subset of V, it means that there does not exist a $\mathbf{B}_{\mathcal{P}}$ and the function returns null (line 12). In the end, we compute the connected component on $\mathcal{H}[V]$ and get the $\mathbf{B}_{\mathcal{P}}$ (lines 13-14), where $\mathcal{H}[V]$ denotes the HIN $\mathcal{H}$ restricted with vertex set V and corresponding edges.

Algorithm 10: Compute Ψ-NMC based on CM-tree

```
1  Function IndexNMC(T, k, Ψ, Q):
2      foreach P ∈ Ψ do
3          B_P ← IndexSingleCore(k, P, Q, T);
4      V ← ∩_{P∈Ψ} B_P;
5      Ψ-NMC ← FastNMC(H, Q, k, Ψ, M);
6      return Ψ-NMC;
7  Function IndexSingleCore(k, P, Q, T):
8      initialize an empty hash set V;
9      foreach k' ∈ T_P.map.keyset() do
10         if k' ⩾ k then
11             add vertices in ⋃_{P'⊑P} T_{P'}.map[k'] to V;
12     if Q ⊄ V then return do not exist B_P ;
13     compute the connected component on H[V];
14     return B_P;
```

Lemma 15. *Function* IndexSingleCore *of Algorithm 10 takes* $O(\frac{1}{2} \cdot L \cdot n_1 + \sum_{i=1}^{L} n_i \cdot b_{i,i+1})$ *time.*

Proof. In the worst case, we need to traverse all vertices in a node of the CM-tree. And in the worst case, a node of the CM-tree needs to store n_1 vertices. Since for a meta-path $\mathcal{P}$ with length of l, it has at most $(\frac{1}{2} \cdot L - 1)$ nested meta-paths. So it totally costs $O(\frac{1}{2} \cdot L \cdot n_1)$ time to traverse vertices in the CM-tree. Apart from this, it costs $O(\sum_{i=1}^{L} n_i \cdot b_{i,i+1})$ time to compute a connected component on the homogeneous graph $\mathcal{H}[V]$. Hence, Lemma 15 holds. □

Lemma 16. *Algorithm 10 takes* $O(|\Psi| \cdot (\frac{1}{2} \cdot L \cdot n_1 + \sum_{i=1}^{L} n_i \cdot b_{i,i+1}) + \Delta)$ *time, where* Δ *denotes the time cost of* FastNMC *(Algorithm 3).*

Proof. In Algorithm 10, it calls $|\Psi|$ times IndexSingleCore and IndexSingleCore costs $O(\frac{1}{2} \cdot L \cdot n_1 + \sum_{i=1}^{L} n_i \cdot b_{i,i+1})$ time for each $\mathcal{P}$ in Ψ. Additionally, it calls FastNMC to compute Ψ-NMC. Hence, Lemma 16 holds. □