Searching and Detecting Structurally Similar Communities in Large Heterogeneous Information Networks

Shu Wang
The Chinese University of Hong Kong, Shenzhen
shuwang3@link.cuhk.edu.cn

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

Wensheng Luo
The Chinese University of Hong Kong, Shenzhen
luowensheng@cuhk.edu.cn

ABSTRACT

Heterogeneous information networks (HINs) are prevalent in various domains, including bibliographic information networks, social media, and knowledge graphs. As a fundamental topic in HIN mining, community mining has found various real applications, such as recommendation, biological data analysis, and event organization. Most existing works often rely on meta-paths, relational constraints, spectral partitioning, label propagation, and network representation to define the communities. However, almost all these works do not explicitly consider the structural similarity between vertices, which plays a vital role in modeling communities and also ignore the specific roles of vertices. In this paper, we propose a novel community model, called *structurally similar community (SSC)*, which models the HIN communities by explicitly considering the structural similarity between vertices. In particular, SSC can not only support various structural similarity measures, but also identify different roles of the vertices in the community, such as cores, non-cores, hubs, and outliers. Based on the SSC, we develop fast online and index-based algorithms that support both efficient searching and detecting SSCs in large HINs, where the former one searches an SSC containing a specific query vertex while the latter one detects all the SSCs from the HIN. Extensive experiments on real-world datasets demonstrate the effectiveness of SSC model in revealing meaningful communities and the high efficiency of our proposed algorithms.

PVLDB Reference Format:
Shu Wang, Yixiang Fang, and Wensheng Luo. Searching and Detecting Structurally Similar Communities in Large 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/sam234990/cs-hin-scan.

1 INTRODUCTION

Heterogeneous information networks (HINs) are networks with multiple typed objects and links denoting different relations [34, 64], which are prevalent in bibliographic networks [63, 70], IMDB movie networks [66, 88], e-commence networks [26], and knowledge graphs [27]. Figure 1 depicts an example HIN of the DBLP network with four types of vertices, i.e., author (A), paper (P), topic (T), and venue (V), where the directed lines denote the semantic relationships between these objects. For example, the author a_1 has *written* a paper p_1, which *mentions* the topic t_1.

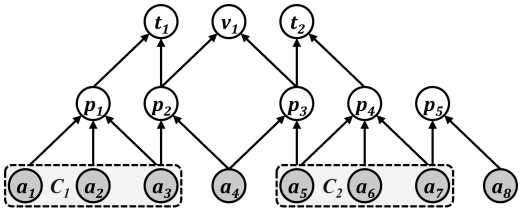


Figure 1: An example HIN of the DBLP network.

As a fundamental topic in graph mining, community mining has received tremendous attention in recent years [25, 28, 29, 57]. Conceptually, a community is a group of vertices densely connected internally and loosely associated with vertices outside the group. Mining communities from HINs have found various real applications, including event organization, friend recommendation, and biological data analysis [22, 58]. Generally, the existing works of community mining over HINs can be classified into *community search (CS)* [27, 41, 54, 78, 97] and *community detection (CD)* [69, 71–73, 73, 95], where the former one aims to search a community containing a specific query vertex, while the latter one often detects all the communities from the HIN.

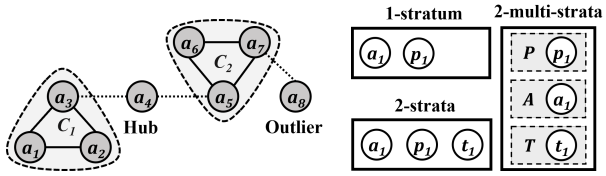
To search for and detect communities in HINs, Fang et al. [27, 41] proposed using meta-paths to model the link relationships between vertices, along with a minimum degree constraint to ensure cohesiveness. Similarly, Jian et al. [40] employed relational constraints to search for communities consisting of vertices from multiple types. Besides, Sun et al. [72] proposed to cluster objects of a target type using weighted meta-paths. Nevertheless, none of these prior works has explicitly considered the structural similarity between vertices, which is crucial for mining communities [85]. In the literature, many similarity measures have been developed, e.g., PathSim [70], HeteSim [62], and StructSim [39], which are effective for measuring the similarity between vertices in the HINs from different angles. Besides, these works ignore the specific roles of vertices such as cores, non-cores, hubs, and outliers. In Figure 1, for example, both vertex pairs (a_1, a_2) and (a_3, a_4) can be linked via instances of the meta-path "APA", but the former one exhibits a higher structural similarity since its two authors co-authored a paper p_1 and only collaborated with each other, showcasing a higher closeness compared to the latter pair. Also, the vertex a_4 plays the role of hub between two different communities $C_1 = \{a_1, a_2, a_3\}$ and $C_2 = \{a_5, a_6, a_7\}$.

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

As shown in previous works of mining communities from homogeneous graphs, the structural similarity is effective for modeling network communities. For example, as a classic and popular CD method, SCAN [85] explicitly considers the structural similarity by using neighbor sets, and requires that each core vertex of the community has at least μ ϵ-neighborhoods, where an ϵ-neighborhood is a neighborhood whose structural similarity with it is at least ϵ, and $\mu > 0$ and $\epsilon > 0$ are user-specific parameters. Particularly, it is capable of identifying the four different roles of vertices in the network [1], i.e., *core*, *non-core*, *hub*, and *outlier* vertices, where the first two roles are used to distinguish the importance of vertices in a community, hubs are vertices connecting different communities, and outliers do not belong to any community. These four roles are formulated based on the concepts of *Jaccard structural similarity* and *ϵ-neighborhood* in homogeneous graph. Due to the existence of multiple vertex and edge types in the HIN, these two concepts cannot be directly applied for the HIN, and none of existing works has formally defined the concept of ϵ-neighborhood for HINs.

SSC model. Motivated by the above, in this paper, we propose a novel community model, called **S**tructural **S**imilar **C**ommunity (SSC), for the community with vertices of the same type. In the HIN, the vertices and edges have different vertex types and edge types respectively. The nature of the heterogeneity above makes the similarity between two vertices of the same type have different meanings when considering the different paths or subgraphs between them. For example, given two authors in a DBLP network, their PathSim [70] similarity values could be different by using two different meta-paths $\mathcal{P}_1$ = "APA"and $\mathcal{P}_2$ ="$APVPA$". The heterogeneity in similarity motivates us to model the structural similarity between two vertices as a multi-dimensional vector. We then formulate the SSC model, in which each core vertex of the SSC has at least μ neighbors whose multi-dimensional structural similarity values surpass a threshold vector $\boldsymbol{\epsilon}$. The two main features of our SSC model are as follows.

• **A generic framework.** SSC provides a generic framework to support various similarity measures. In the literature, various HIN vertex similarity measures have been developed [15, 39, 49, 62, 68, 70, 82, 92], and different measures have different emphases. For example, PathSim [70] employs a symmetric meta-path to measure the semantic similarity between two vertices. Another classic similarity measure is the Jaccard similarity, which measures the semantic similarity between two vertices by using their shared local k-hop neighbors. For instance, in Figure 1, consider two authors a_2 and a_3 with k=2. They have 1 and 2 papers respectively, but they only share 1 paper, so their Jaccard similarity on papers is 0.5. Similarly, their Jaccard similarity on topics is 1.0. As a result, their similarity in papers and topics can be represented by a vector $\boldsymbol{\sigma}(a_2, a_3) = [0.5, 1]^\top$. Besides, such a multi-dimensional vector can more accurately describe the similarity between vertices than a single aggregated similarity value (e.g., average value), since directly aggregating similarities for different types of vertices can lose semantic information. For instance, consider the pairs (p_1, p_2) and (p_2, p_3) in Figure 1. Their similarity vectors on topics and venues are $\boldsymbol{\sigma}(p_1, p_2) = [1.0, 0.0]^\top$ and $\boldsymbol{\sigma}(p_2, p_3) = [0.0, 1.0]^\top$, respectively. However, if we aggregate the similarity values from different vertex types by using an average value, their similarity values are the same (i.e., 0.5), leading to no difference between these two pairs.



(a) Two SSCs, a hub, and an outlier (b) k-strata and 2-multi-strata of a_1

Figure 2: Illustrating SSC and k-multi-strata.

• **Identifying different roles of vertices.** Existing HIN community models, such as $(k, \mathcal{P})$-core [86], HIC [97], and attribute-based model [78], can only differentiate two roles of vertices in the network, i.e., members or non-members of communities. In contrast, our SSC model is able to identify four different roles: core, non-core, hub, and outlier. Core and non-core vertices are part of communities, but the former ones play a more important role; hub vertices bridge multiple communities; outlier vertices do not belong to any community. In Figure 2(a), for example, when we use the threshold $\boldsymbol{\epsilon} = [0.3, 0.7]^\top$, the author a_4 functions as a hub, since s/he does not belong to any community, but is linked to the two communities that contain a_3 and a_5 respectively. These roles are particularly useful in many real applications [31, 48]. For instance, the outlier vertices can be used for identifying anomaly [13, 33, 51, 94], while the hub vertices are believed to play a crucial role in viral marketing and epidemiology [18, 43, 81].

Despite the above advantages of SSC, computing SSCs from large HINs is a very challenging task. First, computing the multi-dimensional similarity between vertices is expensive because it requires searching paths or multi-hop neighbors according to different similarity measures, and there may exist a large number of paths or vertices between two vertices of the same type. Second, in practice, the users may frequently explore the SSCs by varying the values of the parameters, which leads to high costs. Thus, considering the huge sizes of real-world HINs, it is desirable to develop efficient algorithms to retrieve SSCs.

Our technical contributions. To enable efficient retrieval of SSCs, we propose an online CS algorithm that finds the SSC containing a query vertex in a local search manner, along with some effective pruning strategies. Our SSC model well caters to both CS and CD, because it formulates communities by purely using vertices' similarity values and their related threshold. Thus, the online CS algorithm can be easily extended as a CD algorithm by repeating the CS process for the remaining vertices.

In real-world scenarios, users often need to frequently vary the query parameters to find desirable communities in an HIN. Therefore, we develop an index, called the SC-Index, to support frequent queries. We first pre-compute the similarity values between all the vertices, and then identify the nested property of core vertices under different parameters, based on which we can compactly organize the core vertices under all the possible values of parameters. We further develop an efficient index-based CS algorithm, which searches the community by expanding from the query vertex and core vertices. This algorithm can also be easily extended to address the CD problem using the similar idea above. In addition, we propose two algorithms to build the SC-index, and the advanced one efficiently identifies the core vertices, by avoiding repeated computation under some specific parameter values.

We have performed extensive experiments on five real large HIN datasets, and the results show that our SSC model is effective for modeling real-world communities. Besides, the index-based CS algorithm is up to three orders of magnitude faster than the online CS algorithm. In addition, the advanced index construction algorithm is two orders of magnitude faster than the basic algorithm.

Outline. We review the related works in Section 2 and formulate the SSC model, and SSC search and detection problems in Section 3. In Sections 4 and 5, we introduce the online and index-based SSC search algorithms, respectively. We present the experimental results in Section 6, and finally conclude the paper in Section 7.

2 RELATED WORK

In this section, we review of the related works of Community Search (CS) and Community Detection (CD) on large graphs, respectively.

• **Community Search (CS).** Generally, CS aims to search for high-quality communities containing the query vertices in an online manner [17, 25, 27]. On homogeneous graphs, various cohesive subgraph models are used for modeling the communities, such as k-core [2, 4, 17], k-truss [12, 23, 26, 42, 52, 93], k-clique [11, 16, 89] and k-ecc (k-edge connected component) [8, 35]. In particular, the k-core model is the most commonly used and has been extended with methods that consider additional information [4, 24].

Recently, the topic of CS over HINs has attracted much attention [20, 27, 30, 40, 59, 77–79, 96, 97]. There are two groups of existing approaches for CS in HINs, according the two types of HINs: (1) HINs in which each vertex has only one vertex type and edges between different vertex types are uniform, and (2) HINs where vertices or edges possess additional attributes, such as labeled graphs [20, 96], attributed graphs [77, 78], and multi-layer graphs [30]. For the first group, Fang et al. [27] introduced $(k, \mathcal{P})$-core, 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. Jiang et al. [41] studied CS over large star-schema HINs without asking for specifying meta-paths. Besides, Jian et al. [40] searched communities with vertices of multiple types by using relational constraints and minimum degree constraint to ensure cohesiveness. For the second group, the additional information of HINs, such as keywords [59] and influence values [97] has also been considered for CS. However, all these works do not explicitly consider vertex similarity when modeling communities and also ignore the roles of vertices in the HIN. In contrast, our SSC model well considers the vertex similarity and different roles of vertices in the HIN.

• **Community Detection (CD).** On homogeneous graphs, various CD methods have been developed [3, 28, 29], such as spectral clustering [19, 46, 84, 87], consensus clustering [32, 75], modularity [14, 21, 56, 57], statistical inference [44, 60], and structural similarity [6, 83, 85]. Among these methods, SCAN [85] differs from others methods by explicitly considering the structural similarity between vertices. It requires that each core vertex of the community has at least μ ϵ-neighborhoods whose structural similarity values with it are at least ϵ. Many works have further studied this topic [9, 50, 55, 61, 67, 74, 76, 90]. For instance, GS*-Index [83] improves the efficiency by building an index. Although these methods consider the similarity between vertices, they cannot be directly applied to HINs due to the presence of multiple types of vertices and edges.

CD methods over HINs [10, 53, 64, 65, 69–73, 86, 98, 99] can be roughly divided into two groups according to vertex types in communities. The first group aims to detect communities containing objects with multiple types [10, 65, 69, 73], while the second group [71, 72, 99] generates clusters of objects with a specific type. For example, in the former group, Chen et al. [10] adopted incremental approaches to cluster objects on star-schema HINs. In the second group, Sun et al. proposed an algorithm to generate clusters of a specific type of objects [71], and a user-guided algorithm [72] to cluster objects of a target type; in [99], a social influence-based clustering algorithm is presented. Nevertheless, these approaches do not explicitly consider similarity when modeling the community in the HIN, and also do not distinguish the four roles of vertices of the community.

3 PROBLEM FORMULATION

We first formally formulate the SSC model and SSC search and detection problems, and then present two HIN similarity measures.

3.1 Problem formulation

Definition 1 (HIN [70]). *An HIN is a directed graph $\mathcal{H} = (V, E)$ with a vertex type mapping function $\psi : V \to \mathcal{A}$ and an edge type mapping function $\phi : E \to \mathcal{R}$, where each vertex $v \in V$ belongs to a vertex type $\psi(v) \in \mathcal{A}$, and each edge $e \in E$ belongs to an edge type (also called relation) $\phi(e) \in \mathcal{R}$, and $|\mathcal{A}| + |\mathcal{R}| > 2$.*

An HIN often follows a schema, which is a directed graph defined over vertex types $\mathcal{A}$ and edge types $\mathcal{R}$, denoted by $T_{\mathcal{H}} = (\mathcal{A}, \mathcal{R})$.

Before introducing the SSC model, we present the concept of multi-dimensional structural similarity, which is pivotal in our SSC model, as it enables the integration of various similarity measures between vertices into a generic framework.

Definition 2 (Multi-dimensional structural similarity). *Given an HIN $\mathcal{H}$ and two vertices u and v with the same type, their **multi-dimensional structural similarity** is a vector*

$$\boldsymbol{\sigma}(u, v) = [\sigma_1, \cdots, \sigma_h]^\top, \tag{1}$$

reflecting their similarity from different aspects, where σ_i ($i \in [1, h]$, $h \geq 1$, $\sigma_i \in [0, 1]$) is the value of the i-th dimension of similarity vector.

Here, the value in each dimension of $\boldsymbol{\sigma}(u, v)$ can be computed by any existing HIN vertex similarity measure (e.g., PathSim [70], HeteSim [62], HowSim [82]), and we will present two measures used in this paper in Section 3.2.

For ease of comparison, given two vectors $\boldsymbol{a}$[1] and $\boldsymbol{b}$ with same dimensions, we use $\boldsymbol{a} \geq \boldsymbol{b}$ to indicate each dimension of $\boldsymbol{a}$ is not less than the corresponding dimension of $\boldsymbol{b}$; $\boldsymbol{a} > \boldsymbol{b}$ indicates $\boldsymbol{a} \geq \boldsymbol{b}$ with at least one dimension being strictly greater. Given a threshold vector $\boldsymbol{\epsilon}$ with $|\boldsymbol{\epsilon}| \in (0, 1]$, we introduce the concept of $\boldsymbol{\epsilon}$-neighbor to record the similar neighbors that meet the threshold constraint, which is defined as:

Definition 3 ($\boldsymbol{\epsilon}$-neighbor). *Given an HIN $\mathcal{H}$, a threshold vector $\boldsymbol{\epsilon}$, and a vertex $u \in \mathcal{H}$, the **$\boldsymbol{\epsilon}$-neighbor** of u, denoted by $N_{\boldsymbol{\epsilon}}[u]$, is*

[1] In this paper, to represent a vector, we use a boldface letter or number (e.g., $\boldsymbol{a}$ and $\boldsymbol{8}$).

defined as the subset of vertices with the same type as u, in which each vertex v satisfies $\boldsymbol{\sigma}(u,v) \geq \boldsymbol{\epsilon}$, i.e.,

$$N_{\boldsymbol{\epsilon}}[u] = \{v | \psi(v) = \psi(u), \boldsymbol{\sigma}(u,v) \geq \boldsymbol{\epsilon}\}. \quad (2)$$

Note that the $\boldsymbol{\epsilon}$-neighbor of u includes itself. We also call a vertex v an *s-neighbor* of u ($N_s[u]$), if it has the same type with u and $\boldsymbol{\sigma}(u,v) > \mathbf{0}$. Obviously, an $\boldsymbol{\epsilon}$-neighbor must be an s-neighbor.

When the number of $\boldsymbol{\epsilon}$-neighbors of a vertex is large enough, it becomes a *core vertex*, defined as follows:

Definition 4 (Core vertex). *Given a threshold vector $\boldsymbol{\epsilon}$ and an integer $\mu \geq 2$, a vertex u is a **core vertex** if $|N_{\boldsymbol{\epsilon}}[u]| \geq \mu$.*

Reversely, if a vertex of an SSC is not a core vertex, then it is called a *non-core vertex*. We next introduce the concept of structural reachability and formulate our SSC model.

Definition 5 (Structural reachability). *Given an HIN $\mathcal{H}$, a threshold vector $\boldsymbol{\epsilon}$, and two vertices u and v with the target type, v is **structurally reachable** from u, if there is a sequence of vertices $v_1, v_2, \ldots, v_l$ ($l \geq 2$) such that: 1) $v_1 = u, v_l = v$; and 2) for all $1 \leq i \leq l-1$, v_i is a core vertex, and $v_{i+1} \in N_{\boldsymbol{\epsilon}}[v_i]$.*

Definition 6 (Structural Similar Community (SSC)). *Given an HIN $\mathcal{H}$, an integer μ, and a threshold vector $\boldsymbol{\epsilon}$, an **SSC** is a set C ($|C| \geq 2$) of vertices with the same type, satisfying:*

- **Connectivity:** *For any two vertices $v_1, v_2 \in C$, there is a vertex $u \in C$ such that both v_1 and v_2 are structurally reachable from u.*
- **Maximality:** *If a core vertex $u \in C$, then all vertices that are structurally reachable from u also belong to C.*

For example, in Figure 2(a), given a specific $\boldsymbol{\epsilon}$ and $\mu = 3$, the solid lines are used to indicate similarity between vertices exceeding the threshold $\boldsymbol{\epsilon}$, whereas dashed lines represent similarity below. We can deduce that a_2's $\boldsymbol{\epsilon}$-neighbor $N_{\boldsymbol{\epsilon}}[a_2] = \{a_1, a_2, a_3\}$ contains enough vertices, making a_2 be a core vertex. Since a_3 and a_5 are not structurally reachable, two SSCs are formed: $C_1 = \{a_1, a_2, a_3\}$ and $C_2 = \{a_5, a_6, a_7\}$. After identifying all the SSCs, we can obtain the sets of *hub* and *outlier vertices*, defined as follows.

Definition 7 (Hub and outlier vertices). *Given an HIN $\mathcal{H}$ and a vertex $u \in \mathcal{H}$ that does not belong to any SSC, if u satisfies $\boldsymbol{\sigma}(u,v) \geq \mathbf{0}$ for vertices v in two or more different communities, then u is a hub vertex; otherwise, it is an outlier vertex.*

In the case of Figure 2(a), a_4 is a hub vertex, since it connects two different SSCs via vertices a_3 and a_5. Conversely, a_8 is an outlier vertex. Based on the SSC model, we now formally formulate the problems of searching and detecting SSCs as follows.

Problem 1 (SSC search). *Given an HIN $\mathcal{H}$, an h-dimensional similarity measure ($h \geq 1$), an integer μ ($\mu \geq 2$), an h-dimensional vector $\boldsymbol{\epsilon}$, and a query vertex q, return an SSC containing q.*

Problem 2 (SSC detection). *Given an HIN $\mathcal{H}$, an h-dimensional similarity measure ($h \geq 1$), an integer μ ($\mu \geq 2$), an h-dimensional vector $\boldsymbol{\epsilon}$, and a target vertex type, return all SSCs in $\mathcal{H}$ and identify the role of each vertex with the target type.*

Table 1: Notations and meanings.

Notation(s)	Meaning
$\mathcal{H}$=(V, E)	An HIN with vertex set V and edge set E
$\psi(v)$, $\phi(e)$	The vertex and edge type mapping functions
$\boldsymbol{\sigma}(u,v)$	The vector of multi-dimensional structural similarity between vertices u and v
$\boldsymbol{a}$	An h-dimensional vector
$\boldsymbol{a}_i$	The value of the i-th dimension of a vector $\boldsymbol{a}$
$\boldsymbol{\epsilon}, \mu$	Parameters of SSC model
$N_{\boldsymbol{\epsilon}}[u]$	A set of ϵ-neighbors of vertex u
$N_s[u]$	The s-neighbors of vertex u
$\Gamma(u,k)$	The k-multi-strata of vertex u
$N_G[u]$	The neighbors of vertex u in the similarity graph
$\Phi[u]$	The core threshold set (CTS) of vertex u

3.2 Two similarity measures for HIN vertices

We present two vertex similarity measures used in our paper. Note, however, that other similarity measures (e.g., HeteSim [62] and StructSim [39]) can also be used in our generic framework.

• **PathSim [70]**. This measure is defined based on the concept of meta-path. A meta-path $\mathcal{P}$ is a path defined on an HIN schema $T_{\mathcal{H}} = (\mathcal{A}, \mathcal{R})$, denoted in the form $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \cdots \xrightarrow{R_l} A_{l+1}$, where l is the length of $\mathcal{P}$, $A_i \in \mathcal{A}$, and $R_i \in \mathcal{R}$ ($1 \leq i \leq l$). For example, $Author \xrightarrow{write} Paper \xrightarrow{write^{-1}} Author$, abbreviated as APA, denotes the co-authorship between two authors.

Definition 8 (PathSim [70]). *Given a symmetric meta-path $\mathcal{P}$ and two vertices x and y, the PathSim similarity between x and y is:*

$$Sim_{\mathcal{P}}(x,y) = \frac{2 \times |\{p_{x \rightsquigarrow y} : p_{x \rightsquigarrow y} \in \mathcal{P}\}|}{|\{p_{x \rightsquigarrow x} : p_{x \rightsquigarrow x} \in \mathcal{P}\}| + |\{p_{y \rightsquigarrow y} : p_{y \rightsquigarrow y} \in \mathcal{P}\}|} \quad (3)$$

where $p_{x \rightsquigarrow y}$ is a path instance between x and y, $p_{x \rightsquigarrow x}$ is that between x and x, and $p_{y \rightsquigarrow y}$ is that between y and y.

Therefore, given h meta-paths ($h \geq 1$), we can represent the multi-dimensional similarity between two vertices as a vector:

$$\boldsymbol{\sigma}(u,v) = [Sim_{\mathcal{P}_1}(u,v), Sim_{\mathcal{P}_2}(u,v), \cdots, Sim_{\mathcal{P}_h}(u,v)]^\top. \quad (4)$$

• **Jaccard similarity**. As shown in SCAN algorithm [85], given two vertices u and v, their Jaccard similarity is defined as

$$Sim(u,v) = \frac{|\Gamma(u) \cap \Gamma(v)|}{|\Gamma(u) \cup \Gamma(v)|}, \quad (5)$$

where $\Gamma(u)$ is the neighbor set of u, including u itself.

However, in the HIN, as vertices with the same type are often not directly connected, we have to consider the multi-hop neighbors to establish their relationship. Inspired by *k-strata* [91], we introduce the concept of k-multi-strata to include the k-hop neighbors.

Definition 9 (k-multi-strata). *Given an HIN, an integer k, and a vertex u, u's k-**multi-strata** is a multi-set of vertices,*

$$\Gamma(u,k) = \{\Gamma_1(u,k), \Gamma_2(u,k), \cdots, \Gamma_h(u,k)\} \quad (6)$$

where each $\Gamma_i(u,k)$ ($i \in [1,h]$) contains a set of all the vertices with the same type which are within the k-hop distances to u.

By considering the different vertex types in the k-multi-strata, we can define the multi-dimensional similarity between two vertices as a vector:

$$\boldsymbol{\sigma}(u,v) = [Sim_1(u,v), Sim_2(u,v), \cdots, Sim_h(u,v)]^\top, \quad (7)$$

Algorithm 1: Online SSC search algorithm.

Input: An HIN $\mathcal{H}(V, E)$, a core vertex q, parameters (e.g., ϵ and μ)
Output: An SSC C containing q

1 Obtain $N_s[q] = \{v | \psi(v) = \psi(q), \boldsymbol{\sigma}(q, v) \geq \boldsymbol{0})\}$;
2 $sd(q) \leftarrow 0, ed(q) \leftarrow |N_s[q]|$;
3 $C \leftarrow \{q\}$, $Q.enqueue(q)$;
4 **while** $Q \neq \emptyset$ **do**
5 $u \leftarrow Q.dequeue()$;
6 **if** *CheckCore(u) is true* **then**
7 **for** *each vertex* $v \in N_s[u]$ **do**
8 **if** $v \notin C$ ***and*** $\boldsymbol{\sigma}(u, v) \geq \boldsymbol{\epsilon}$ **then**
9 $C \leftarrow C \cup \{v\}$, $Q.enqueue(v)$;
10 **for** *each vertex* $v \in N_s[u]$ **do**
11 **if** v *is not explored* **then**
12 Obtain $N_s[v] = \{w | \psi(w) = \psi(v), \boldsymbol{\sigma}(v, w) \geq \boldsymbol{0})\}$;
13 $sd(v) \leftarrow 0, ed(v) \leftarrow |N_s[v]|$;
14 **if** $\boldsymbol{\sigma}(u, v) \geq \boldsymbol{\epsilon}$ **then** $sd(v) \leftarrow sd(v) + 1$;
15 **else** $ed(v) \leftarrow ed(v) - 1$;
16 **return** C;

where $Sim_i(u, v) = \frac{|\Gamma_i(u,k) \cap \Gamma_i(v,k)|}{|\Gamma_i(u,k) \cup \Gamma_i(v,k)|}$.

For instance, in Figure 1, a_1's 1-stratum is $\{a_1, p_1\}$, the 2-strata includes $\{a_1, p_1, t_1\}$, and the 2-multi-strata is $\Gamma(a_1, 2) = \{\Gamma_A(a_1, 2) = \{a_1\}, \Gamma_P(a_1, 2) = \{p_1\}, \Gamma_T(a_1, 2) = \{t_1\}\}$, as depicted in Figure 2(b). Similarly, we can obtain $\Gamma(a_2, 2)$ and calculate $\boldsymbol{\sigma}(a_1, a_2) = [0, 1, 1]^\top$, since $Sim_A(a_1, a_2) = 0$, $Sim_P(a_1, a_2) = 1$, and $Sim_T(a_1, a_1) = 1$.

4 ONLINE SSC SEARCH ALGORITHM

Our online algorithm performs a local search starting from the query vertex q in an iterative procedure. Specifically, we first check whether q is a member of an SSC by verifying the similarity constraints of $\boldsymbol{\epsilon}$ and μ. If it is not a member, then we return null; otherwise, it is either a core vertex or a non-core vertex of an SSC. In case q is a core vertex, we use a queue, which is initialized with q, to enlarge the SSC by expanding q's $\boldsymbol{\epsilon}$-neighbors iteratively, as shown in Algorithm 1. In case q is a non-core vertex, we can first identify its $\boldsymbol{\epsilon}$-neighbors who are core vertices and run Algorithm 1 for each of these core vertices.

Particularly, in line with previous works [6], to speed up the search process, we also use the concepts of *similar-degree (sd)* and *effective-degree (ed)* to denote the lower and upper bounds of the numbers of $\boldsymbol{\epsilon}$-neighbor of a vertex. More precisely, $sd(u)$ is the number of currently determined structurally similar neighbors of u in the search process, while $ed(u)$ is the number of currently maximum potentially structurally similar neighbors.

Algorithm 1 works for the case that q is a core vertex. We first obtain the s-neighbors of q and initialize $sd(q)$ and $ed(q)$ (lines 1-2). Then, we initialize an SSC C by q and a priority queue Q by q (line 3). The priority of a vertex u is determined by $ed(u)$, where a higher $ed(u)$ value implies a greater likelihood of u becoming a core vertex. Afterward, we use a while-loop to complete the SSC expansion process (lines 4-15). Specifically, we first dequeue a vertex u and check whether u is a core vertex (lines 5-6). If yes, we enumerate u's s-neighbors v and add v to the queue if $\boldsymbol{\sigma}(u, v) \geq \boldsymbol{\epsilon}$, for the expansion in the next iteration (lines 7-9). Besides, for each s-neighbors v of u, we obtain its s-neighbors $N_s[v]$, initialize $sd(v)$ and $ed(v)$, and update their values (lines 10-15). Finally, the SSC is returned when the expansion terminates (line 16).

Algorithm 2: CheckCore(v)

1 **if** $ed(v) \geq \mu$ ***or*** $sd(v) < \mu$ **then**
2 $sd(v) \leftarrow 0, ed(v) \leftarrow |N_s[v]|$;
3 **for** *each vertex* $w \in N_s[v]$ **do**
4 compute $\boldsymbol{\sigma}(v, w)$ under the similarity measure;
5 **if** $\boldsymbol{\sigma}(v, w) \geq \boldsymbol{\epsilon}$ **then** $sd(v) \leftarrow sd(v) + 1$;
6 **else** $ed(v) \leftarrow ed(v) - 1$;
7 **if** $sd(v) \geq \mu$ ***or*** $ed(v) < \mu$ **then break**;
8 Mark v as explored;
9 **if** $sd(v) \geq \mu$ **then return** *true*;
10 **return** *false*

Note that to check whether a vertex v is a core vertex, we use the function *CheckCore*, as described in Algorithm 2. The key idea is to verify whether there are μ vertices in s-neighbors whose similarity values with v are at least $\boldsymbol{\epsilon}$. During this process, we use the lower bound $sd(v)$ and upper bound $ed(v)$ to achieve early termination.

Example 1. *Consider the HIN in Figure 1 and let q=a_2, μ=3, and $\boldsymbol{\epsilon} = [\boldsymbol{\epsilon_P} = 0.5, \boldsymbol{\epsilon_T} = 0.7]^\top$ with Jaccard similarity. The above algorithm first obtains $N_s[a_2] = \{a_1, a_2, a_3\}$ and initializes $sd(a_2) = 0$ and $ed(a_2) = 3$. Subsequently, it calculates the similarity for each vertex in $N_s[a_2]$. Since a_1 and a_3 have not been explored yet and are $\boldsymbol{\epsilon}$-neighbors of a_2, $sd(a_1), sd(a_2)$ and $sd(a_3)$ are increased by one. Next, it expands from a_3 as $ed(a_3) = 4 > ed(a_1) = 3$. Similarly, it stops after executing $CheckCore(a_1)$ and returns the SSC $C = \{a_1, a_2, a_3\}$.*

Lemma 1. *Algorithm 1 completes in $O(\alpha n + \beta m)$ time, where α represents the cost of computing the s-neighbors for a vertex, n is the number of vertices within SSC and their s-neighbors, β represents the cost of computing similarity between two vertices, and m represents the total number of s-neighbors of all these vertices.*

For lack of space, the proofs of all the lemmas in this paper are included in the appendix of our technical report [80].

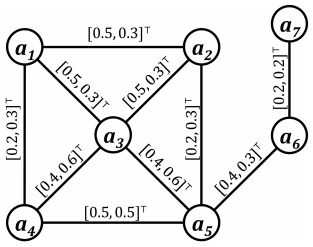
5 INDEX-BASED SSC SEARCH ALGORITHMS

In real-world scenarios, to obtain the desirable communities, users often need to frequently vary the query parameters [83]. While the online algorithm efficiently retrieves results for singular queries, it is inefficient for such scenarios. In existing SCAN algorithms, the GS*-Index [83] is effective for mining communities from homogeneous graphs. GS*-Index uses two ordered indices, Core-Order and Neighbor-Order, to achieve rapid core vertex retrieval and truncation of similarity comparison. However, since SSC is defined on HINs measured by an h-dimensional similarity measure, making it difficult to define a fixed order for the vertices, the GS*-Index cannot be directly extended for indexing SSCs.

In this section, we propose an index by pre-computing **S**imilarity and **C**ore vertices, called SC-Index. In the following, we first give an overview of SC-Index, then present the index-based SSC search algorithm, and finally discuss the index construction algorithms.

5.1 Overview of SC-Index

Essentially, an SSC is a structurally reachable component with some core vertices. To efficiently find the structurally reachable components and core vertices in the network, we propose a novel index



(a) SG

Vertex	Core threshold set
a_1	$\{[0.5,0.3]^\top\}$
a_2	$\{[0.5,0.3]^\top\}$
a_3	$\{[0.5,0.3]^\top, [0.4,0.5]^\top\}$
a_4	$\{[0.4,0.5]^\top\}$
a_5	$\{[0.4,0.5]^\top\}$

(b) CTS ($\mu = 3$)

Figure 3: An example of SC-Index.

structure, called SC-Index, which contains the *similarity graph* (SG) and *core threshold set* (CTS). Particularly, the similarity graph G is a homogeneous graph for the vertices with the target type, enabling the rapid expansion of $\boldsymbol{\epsilon}$-neighbors. The CTS of a vertex u, denoted by $\Phi[u]$, is a set of vectors facilitating the rapid determination of whether u is a core vertex or not. We now introduce these two components as follows.

• **Similarity graph (SG)**. The SG is a homogeneous graph $G = (V_0, E_0)$, where V_0 is the set of vertices with the target type and E_0 is the set of edges between vertex pairs whose structural similarity values are larger than zero under a specific similarity measure. Besides, each edge in G is associated with a similarity vector $\boldsymbol{\sigma}(u, v)$. Denote by $N_G[u]$ the set of neighbors of a vertex u in G, which equals to $N_s[u]$ in the HIN. We represent G using adjacency lists, where each vertex u is associated with $N_G[u]$ and their similarity values. Note that for ease of similarity comparison, the neighbors $N_G[u]$ are sorted in descending order according to their similarity values.

Example 2. *Figure 3(a) gives an example SG G constructed from a certain HIN, where the similarity between each pair of vertices is a 2-dimensional vector. For example, $N_G[a_4]$ is {(a_4, $[1.0, 1.0]^\top$), (a_5, $[0.5, 0.5]^\top$), (a_3, $[0.4, 0.6]^\top$), (a_1, $[0.2, 0.3]^\top$)}. The neighbors of each vertex are sorted based on the similarity of the first dimension and then the second.*

• **Core threshold set (CTS)**. A straightforward approach for fast core vertex determination is to pre-compute all the maximum threshold vectors $\boldsymbol{\epsilon}$ under each μ value for each vertex. However, when the number of similarity vector dimensions h is high, the computation of the above approach is extremely prohibitive. For instance, let $|N_G[u]| = 20$, $h = 4$, and $\mu = 5$, we need to enumerate $\binom{4\times5}{5} = 15,504$ neighbor combinations, and compute the maximum threshold values for each of them. Inspired by the notion of k-core [2], we propose the notion of *similarity-based k-core*, or $(\mu, \boldsymbol{\epsilon})$-core, which enables efficient computation and provides analogous boundaries.

Definition 10 ($(\mu, \boldsymbol{\epsilon})$-core). *Given a similarity graph G, a threshold vector $\boldsymbol{\epsilon}$, and an integer μ, the **$(\mu, \boldsymbol{\epsilon})$-core** is the maximum subgraph of G, such that for each vertex in it, the number of its $\boldsymbol{\epsilon}$-neighbors within this subgraph is at least μ.*

In Figure 3(a), let $\boldsymbol{\epsilon} = [0.2, 0.2]^\top$, $\{a_1, \cdots, a_5\}$ constitutes a $(3, \boldsymbol{\epsilon})$-core, while $\{a_1, \cdots, a_7\}$ forms an SSC. The difference between an SSC and $(3, \boldsymbol{\epsilon})$-core lies in the inclusion of the core vertex a_6 and non-core vertex a_7 or not. Clearly, the $(\mu, \boldsymbol{\epsilon})$-core excludes structurally reachable vertices that do have at least μ $\boldsymbol{\epsilon}$-neighbors in the subgraph, i.e., $\forall v \in (\mu, \boldsymbol{\epsilon})$-core, $|N_{\boldsymbol{\epsilon}}[v]| \geq \mu$. Moreover, the $(\mu, \boldsymbol{\epsilon})$-core is a subset of the core vertex set in the SSC, as stated by Property 1. We also observe that the $(\mu, \boldsymbol{\epsilon})$-cores are nested within each other, as presented in Property 2.

Property 1. *Under the same pair of parameters μ and $\boldsymbol{\epsilon}$, the $(\mu, \boldsymbol{\epsilon})$-core is a subset of the core vertex set in an SSC. In other words, if a vertex is in a $(\mu, \boldsymbol{\epsilon})$-core, it is definitely a core vertex.*

Property 2. *Under the same μ, $(\mu, \boldsymbol{\epsilon})$-cores are nested: given two similarity thresholds $\boldsymbol{\epsilon}$ and $\boldsymbol{\epsilon}'$, if $\boldsymbol{\epsilon} < \boldsymbol{\epsilon}'$, then $(\mu, \boldsymbol{\epsilon}')$-core $\subseteq$ $(\mu, \boldsymbol{\epsilon})$-core.*

The key to mining the SSC is to determine the core vertices, and we can utilize Property 1 to achieve this. Thus, we propose to use *core threshold set* (CTS) to store the thresholds required for a vertex to be included in a $(\mu, \boldsymbol{\epsilon})$-core, defined as follows:

Definition 11 (Core threshold set (CTS)). *Given an HIN, a positive integer μ, and an h-dimensional similarity measure, for any vertex u, its **core threshold set** (CTS) is the maximum set of threshold vectors that make vertex u belong to a $(\mu, \boldsymbol{\epsilon})$-core, denoted by $\Phi[u] = \{\boldsymbol{e}^1, \cdots, \boldsymbol{e}^l\}$, where l is the number of threshold vectors.*

Essentially, CTS provides the upper bounds for all $\boldsymbol{\epsilon}$ parameters that make vertex u belong to a non-empty $(\mu, \boldsymbol{\epsilon})$-core. In other words, given any $\boldsymbol{e} \in \Phi[u]$, if $\boldsymbol{\epsilon} > \boldsymbol{e}$, u is not in the $(\mu, \boldsymbol{\epsilon})$-core.

Example 3. *In the similarity graph in Figure 3(a) with $\mu = 3$, $\{a_3, a_4, a_5\}$ forms a $(3, \boldsymbol{\epsilon})$-core, where $\boldsymbol{\epsilon} = [0.4, 0.5]^\top$, so $[0.4, 0.5]^\top$ belongs to $\Phi[a_3]$, $\Phi[a_4]$, and $\Phi[a_5]$, as shown in Figure 3(b).*

Lemma 2. *Given an HIN, an h-dimensional similarity measure ($h \geq 1$), and an integer μ, SC-Index costs $O(hm)$ space, where m is the number of edges in its similarity graph.*

5.2 SC-Index-based SSC search algorithm

In this section, we discuss the SC-Index-based SSC search algorithm. Given an SC-Index, the key idea is to iteratively expand unexplored core vertices and identify all vertices structurally reachable from these core vertices. Specifically, if we can determine core vertices using CTS, we directly expand based on the similarity graph G. Otherwise, we first obtain $N_{\boldsymbol{\epsilon}}[u]$ for core determination and then expand unvisited structurally reachable vertices.

Algorithm 3 implements our idea above. Similar to the online algorithm, we also use a while-loop to complete the SSC expansion process (lines 2-14). For each vertex u in queue Q, if any vector in $\Phi[u]$ is not less than $\boldsymbol{\epsilon}$, we can determine that vertex u is a core vertex (line 6). In the case when vertex u is identified as a core vertex, we traverse and expand each unexplored vertex v in $N_G[u]$ with $\boldsymbol{\sigma}(u, v) \geq \boldsymbol{\epsilon}$ (lines 7-9). In another case, we filter the $\boldsymbol{\epsilon}$-neighbors from $N_G[u]$ and add these uncovered vertices to SSC if $|N_{\boldsymbol{\epsilon}}[u]| \geq \mu$ (lines 11-16). It is worth noting that we first check the core determination using CTS before enqueuing a vertex, resulting in the expansion of core vertices before potential core vertices. Finally, the SSC is obtained when the expansion process terminates.

Example 4. *Consider the SC-Index in Figure 3 with $q = a_5$, $\mu = 3$, and $\boldsymbol{\epsilon} = [0.4, 0.3]^\top$. By Algorithm 3, a_5 is first identified as a core vertex by comparing each vector in $\Phi[a_5]$ with threshold $\boldsymbol{\epsilon}$. Then, the algorithm prioritizes a_3 and a_4 before a_6 in Q, as we can quickly determine that a_3 and a_4 are core vertices based on the CTS shown in Figure*

Algorithm 3: SC-Index-based SSC search algorithm.

Input: SC-Index (G, Φ), a core vertex q, parameters (e.g., $\boldsymbol{\epsilon}, \mu$)
Output: An SSC C containing q

```
1  C ← {q};
2  Q.enqueue(q);
3  while Q ≠ ∅ do
4      u ← Q.dequeue();
5      mark u as explored;
6      if Check e^i ≥ ε (e^i ∈ Φ[u]) then
7          for each vertex v ∈ N_G[u] do
8              if v is not explored and v ∉ C and σ(u,v) ≥ ε then
9                  C ← C ∪ {v}; Q.enqueue(v)
10     else
11         for each vertex v ∈ N_G[u] do
12             if σ(u,v) ≥ ε then N_ε[u] ← N_ε[u] ∪ {v} ;
13         if |N_ε[u]| ≥ μ then
14             for each vertex v ∈ N_ε[u] do
15                 if v is not explored and v ∉ C then
16                     C ← C ∪ {v}; Q.enqueue(v);
17 return C;
```

3(b). Upon expanding vertex a_6, it filters $N_G[a_6]$ and determines a_6 as a non-core, ultimately obtaining the SSC $C = \{a_1, \ldots, a_6\}$.

Lemma 3. *Given an SC-Index, Algorithm 3 completes in $O(m)$ time, where m is the number of edges between vertices of the SSC in the similarity graph.*

5.3 SC-Index construction algorithms

In this section, we present two index construction algorithms, both working in an iterative manner, with the advanced algorithm reducing redundant computation through pruning.

5.3.1 Basic SC-Index construction algorithm. Recall that the SC-Index mainly consists of two components, i.e., similarity graph and core threshold set. The construction of the similarity graph G is intuitive, which involves the computation of structural similarity between each pair of vertices with the target type. The construction of the CTS can be conducted by iteratively increasing the threshold values and updating the current threshold vectors. Specifically, we borrow the idea of k-core decomposition algorithm [2] and leverage the nested property of $(\mu, \boldsymbol{\epsilon})$-core to obtain the CTS of each vertex.

As shown in [2], the k-core decomposition algorithm computes all the k-cores in a homogeneous graph, with a linear time complexity. Specifically, it iteratively deletes vertices whose degrees are less than k and updates the degrees of the remaining incident vertices until all the k-cores are obtained. By considering similarities, we can compute all the $(\mu, \boldsymbol{\epsilon})$-cores through the following two steps: 1) filtering out edges with similarity values smaller than $\boldsymbol{\epsilon}$ in G, and 2) applying k-core decomposition on the filtered G. Obviously, if a vertex u belongs to a $(\mu, \boldsymbol{\epsilon})$-core with threshold at $\boldsymbol{\epsilon}$ but not beyond it, then $\boldsymbol{\epsilon}$ becomes a threshold vector in $\Phi[u]$. Hence, we iteratively compute all the $(\mu, \boldsymbol{\epsilon})$-cores starting from the smallest $\boldsymbol{\epsilon}$, and obtain the CTS of each vertex as their deletions' threshold $\boldsymbol{\epsilon}$. Note that the nested property of the $(\mu, \boldsymbol{\epsilon})$-core avoids the need for recomputation from scratch each time.

Algorithm 4 presents the steps of the idea above. Akin to the online algorithm, we first derive the similarity graph G, by identifying the s-neighbors, calculating their structural similarities,

Algorithm 4: Basic SC-Index construction algorithm.

Input: An HIN $\mathcal{H}(V, E)$, target type Q, a parameter μ
Output: SC-Index of $\mathcal{H}$

```
1  G ← ∅, Φ ← ∅, V_Q ← all the vertices with the target type;
2  for each vertex u ∈ V_Q do
3      obtain N_G[u] = {(v, σ(u,v)) | ψ(v) = ψ(u), σ(u,v) ≥ 0};
4      sort N_G[u] according to the predefined order;
5  c ← obtain similarity sets in G;
6  G' ← G;
7  ε ← [ε_i = c_i^0]^⊤ (i ∈ [1,h]) ;
8  while ε ≠ [c_i^max]^⊤ (i ∈ [1,h]) do
9      Φ ← CTSMono(G', Φ, ε, μ, ∅);
10     if h ≥ 2 then update ε ;
11 return G and Φ
```

Algorithm 5: CTSMono$(G', \Phi, \boldsymbol{\epsilon}, \mu, V_A)$

```
1  G' ← delete all the edges smaller than ε in G';
2  V_c ← compute all the (μ, ε)-core in G';
3  for c_1^t ∈ c_1 in ascending order. do
4      ε' ← ε ⊕ c_1^t;
5      G' ← delete all the edges smaller than ε' in G' ;
6      V_c' ← compute all the (μ, ε')-core in G' ;
7      UpdateCTS(Φ, V_c − V_c', ε);
8      if ε_i' = c_i^max then UpdateCTS(Φ, V_c', ε') ;
9      if V_A ≠ ∅ then
10         G' ← connected component in G' containing V_A;
11         V_c' ← vertex set in G'
12     V_c ← V_c'; ε ← ε';
13 return Φ;
14 Procedure UpdateCTS(Φ, V, ε);
15 for each vertex u ∈ V do
16     for e ∈ Φ[u] do
17         if e < ε then delete e from Φ[u];
18     Φ[u] ← Φ[u] ∪ {ε};
```

and sorting the neighbors for each vertex (lines 2-4). The subsequent parts involve computing the CTS (lines 5-10). Initially, we obtain the set of similarity values c for varying the threshold, where $c = \{c_i = \{c_i^0, c_i^1, \cdots, c_i^{max}\}\}$, with $i \in [1, h]$ (line 5). Afterward, we establish the minimum threshold value and employ a while-loop to execute the threshold traverse process (lines 7-10). In each iteration, we invoke the *CTSMono* algorithm, focusing solely on the variation of the 1^{st} dimension; subsequently, if the number of dimensions exceeds two, we update the threshold until reaching the maximum value. Specifically, the threshold update is achieved by resetting the dimension with the maximum value and incrementing the value of the next dimension. Assuming there are three dimensions, when the second dimension of the threshold$\boldsymbol{\epsilon}_2$ reaches its maximum c_2^{max}, we reset $\boldsymbol{\epsilon}_2 = c_2^0$ and increment $\boldsymbol{\epsilon}_3$.

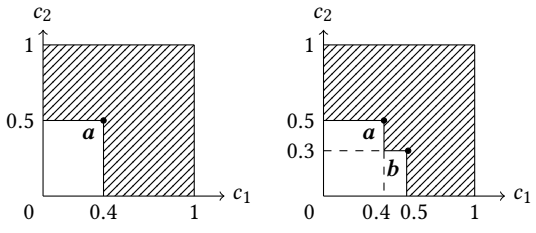
By continuously increasing the threshold of the i^{th} dimension and computing the difference set of $(\mu, \boldsymbol{\epsilon})$-core, *CTSMono* determines the maximum threshold of the i^{th} dimension for all vertices. Here, we use $\oplus c_i^t$ to denote the transformation of $\boldsymbol{\epsilon_i}$ to c_i^t (line 4). During the iteration process, we obtain $(\mu, \boldsymbol{\epsilon})$-core after increasing the threshold, and updating the CTS for the difference set between the new and previous one by invoking procedure *UpdateCTS* (lines

5-7). Here, we examine each element in Φ and eliminate those smaller than the new threshold $\boldsymbol{\epsilon}$, ensuring a complete computation of Φ (lines 15-18). It is crucial to emphasize that the $(\mu, \boldsymbol{\epsilon})$-core vertices with the maximum similarity are stored in V_c', requiring the update of CTS (line 8). If the i^{th} dimension has not reached its maximum value, we replace V_c and $\boldsymbol{\epsilon}$, reusing the results of previously computed $(\mu, \boldsymbol{\epsilon})$-core, as indicated by Property 2 (line 12). The anchored vertex set V_A is used in the subsequent advanced method, and we omit the discussions here.

Example 5. *Assume we have derived G as depicted in Figure 3(a). Algorithm 4 first obtains the similarity set c, sets threshold $\boldsymbol{\epsilon}$ as the minimum value, and then incrementally raises $\boldsymbol{\epsilon}$. When $\boldsymbol{\epsilon}'$ becomes $[0.5, 0.3]^\top$, a_4 and a_5 are removed from G, and it temporarily stores $\boldsymbol{\epsilon} = [0.4, 0.3]^\top$ in $\Phi[a_4]$ and $\Phi[a_5]$. Next, $[0.5, 0.3]^\top$ is added to $\Phi[a_1]$ and $\Phi[a_3]$, since $\boldsymbol{\epsilon}'$ reaches the maximum value of the first dimension. Following this, when $\boldsymbol{\epsilon}' = [0.4, 0.5]$, the $\Phi[a_3]$ and $\Phi[a_5]$ are updated, replacing the previous threshold for a_4 and a_5. Figure 3(b) shows all the CTS.*

Lemma 4. *Given an HIN, Algorithm 4 costs $O(\alpha n + \beta m + \gamma^{h-1} m)$ time, where γ is the maximum number of different similarity values among h dimensions.*

5.3.2 Advanced SC-Index construction algorithm. Although the basic algorithm provides a correct method to find all CTS, it involves many redundant computations. Consider a vertex u with $\Phi[u] = \{\boldsymbol{a} = [\boldsymbol{a}_1, \boldsymbol{a}_2]^\top\}$. In the basic algorithm, for all the thresholds $\boldsymbol{\epsilon}$ satisfying $\boldsymbol{\epsilon}_2 < \boldsymbol{a}_2$, u is always involved in the core decomposition computation of these different $(\mu, \boldsymbol{\epsilon})$-cores in *CTSMono* and $\Phi[u]$ is updated in the procedure *UpdateCTS* with threshold $[\boldsymbol{a}_1, \boldsymbol{\epsilon}_2]^\top$. To alleviate this issue, we propose an advanced SC-Index construction algorithm, aimed at optimizing the computation of CTS, particularly for the cases where the number of dimensions exceeds two, i.e., $h \geq 2$. It adopts a reverse traversal of the threshold $\boldsymbol{\epsilon}$, aiming to directly obtain the maximum threshold vectors.



(a) Before update (b) After update

Figure 4: An example search area of vertex a_3.

In the advanced algorithm, to avoid redundant core decomposition computations, we adopt a descending order for updating the threshold $\boldsymbol{\epsilon}$ with pruning strategies, as shown in Algorithm 6. To identify unnecessary computations and leverage Property 2 for $h \geq 2$, we introduce two concepts: *search area* and *anchored vertex*:

- *Search area:* When computing CTS for a vertex u, its search area, denoted by $sa[u]$, is a 2-dimensional space with the 1^{st} and 2^{nd} dimensions formed by all similarity values greater than the corresponding values in the current $\Phi[u]$, i.e., $sa[u] = \{(c_1, c_2) | \forall \boldsymbol{e} \in \Phi[u]$ s.t. $c_1 > \boldsymbol{e}_1$ or $c_2 > \boldsymbol{e}_2\}$.
- *Anchored vertex:* During the iterative computation, if a vertex u has not visited its neighbors v and their 1^{st} and 2^{nd} dimension similarity falls in $sa[u]$, then u is called an anchored vertex.

Algorithm 6: Advanced SC-Index construction algorithm.

Input: An HIN $\mathcal{H}(V, E)$, target type Q, a parameter μ
Output: SC-Index of $\mathcal{H}$

1 Compute G and c (same as that of Algorithm 4);
2 $\boldsymbol{\epsilon} \leftarrow [\boldsymbol{\epsilon_i} = c_i^0]^\top$ $(i \in [1, h])$;
3 $G' \leftarrow G$;
4 **if** $h = 1$ **then** $\Phi \leftarrow$ CTSMono$(G', \Phi, \boldsymbol{\epsilon}, \mu, \emptyset)$; **return** G and Φ;
5 **if** $h = 2$ **then** $\Phi \leftarrow$ CTSDual$(G', \Phi, \boldsymbol{\epsilon}, \mu)$; **return** G and Φ;
6 $\boldsymbol{\epsilon} \leftarrow [\boldsymbol{\epsilon_i} = c_i^{max}]^\top$ $(i \in [1, h])$;
7 **while** $\boldsymbol{\epsilon} \neq [c_i^0]^\top (i \in [1, h])$ **do**
8 $\quad \Phi \leftarrow$ CTSDual$(G', \Phi, \boldsymbol{\epsilon}, \mu)$;
9 $\quad$ reverse update $\boldsymbol{\epsilon}$;
10 **return** G and Φ

Algorithm 7: CTSDual$(G', \Phi, \boldsymbol{\epsilon}, \mu)$

1 $G' \leftarrow$ delete all the edges smaller than $\boldsymbol{\epsilon}$ in G ;
2 $V_S' \leftarrow$ compute all the $(\mu, \boldsymbol{\epsilon})$-core in G' ;
3 $V_A \leftarrow \emptyset$;
4 **for** *each vertex* $u \in V_S'$ **do**
5 $\quad$ **for** *each vertex* $v \in N_{G'}[u]$ **do**
6 $\quad\quad$ **if** $G'(u, v)$ *is not visited* **and** $\boldsymbol{\sigma}(u, v)_{i,j}$ *in* $\mathbf{sa}[u]$ **then**
7 $\quad\quad\quad V_A \leftarrow V_A \cup \{u\}$; mark $G'(u, v)$ as visited;
8 **for** $c_j^t \in c_j$ *in descending order.* **do**
9 $\quad \boldsymbol{\epsilon} \leftarrow \boldsymbol{\epsilon} \oplus c_j^t$;
10 $\quad$ update anchored vertex set V_A and search area;
11 $\quad G' \leftarrow$ connected component in G' containing V_A;
12 $\quad \Phi \leftarrow$ CTSMono$(G', \Phi, \boldsymbol{\epsilon}, \mu, V_A)$;
13 **return** Φ;

As we update the threshold $\boldsymbol{\epsilon}$ in reverse order, the values of the lowest two dimensions of the current CTS are always smaller than the newly added threshold. Thus, the search area delineates the region where the next threshold vector of CTS will be added during the computation process, while anchored vertices exclude vertices that will not undergo updates.

Example 6. *In Figure 3(a), $\Phi[a_3] = \{\boldsymbol{a}, \boldsymbol{b}\}$ with $\boldsymbol{a} = [0.4, 0.5]^\top$ and $\boldsymbol{b} = [0.5, 0.3]^\top$. As we perform a descending search of the threshold, we encounter $\boldsymbol{a}$ first, forming the search area shown in Figure 4(a). As $\boldsymbol{\epsilon}_2$ becomes to 0.3, since $\boldsymbol{\sigma}(a_3, a_1) = [0.5, 0.3]^\top$ falls within the search area, both a_1 and a_3 become anchored vertices. After computing $\Phi[a_1]$, $\Phi[a_2]$, and $\Phi[a_3]$ with $\boldsymbol{b}$ included, the updated search area for a_3 is depicted in Figure 4(b). Because a_4 and a_5 are not involved in the computation, the redundant computation of $\Phi[a_4]$ and $\Phi[a_5]$ is avoided.*

Algorithm 7 lists the steps of the utilization and updates of the search space. First, it computes all the $(\mu, \boldsymbol{\epsilon})$-cores based on the current threshold $\boldsymbol{\epsilon}$ (lines 1-2), and then obtains the anchored vertex set V_A according to the similarity of the G (lines 3-7). Subsequently, it iteratively computes the CTS in descending order and updates the anchored set V_A and the search area accordingly (lines 8-12). The key step to reduce redundant computations is to remove the $(\mu, \boldsymbol{\epsilon})$-core in G' that does not contain V_A. This is because if the similarity of edges connected to vertex u does not fall within the search area, then the $\Phi[u]$ will not change during the current iteration of core

Algorithm 8: Online SSC detection algorithm.

Input: An HIN $\mathcal{H}(V, E)$, parameters (e.g., ϵ and μ)
Output: All SSCs $\mathbb{C}$, hubs $\mathbb{H}$, outliers $\mathbb{O}$

1 $\mathbb{C} \leftarrow \emptyset, \mathbb{H} \leftarrow \emptyset, \mathbb{O} \leftarrow \emptyset$;
2 $V_C \leftarrow \emptyset$, $V_Q \leftarrow$ all the vertices with the target type;
3 **for** *each vertex* $u \in V_Q$ **do**
4 **if** $u \in V_C$ ***or*** *CheckCore(u) is false* **then continue**;
5 $C \leftarrow$ search the SSC containing u by Algorithm 1;
6 $\mathbb{C} \leftarrow \mathbb{C} \cup \{C\}$;
7 $V_C \leftarrow V_C \cup C$;
8 **for** *each non-core vertex* $v \in C$ **do** mark v as unexplored;
9 **for** *each vertex* $u \in V_Q \backslash V_C$ **do**
10 $X \leftarrow \emptyset$;
11 **for** *each vertex* $v \in N_s[u]$ **do**
12 add the SSC containing v to X;
13 **if** $|X| \geq 2$ **then** $\mathbb{H} \leftarrow \mathbb{H} \cup \{u\}$;
14 **else** $\mathbb{O} \leftarrow \mathbb{O} \cup \{u\}$;
15 **return** $\mathbb{C}, \mathbb{H}, \mathbb{O}$;

decomposition on G'. Consequently, when all anchored vertices are removed from any (μ, ϵ)-core, this (μ, ϵ)-core should be excluded from further computation, regardless of whether it is empty or not. Hence, we continuously update the anchored vertex set V_A and the connected component G' containing anchored vertices in lines 9-10 of Algorithm 7 and line 10 of Algorithm 5.

Lemma 5. *Algorithm 6 correctly computes all the CTSs for vertices with the target type.*

Lemma 6. *Given an HIN $\mathcal{H}$, Algorithm 6 completes in $O(\alpha n + \beta m + \gamma^e m)$ time, where $e = max\{h - 2, 0\}$.*

Compared to the basic algorithm, the advanced index construction algorithm adopts reverse order traversal of threshold and constructs a search area for each vertex. Filtering the subgraphs associated with anchored vertices effectively reduces the number of vertices involved in each core decomposition computation and ensures that each updated threshold is always the maximum one.

• Optimizations for similarity computation. Both our online algorithms and SC-Index construction algorithms require the computation of structural similarity between vertices, which is costly. To speed up this process, we develop three kinds of optimization for similarity computation: (1) During the index construction, we can parallelize the computation of similarity between vertex pairs with similarity values larger than zero. (2) We introduce an early termination strategy: if the similarity value of a given dimension falls below the threshold during computation, we halt the calculation for the remaining dimensions. (3) Since the similarity calculation is orthogonal to CS and CD, optimizations for any specific similarity measure can be seamlessly integrated into our solution. We briefly review existing optimization methods [5, 6, 45, 47, 76, 90] in the appendix of our technical report [80].

5.4 Extension to SSC Detection

In this section, we show that both the online and index-based SSC search algorithms can be easily extended to solve the SSC detection problem. The main idea of SSC detection algorithms is to repeatedly invoke the SSC search algorithms for each unexplored vertex, to obtain hubs, outliers, and SSCs.

Algorithm 8 presents the online SSC detection algorithm. We use V_C to collect all the vertices in the SSCs, which is empty initially, and use V_Q to include all the vertices with the target type (line 2). For each core vertex, we invoke the online search algorithm to search the SSC containing it (lines 3-8). Note that when a vertex is already in another SSC or is not a core vertex, we will not invoke Algorithm 1 for it to avoid duplicated search (line 4). After obtaining an SSC, all its vertices are added to V_C and and all non-core vertices are marked as unexplored because they may belong to different SSCs (lines 7-8). For the remaining vertices that are not in any SSC, we classify them as either outlier vertices or hub vertices by checking the SSCs containing their s-neighbors (lines 9-15).

In addition, the SC-Index can be used to support the SSC detection. Specifically, we just need to slightly modify Algorithm 8 by replacing the search process in line 5 with Algorithm 3, and replacing $N_s[u]$ in line 11 with $N_G[u]$.

6 EXPERIMENTS

We now present the experimental results. Section 6.1 discusses the setup. We discuss the experimental results in Sections 6.2 and 6.3.

6.1 Setup

Table 2: Datasets used in our experiments.

Dataset	Vertices	Edges	Vertex types	Edge types
Amazon	13,114	90,340	4	3
DBLP	682,819	3,902,418	4	3
IMDB	2,467,806	15,195,182	4	3
DBpedia	5,900,558	35,282,572	413	637
Freebase	29,119,948	108,989,856	984	2,580

Datasets. We use five real-world HIN datasets: Amazon[2], DBLP[3], IMDB[4], DBpedia[5], and Freebase[6]. The detailed statistics are shown in Table 2, which includes the numbers of vertices, edges, vertex types, and edge types. Amazon is a network of product evaluations, documenting the relationships among users, products, brands, and views. DBLP archives publication records in computer science areas, and the vertex types are authors, papers, topics, and venues. IMDB includes movie rating records since 2000, and it has four types of vertices (authors, directors, writers, and movies). DBpedia contains the data extracted from Wikipedia infoboxes using the mapping-based extraction (object properties only). Freebase contains all the entities and relations under the domains of music, film, and TV.

Algorithms. In our experimental evaluation, we evaluate and compare the following CS algorithms:

- `SSCS`: our online SSC search algorithm discussed in Sec. 4.
- `SC-Q`: our SC-Index-based SSC search algorithm in Sec. 5.2.
- `CSH`: a CS algorithm using $(k, \mathcal{P})$-core for HINs in [27].
- `CSSH`: a CS algorithm for star-schema HINs [41].
- `CSH-E`: a CS approach using edge-disjoint $(k, \mathcal{P})$-core [27].
- `CSH-V`: a CS approach using vertex-disjoint $(k, \mathcal{P})$-core [27].

[2] http://jmcauley.ucsd.edu/data/amazon/
[3] http://dblp.uni-trier.de/xml/
[4] https://www.imdb.com/interfaces/
[5] https://wiki.dbpedia.org/Datasets
[6] https://freebase-easy.cs.uni-freiburg.de/dump/

We also evaluate and compare the following CD algorithms:

- SSCD: the online SSC detection algorithm in Sec. 5.4.
- SCAN-$\mathcal{P}$: we first induce the HIN to a homogeneous graph using a symmetric meta-path $\mathcal{P}$ and then run SCAN algorithm on it [7].
- k-Btruss: a k-truss-based CD method over HINs [86].

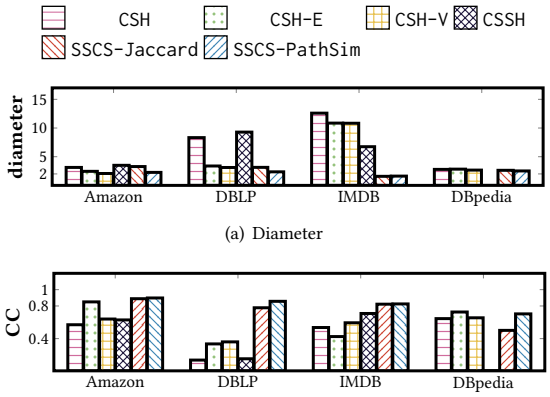
As reviewed in Section 2, our SSCS belongs to the first group of existing CS works that focus on HINs without additional attributes. Apart from the above CS methods, r-com [40] also belongs to the first group, but it finds communities involving multiple vertex types, while CSH, CSSH, and our approach focuses on communities with a single target vertex type, so we omit it in our experiments.

Parameter settings. For parameter settings, we drew inspiration from the existing works on SCAN [6, 83] and CS [27, 41], opting for a similar settings. For $\mu \geq 2$, we choose 3, 5, 7, 9, 11, 13, and 15, with 5 as default. For $0 < |\epsilon| \leq 1$, we set all dimensions to a uniform value and choose 0.2, 0.4, 0.6, and 0.8, with 0.2 as the default. In practice, we can first sample the similarity values between vertices in HINs and use the mean as the default value, then adjust it based on specific requirements. For the similarity measure, we implement our SSC model using two widely used metrics: Jaccard and PathSim, resulting in SSCS-Jaccard and SSCS-PathSim, respectively. By default, SSCS-PathSim sets ϵ to a uniform value of 0.4. If not specified otherwise, we use Jaccard similarity as the default. To mitigate the risk of an extensive selection leading to null results, we set $k = [1, 2]$ in k-multi-strata, with 2 being the default value if the maximum distance is at least 2. We set all the possible vertex types for the first three datasets as the target type. For the two knowledge graph HINs, we use the top 10 vertex types with the highest vertex frequencies. The default core number k for CSH, CSH-E, CSH-V, and CSSH is set to 5, with $k = \mu$ used in the efficiency experiments. In line with existing works [27, 39, 70], for baselines using meta-paths, we focus on meta-paths with lengths up to four and select all the possible meta-paths for the first three datasets; for the rest two datasets, we select the top-1000 meta-paths with the highest frequencies, ensuring that each chosen meta-path starts and ends with the target type. To evaluate the SSC search algorithms, we randomly select 1,000 query vertices, each of which is contained by an SSC. We implement our all algorithms in C++ with -O3 level optimization and all the experiments are conducted on a Linux operating system running on a machine with an Intel Xeon 2.4GHz CPU and 512GB of memory. If an algorithm cannot finish in three days, we mark its running time as INF.

6.2 Effectiveness evaluation

We analyze the quality of SSCs from the following aspects. Due to space limitations, additional experimental results are provided in the appendix of our technical report [80].

• **Closeness of community members.** To measure the closeness of communities, a commonly used metric is the diameter [38], or the largest shortest distance between any pair of vertices in the subgraph of the community. To adapt it for communities in HINs, we follow the diameter measure in [27], which is defined on the $\mathcal{P}$-distance (the $\mathcal{P}$-distance between two vertices linked by an instance of $\mathcal{P}$ is 1). We report the average diameter of communities found by different methods in Figure 5(a). We observe that the communities of SSC model have smaller diameters. For example, on IMDB, the average diameter for CSH is 12.6, while for SSCS it is 1.58. Hence, we can conclude that the communities modeled by SSC tend to have closer relationships between their vertices.

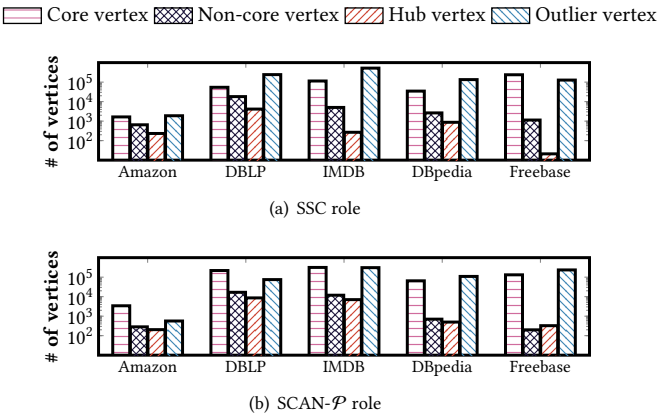


(a) Diameter

(b) CC

Figure 5: Diameters and CC values of communities.

• **Clustering coefficient (CC) of communities.** The CC measures the tightness of connections between vertices, indicating the proportion of connections among the neighbors of vertices. A higher CC value of a community implies that its vertices are connected more tightly. We utilize $\mathcal{P}$-induced homogeneous and report the CC values for different models in Figure 5(b). We can observe that on most datasets, the communities of SSC model achieve significantly higher CC values than communities of the other two models. In addition, on DBpedia, communities of SSCS achieve smaller CC values due to the lack of consideration of some long-distance types, but SSCS-PathSim, which adopts PathSim measures, obtains communities with higher CC values.



(a) SSC role

(b) SCAN-$\mathcal{P}$ role

Figure 6: Community roles analysis.

• **SSC roles analysis.** Figure 6 shows the number of vertices with four different roles obtained by SSCD and SCAN-$\mathcal{P}$ across all datasets under default parameters. Clearly, on all datasets, the number of core vertices is much larger than that of non-core vertices, while hub vertices are the least numerous, with the remaining vertices being outlier vertices. SSC vertices (comprising of core vertices and non-core vertices) typically constitute 20% ∼ 60% of the total vertices across different datasets. Note that by adjusting the parameters, we

can vary the proportion of SSC vertices, reducing hub vertices and outlier vertices.

• **A case study.** We conduct a case study to identify the SSC containing Prof. Li Fei-Fei, a prominent scholar in the fields of computer vision and deep learning. On a small DBLP network with 467,511 vertices and 3,666,418 edges (randomly extracted from the original DBLP network), we set $\mu = 5$ and choose two different $\boldsymbol{\epsilon}$ to capture distinct semantic interpretations. Figure 7 depicts the two SSCs obtained, where the $\boldsymbol{\epsilon}$ for C_1 on the left is $[0.095, 0.3, 0.15]^\top$ (Paper, Venue, Topic), and the $\boldsymbol{\epsilon}$ for C_2 on the right is $[0.05, 0.3, 0.21]^\top$. The scholars with blue circles are core vertices, and the edges in the graph denote that the similarity between them exceeds the threshold. By examining Prof. Li's collaborated publications, we find that for C_1, these scholars exhibit a closer collaboration, with higher paper similarity. Conversely, in C_2, although the threshold value of paper type is not as high as that in C_1, there exists a higher threshold for topic similarity, indicating a shared interest in their research areas. From this case study, it is evident that our SSC model can capture communities with semantic richness.

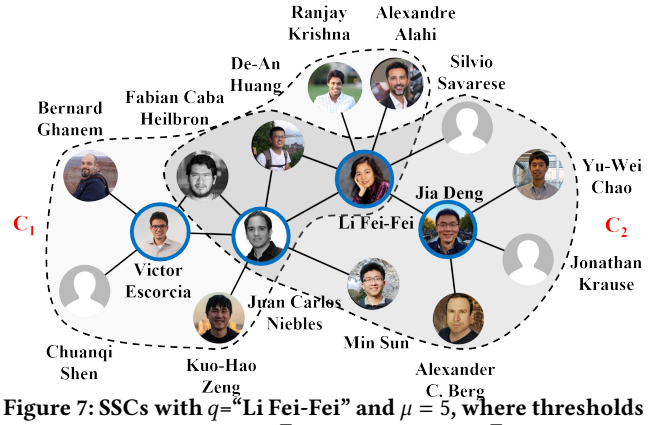
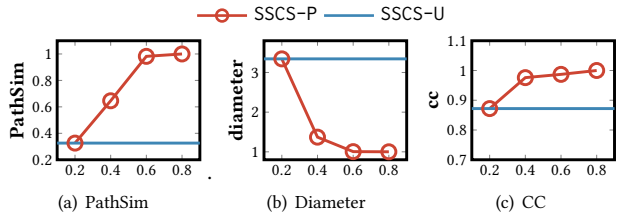


Figure 7: SSCs with q="Li Fei-Fei" and $\mu = 5$, where thresholds $\boldsymbol{\epsilon}$ are set to $[0.095, 0.3, 0.15]^\top$ and $[0.05, 0.3, 0.21]^\top$ respectively.



(a) PathSim (b) Diameter (c) CC

Figure 8: Threshold vector test.

• **Threshold vector analysis.** To evaluate the effect of different threshold vectors, we compare the communities returned by different query threshold vectors on the DBLP dataset. We use two variants: SSCS-U, where all dimensions are set to a uniform value, and SSCS-P, where the paper dimension is varied while the other two dimensions remain uniform. The query vertex type is set to *Author*, and Jaccard similarity is used for computing the SSC. As shown in Figure 8, The results of SSCS-U remain unchanged since the values of the three dimensions of $\boldsymbol{\epsilon}$ are set to 0.2. We observe that when increasing the threshold values in the paper dimension, the communities achieve higher PathSim values which reflect higher semantic similarity in paper dimension, and also have smaller diameters and higher CC values which demonstrate a close relationship between authors. This indicates that our approach, which utilizes a threshold vector, can capture communities with more accurate semantic relevance.

Table 3: F1-Score values of different methods on S-DBLP.

Algorithm	k or μ values				
	5	6	7	8	9
CSH	0.066	0.08	0.074	0.048	0.098
CSSH	0.083	0.089	0.048	0.048	0.048
SCAN-$\mathcal{P}$	0.053	0.059	0.071	0.072	0.077
k-Btruss	0.05	0.058	0.056	0.054	0.053
SSCD	**0.123**	**0.123**	**0.131**	**0.131**	**0.145**

• **F1-scores.** To evaluate the effectiveness of our SSC model, we compare its performance with existing CS and CD methods on a small DBLP dataset (S-DBLP) containing ground-truth communities. For CS methods, we use one vertex from each ground-truth community as the query vertex to search for its corresponding community and compute the average F1 score across all communities. S-DBLP includes publications from major conferences between 2019 and 2021, comprising 19,316 papers, 21,164 authors, 16 conferences, and 13,795 topics. We categorize authors into different communities and outliers according to the following steps: First, conferences are grouped into five categories based on research areas. Then, authors are divided into different communities based on the categories of the conferences they published in, the topics mentioned in their papers, and their research interests. Additionally, authors with significantly few publications and a low h-index are identified as outliers. On average, each community has about 150 authors. Following previous studies [36, 37], we calculate the best F1-scores for each method under different k or μ values, as shown in Table 3. SSCD consistently achieves the highest F1-scores, demonstrating that it is able to well detect the ground-truth communities.

6.3 Efficiency evaluation

In this section, we evaluate the efficiency of SSC search. We skip the results of SSC detection as it repeatedly calls SSC search algorithms.

• **Efficiency of SSC search algorithms.** We compare the efficiency of both online and index-based SSC search algorithms under various parameter settings. Figure 9 depicts the efficiency of SSCS and SC-Q by varying $\boldsymbol{\epsilon}$. Clearly, as $\boldsymbol{\epsilon}$ continues to increase, the time cost of SC-Q consistently decreases and remains significantly faster than SSCS. For example, on DBLP, SC-Q spends 0.1 seconds while SSCS needs 80 seconds when $\boldsymbol{\epsilon} = \mathbf{0.2}$. As $\boldsymbol{\epsilon}$ increases to **0.8**, the runtime decreases to 27 milliseconds for SC-Q and 5.4 seconds for SSCS. This is primarily due to the reduction in community size as $\boldsymbol{\epsilon}$ increases, leading to an increase in cases where the SSC cannot be found and the acceleration of SC-Q diminishes.

Besides, Figure 10 illustrates the effect of μ on performance, with $k = \mu$ applied for other baseline CS methods. These baseline methods are slower or comparable to SSCS and significantly slower than SC-Q. Their comparable runtimes to SSCS stem from failing to identify communities for certain query vertices, which reduces computation time. We see that SC-Q is up to three orders of magnitude faster than SSCS. For instance, on IMDB, when $\mu = 9$, SC-Q takes 0.297 seconds, while SSCS costs 309 seconds. Thus, the

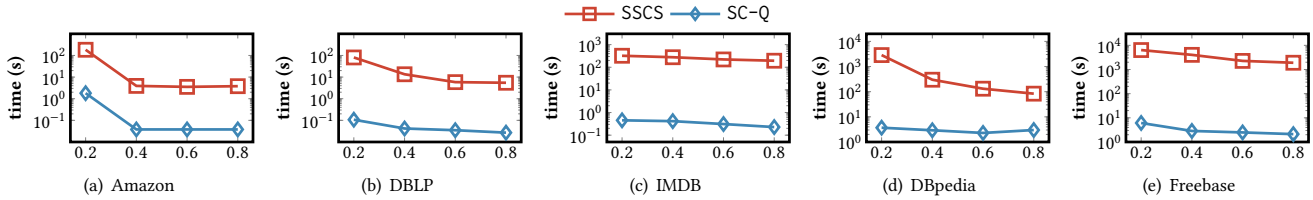


Figure 9: SSC search efficiency by varying $\epsilon(\mu = 5)$.

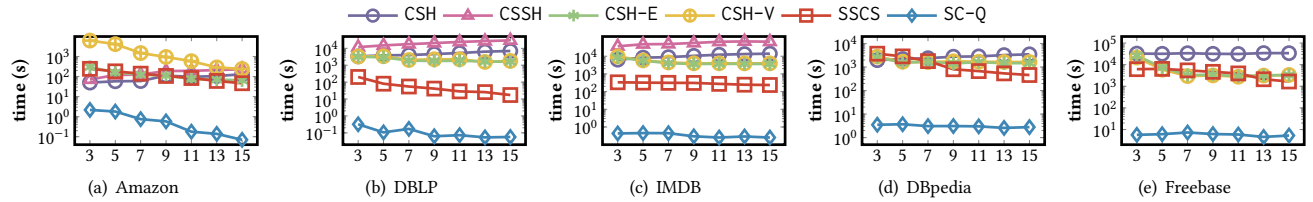


Figure 10: SSC search and baselines search efficiency by varying $\mu(\epsilon = 0.2)$ and the core number, respectively.

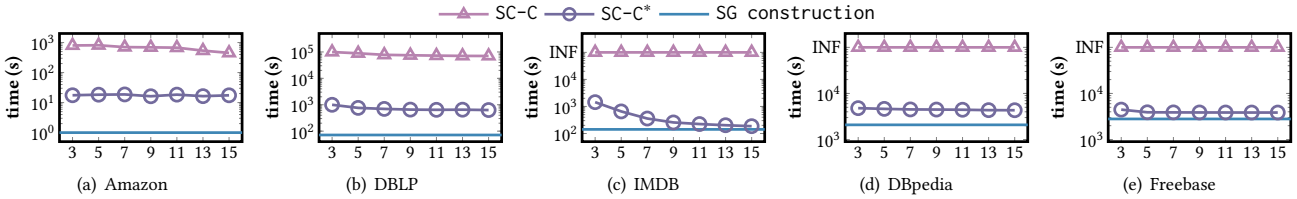


Figure 11: Index construction efficiency by varying μ.

core determination and structural reachability expansion offered by SC-Index are effective for improving efficiency.

• **Efficiency of index construction algorithms.** The efficiency of two index construction algorithms is presented in Figure 11, where SC-C and SC-C* denote the basic and advanced algorithms respectively. The blue line represents the time cost of the computation of Similarity Graph, which constitutes only a small portion of the overall time cost. We see that with the increase in μ, the runtime of both algorithms continuously decreases. Particularly, SC-C* is up to two orders of magnitude faster than SC-C because it significantly reduces the computational cost of CTS by avoiding the redundant core decomposition. For instance, on DBLP with $\mu = 5$, SC-C requires 88,753 seconds, whereas SC-C* completes in only 689 seconds. Furthermore, on the last three datasets, SC-C has exceeded time limitations for certain vertex types.

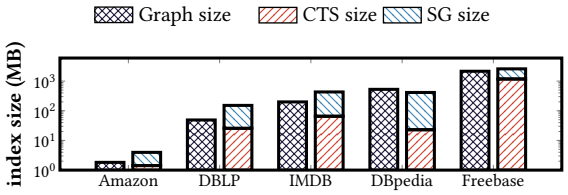


Figure 12: Index space cost.

• **Index space cost.** In this experiment, we compare the space cost of the SC-Index on all datasets, as illustrated in Figure 12. Note that the CTS size is the sum of results under 7 different μ values. We observe that on most datasets, the storage space cost of our SC-Index is just slightly larger than that of the original input HIN. For example, on Freebase, with a graph size of 2,153MB, the space cost of SC-Index is just 2,259MB. Thus, SC-Index is space efficient.

• **Scalability test.** For each dataset, we randomly select 20%, 40%, 60%, 80%, and 100% vertices and obtain five sub-HINs induced by these vertices respectively. Figure 13 reports the time cost of our two index construction algorithms on these sub-HINs. We can observe that the time cost of the advanced one SC-C* increases linearly with the size of the dataset, demonstrating its scalability in processing large HINs. Again, SC-C* is much faster than SC-C, but SC-C times out for large sub-HINs of IMDB dataset. Note that the blue lines represent the runtime of the computation of SG, which constitutes only a small portion of the overall time.

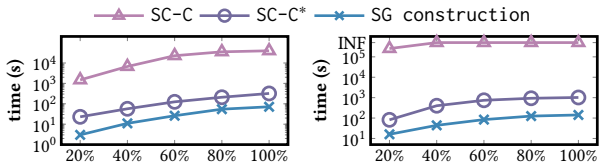


Figure 13: Scalability test.

7 CONCLUSION

Community mining is a fundamental problem in network science. Inspired by the SCAN algorithm, in this paper, we propose a novel community model, called structurally similar community (SSC), which is not only generic to support various similarity measures but also able to identify the roles of vertices such as hubs and outliers. To efficiently search the SSC containing a query vertex, we develop an efficient online solution and an index-based solution to enhance the search performance. We also develop efficient index construction algorithms. Both the online and index-based solutions can be easily extended for detecting all the SSCs. Experimental results on five real-large HINs demonstrate the high effectiveness and efficiency of our proposed solutions. In the future, we will study efficient parallel and distributed algorithms to further improve the efficiency of SSC search and detection.

REFERENCES

[1] Leman Akoglu, Hanghang Tong, and Danai Koutra. 2015. Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery* 29 (2015), 626–688.

[2] V. Batagelj and M. Zaversnik. 2003. An O(m) Algorithm for Cores Decomposition of Networks. arXiv:cs/0310049

[3] Johannes Berg and Michael Lässig. 2006. Cross-Species Analysis of Biological Networks by Bayesian Alignment. *Proceedings of the National Academy of Sciences* 103, 29 (July 2006), 10967–10972.

[4] Francesco Bonchi, Arijit Khan, and Lorenzo Severini. 2019. Distance-Generalized Core Decomposition. In *Proceedings of the 2019 International Conference on Management of Data*. ACM, 1006–1023.

[5] Andrei Z Broder. 1997. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*. IEEE, 21–29.

[6] Lijun Chang, Wei Li, Xuemin Lin, Lu Qin, and Wenjie Zhang. 2016. pSCAN: Fast and Exact Structural Graph Clustering. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, 253–264.

[7] Lijun Chang, Wei Li, Lu Qin, Wenjie Zhang, and Shiyu Yang. 2017. TKDE – pSCAN: Fast and Exact Structural Graph Clustering. *IEEE Transactions on Knowledge and Data Engineering* 29, 2 (Feb. 2017), 387–401.

[8] 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 (SIGMOD '15)*. ACM, 459–474.

[9] Yulin Che, Shixuan Sun, and Qiong Luo. 2018. Parallelizing Pruning-based Graph Structural Clustering. In *Proceedings of the 47th International Conference on Parallel Processing (ICPP '18)*. ACM, New York, NY, USA, 1–10.

[10] 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)*. 256–267.

[11] Lu Chen, Chengfei Liu, Xiaochun Yang, Bin Wang, Jianxin Li, and Rui Zhou. 2016. Efficient Batch Processing for Multiple Keyword Queries on Graph Data. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management (CIKM '16)*. ACM, 1261–1270.

[12] Lu Chen, Chengfei Liu, Rui Zhou, Jiajie Xu, Jeffrey Xu Yu, and Jianxin Li. 2020. Finding Effective Geo-social Group for Impromptu Activities with Diverse Demands. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '20)*. ACM, New York, NY, USA, 698–708.

[13] Wenyu Chen. 2023. Relationship Prediction Based Anomaly Detection in Heterogeneous Information Networks. In *2023 4th International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT)*. 206–210.

[14] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. 2004. Finding community structure in very large networks. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 70, 6 (2004), 066111.

[15] Alessio Conte, Gaspare Ferraro, Roberto Grossi, Andrea Marino, Kunihiko Sadakane, and Takeaki Uno. 2018. Node Similarity with q -Grams for Real-World Labeled Networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, London United Kingdom, 1282–1291.

[16] 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 (SIGMOD '13)*. ACM, New York, NY, USA, 277–288.

[17] 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 (SIGMOD '14)*. ACM, New York, NY, USA, 991–1002.

[18] Pedro Domingos and Matt Richardson. 2001. Mining the Network Value of Customers. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, San Francisco California, 57–66.

[19] W. E. Donath and A. J. Hoffman. 1973. Lower Bounds for the Partitioning of Graphs. *IBM Journal of Research and Development* 17, 5 (Sept. 1973), 420–425.

[20] Zheng Dong, Xin Huang, Guorui Yuan, Hengshu Zhu, and Hui Xiong. 2021. Butterfly-core community search over labeled graphs. *Proc. VLDB Endow.* 14, 11 (2021), 2006–2018.

[21] Jordi Duch and Alex Arenas. 2005. Community detection in complex networks using extremal optimization. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 72, 2 (2005), 027104.

[22] 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.

[23] Soroush Ebadian and Xin Huang. 2019. Fast algorithm for K-truss discovery on public-private graphs. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. 2258–2264.

[24] Yixiang Fang, Reynold Cheng, Siqiang Luo, and Jiafeng Hu. 2016. Effective Community Search for Large Attributed Graphs. *Proc. VLDB Endow.* 9, 12 (Aug. 2016), 1233–1244.

[25] 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 (2020), 353–392.

[26] Yixiang Fang, Kai Wang, Xuemin Lin, and Wenjie Zhang. 2021. Cohesive Subgraph Search over Big Heterogeneous Information Networks: Applications, Challenges, and Solutions. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21)*. ACM, 2829–2838.

[27] Yixiang Fang, Yixing Yang, Wenjie Zhang, Xuemin Lin, and Xin Cao. 2020. Effective and Efficient Community Search over Large Heterogeneous Information Networks. *Proc. VLDB Endow.* 13, 6 (Feb. 2020), 854–867.

[28] Santo Fortunato. 2010. Community Detection in Graphs. *Physics Reports* 486, 3 (Feb. 2010), 75–174.

[29] Santo Fortunato and Darko Hric. 2016. Community Detection in Networks: A User Guide. *Physics Reports* 659 (Nov. 2016), 1–44.

[30] 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 (CIKM '17)*. ACM, 1807–1816.

[31] Zheng Gao and Xiaozhong Liu. 2017. Personalized Community Detection in Scholarly Network. *iConference 2017 Proceedings Vol. 2* (2017).

[32] Andrey Goder and Vladimir Filkov. 2008. Consensus Clustering Algorithms: Comparison and Refinement. In *2008 Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX)*. Society for Industrial and Applied Mathematics, 109–117.

[33] Malik Khizar Hayat and Ali Daud. 2017. Anomaly Detection in Heterogeneous Bibliographic Information Networks Using Co-Evolution Pattern Mining. *Scientometrics* 113, 1 (Oct. 2017), 149–175.

[34] Jiafeng Hu, Reynold Cheng, Kevin Chen-Chuan Chang, Aravind Sankar, Yixiang Fang, and Brian Y.H. Lam. 2019. Discovering Maximal Motif Cliques in Large Heterogeneous Information Networks. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. 746–757.

[35] 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 (CIKM '16)*. ACM, 1241–1250.

[36] 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.

[37] Xin Huang and Laks V. S. Lakshmanan. 2017. Attribute-driven community search. *Proc. VLDB Endow.* 10, 9 (may 2017), 949–960.

[38] Xin Huang, Laks V. S. Lakshmanan, Jeffrey Xu Yu, and Hong Cheng. 2015. Approximate Closest Community Search in Networks. *Proc. VLDB Endow.* 9, 4 (Dec. 2015), 276–287.

[39] 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*. ACM, 1595–1604.

[40] Xun Jian, Yue Wang, and Lei Chen. 2020. Effective and Efficient Relational Community Detection and Search in Large Dynamic Heterogeneous Information Networks. *Proc. VLDB Endow.* 13, 10 (June 2020), 1723–1736.

[41] Yangqin Jiang, Yixiang Fang, Chenhao Ma, Xin Cao, and Chunshan Li. 2022. Effective Community Search over Large Star-Schema Heterogeneous Information Networks. *Proc. VLDB Endow.* 15, 11 (July 2022), 2307–2320.

[42] Yuli Jiang, Xin Huang, and Hong Cheng. 2021. I/O Efficient k-Truss Community Search in Massive Graphs. *The VLDB Journal* 30, 5 (Sept. 2021), 713–738.

[43] Jon M. Kleinberg. 1999. Authoritative Sources in a Hyperlinked Environment. *J. ACM* 46, 5 (Sept. 1999), 604–632.

[44] Johan H. Koskinen and Tom A.B. Snijders. 2007. Bayesian Inference for Dynamic Social Network Data. *Journal of Statistical Planning and Inference* 137, 12 (Dec. 2007), 3930–3938.

[45] Ping Li, Art Owen, and Cun-Hui Zhang. 2012. One permutation hashing. *Advances in Neural Information Processing Systems* 25 (2012).

[46] Xiang Li, Ben Kao, Zhaochun Ren, and Dawei Yin. 2019. Spectral Clustering in Heterogeneous Information Networks. *Proceedings of the AAAI Conference on Artificial Intelligence* 33, 01 (July 2019), 4221–4228.

[47] Chunxu Lin, Wensheng Luo, Yixiang Fang, Chenhao Ma, Xilin Liu, and Yuchi Ma. 2024. On Efficient Large Sparse Matrix Chain Multiplication. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–27.

[48] Xiao Lin, Min Zhang, Yiqun Liu, and Shaoping Ma. 2019. Enhancing Personalized Recommendation by Implicit Preference Communities Modeling. *ACM Transactions on Information Systems* 37, 4 (Nov. 2019), 48:1–48:32.

[49] Dandan Liu and Zhaonian Zou. 2023. gCore: Exploring Cross-Layer Cohesiveness in Multi-Layer Graphs. *Proc. VLDB Endow.* 16, 11 (July 2023), 3201–3213.

[50] Kaixin Liu, Sibo Wang, Yong Zhang, and Chunxiao Xing. [n.d.]. An Efficient Algorithm for Distance-based Structural Graph Clustering. 1, 1 ([n. d.]).

[51] Lu Liu and Shang Wang. 2020. Meta-Path-Based Outlier Detection in Heterogeneous Information Network. *Frontiers of Computer Science* 14, 2 (April 2020),

388–403.

[52] Qing Liu, Minjun Zhao, Xin Huang, Jianliang Xu, and Yunjun Gao. 2020. Truss-Based Community Search over Large Directed Graphs. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20)*. ACM, 2183–2197.

[53] Linhao Luo, Yixiang Fang, Moli Lu, Xin Cao, Xiaofeng Zhang, and Wenjie Zhang. 2023. GSim: a graph neural network based relevance measure for heterogeneous graphs. *IEEE Transactions on Knowledge and Data Engineering* 35, 12 (2023), 12693–12707.

[54] Wensheng Luo, Xu Zhou, Jianye Yang, Peng Peng, Guoqing Xiao, and Yunjun Gao. 2021. Efficient Approaches to Top-r Influential Community Search. *IEEE Internet of Things Journal* 8, 16 (Aug. 2021), 12650–12657.

[55] Lingkai Meng, Long Yuan, Zi Chen, Xuemin Lin, and Shiyu Yang. 2022. Index-Based Structural Clustering on Directed Graphs. In *ICDE*. IEEE, 2831–2844.

[56] Mark EJ Newman. 2004. Fast algorithm for detecting community structure in networks. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 69, 6 (2004), 066133.

[57] M. E. J. Newman and M. Girvan. 2004. Finding and Evaluating Community Structure in Networks. *Physical Review E* 69, 2 (Feb. 2004), 026113.

[58] Paola Pesantez-Cabrera and Ananth Kalyanaraman. 2019. Efficient Detection of Communities in Biological Bipartite Networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 16, 1 (Jan. 2019), 258–271.

[59] Lianpeng Qiao, Zhiwei Zhang, Ye Yuan, Chen Chen, and Guoren Wang. 2021. Keyword-Centric Community Search over Large Heterogeneous Information Networks. In *Database Systems for Advanced Applications (Lecture Notes in Computer Science)*. Springer International Publishing, 158–173.

[60] Jörg Reichardt and Stefan Bornholdt. 2006. Statistical mechanics of community detection. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 74, 1 (2006), 016110.

[61] Boyu Ruan, Junhao Gan, Hao Wu, and Anthony Wirth. 2021. Dynamic Structural Clustering on Graphs. In *Proceedings of the 2021 International Conference on Management of Data*. ACM, 1491–1503.

[62] Chuan Shi, Xiangnan Kong, Yue Huang, Philip S. Yu, and Bin Wu. 2014. HeteSim: A General Framework for Relevance Measure in Heterogeneous Networks. *IEEE Transactions on Knowledge and Data Engineering* 26, 10 (2014), 2479–2492.

[63] 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*. ACM, 180–191.

[64] Chuan Shi, Yitong Li, Jiawei Zhang, Yizhou Sun, and Philip S. Yu. 2017. A Survey of Heterogeneous Information Network Analysis. *IEEE Transactions on Knowledge and Data Engineering* 29, 1 (Jan. 2017), 17–37.

[65] 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 (CIKM '14)*. ACM, 699–708.

[66] 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 (KDD '12)*. ACM, 1552–1555.

[67] Hiroaki Shiokawa, Yasuhiro Fujiwara, and Makoto Onizuka. 2015. SCAN++: Efficient Algorithm for Finding Clusters, Hubs and Outliers on Large-Scale Graphs. *Proc. VLDB Endow.* 8, 11 (July 2015), 1178–1189.

[68] G Simpson. 1960. Notes on the measurement of faunal resemblance. *Amer J Sci* 258 (1960), 300.

[69] Yizhou Sun, Charu C Aggarwal, and Jiawei Han. 2012. Relation Strength-Aware Clustering of Heterogeneous Information Networks with Incomplete Attributes. *Proc. VLDB Endow.* 5, 5 (2012).

[70] 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. *Proc. VLDB Endow.* 4, 11 (Aug. 2011), 992–1003.

[71] 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 (EDBT '09)*. ACM, 565–576.

[72] 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* 7, 3 (Sept. 2013), 11:1–11:23.

[73] 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 (KDD '09)*. ACM, 797–806.

[74] Tomokatsu Takahashi, Hiroaki Shiokawa, and Hiroyuki Kitagawa. 2017. SCAN-XP: Parallel Structural Graph Clustering Algorithm on Intel Xeon Phi Coprocessors. In *Proceedings of the 2nd International Workshop on Network Data Analytics (NDA'17)*. ACM, 1–7.

[75] A. Topchy, A.K. Jain, and W. Punch. 2005. Clustering Ensembles: Models of Consensus and Weak Partitions. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27, 12 (Dec. 2005), 1866–1881.

[76] Tom Tseng, Laxman Dhulipala, and Julian Shun. 2021. Parallel index-based structural graph clustering and its approximation. In *Proceedings of the 2021 International Conference on Management of Data*. 1851–1864.

[77] Jianwei Wang, Kai Wang, Xuemin Lin, Wenjie Zhang, and Ying Zhang. 2024. Neural Attributed Community Search at Billion Scale. *Proceedings of the ACM on Management of Data* 1, 4 (2024), 1–25.

[78] Jialong Wang, Lihua Zhou, Xiaoxu Wang, Lizhen Wang, and Shijin Li. 2024. Attribute-Sensitive Community Search over Attributed Heterogeneous Information Networks. *Expert Systems with Applications* 235 (Jan. 2024), 121153.

[79] Ruby W. Wang and Fred Y. Ye. 2019. Simplifying Weighted Heterogeneous Networks by Extracting H-Structure via s-Degree. *Scientific Reports* 9, 1 (Dec. 2019), 18819.

[80] Shu Wang, Yixiang Fang, and Wensheng Luo. [n.d.]. Searching and Detecting Structurally Similar Communities in Large Heterogeneous Information Networks (technical report). https://github.com/sam234990/cs-hin-scan.

[81] Yang Wang, D. Chakrabarti, Chenxi Wang, and C. Faloutsos. 2003. Epidemic Spreading in Real Networks: An Eigenvalue Viewpoint. In *22nd International Symposium on Reliable Distributed Systems, 2003. Proceedings.* 25–34.

[82] Yue Wang, Zhe Wang, Ziyuan Zhao, Zijian Li, Xun Jian, Hao Xin, Lei Chen, Jianchun Song, Zhenhong Chen, and Meng Zhao. 2022. Effective Similarity Search on Heterogeneous Networks: A Meta-Path Free Approach. *IEEE Transactions on Knowledge and Data Engineering* 34, 7 (July 2022), 3225–3240.

[83] Dong Wen, Lu Qin, Ying Zhang, Lijun Chang, and Xuemin Lin. 2019. Efficient Structural Graph Clustering: An Index-Based Approach. *The VLDB Journal* 28, 3 (June 2019), 377–399.

[84] Scott White and Padhraic Smyth. 2005. A Spectral Clustering Approach To Finding Communities in Graphs. In *Proceedings of the 2005 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, 274–285.

[85] Xiaowei Xu, Nurcan Yuruk, Zhidan Feng, and Thomas A. J. Schweiger. 2007. SCAN: A Structural Clustering Algorithm for Networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 824–833.

[86] Yixing Yang, Yixiang Fang, Xuemin Lin, and Wenjie Zhang. 2020. Effective and Efficient Truss Computation over Large Heterogeneous Information Networks. In *ICDE*. IEEE, 901–912.

[87] Hao Yin, Austin R. Benson, Jure Leskovec, and David F. Gleich. 2017. Local Higher-Order Graph Clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*. ACM, 555–564.

[88] 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 (RecSys '13)*. ACM, 347–350.

[89] Long Yuan, Lu Qin, Wenjie Zhang, Lijun Chang, and Jianye Yang. 2018. Index-Based Densest Clique Percolation Community Search in Networks. *IEEE Transactions on Knowledge and Data Engineering* 30, 5 (May 2018), 922–935.

[90] Fangyuan Zhang and Sibo Wang. 2022. Effective Indexing for Dynamic Structural Graph Clustering. *Proc. VLDB Endow.* 15, 11 (July 2022), 2908–2920.

[91] Jie Zhang, Jinru Ding, Suyuan Liu, and Hongyan Wu. 2021. META-PATH-FREE REPRESENTATION LEARNING ON HETEROGENEOUS NETWORKS. *arXiv preprint arXiv:2102.08120* (Feb. 2021). arXiv:2102.08120

[92] Mingxi Zhang, Hao Hu, Zhenying He, and Wei Wang. 2015. Top-k Similarity Search in Heterogeneous Information Networks with x-Star Network Schema. *Expert Systems with Applications* 42, 2 (Feb. 2015), 699–712.

[93] 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 (SIGMOD '19)*. ACM, 1024–1041.

[94] Bei Zhao, Yuliang Shi, Kun Zhang, and Zhongmin Yan. 2019. Health Insurance Anomaly Detection Based on Dynamic Heterogeneous Information Network. In *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. 1118–1122.

[95] Huanjing Zhao, Pinde Rui, Jie Chen, Yanping Zhang, Yi Wang, Shu Zhao, and Jie Tang. 2023. HINChip: Heterogeneous Information Network Representation with Community Hierarchy Preserving. *Knowledge-Based Systems* 264 (March 2023), 110343.

[96] Alexander Zhou, Yue Wang, and Lei Chen. 2020. Finding Large Diverse Communities on Networks: The Edge Maximum K*-Partite Clique. *Proc. VLDB Endow.* 13, 12 (July 2020), 2576–2589.

[97] Yingli Zhou, Yixiang Fang, Wensheng Luo, and Yunming Ye. 2023. Influential Community Search over Large Heterogeneous Information Networks. *Proc. VLDB Endow.* 16, 8 (April 2023), 2047–2060.

[98] Yingli Zhou, Yixiang Fang, Chenhao Ma, Tianci Hou, and Xin Huang. 2024. Efficient Maximal Motif-Clique Enumeration over Large Heterogeneous Information Networks. *Proc. VLDB Endow.* 17, 11 (2024), 2946–2959.

[99] Yang Zhou and Ling Liu. 2013. Social Influence Based Clustering of Heterogeneous Information Networks. In *ACM SIGKDD (KDD '13)*. ACM, 338–346.

A PROOF OF LEMMAS

Lemma 1. *Algorithm 1 completes in $O(\alpha n + \beta m)$ time, where α represents the cost of computing the s-neighbors for a vertex, n is the number of vertices within SSC and their s-neighbors, β represents the cost of computing similarity between two vertices, and m represents the total number of s-neighbors of all these vertices.*

Proof. We spend $O(\alpha n)$ time to compute $N_s[v]$ for each vertex v in the SSC and their s-neighbors, and we incur a maximum of m similarity calculations, which costs $O(\beta m)$ time. □

Lemma 2. *Given an HIN, an h-dimensional similarity measure ($h \geq 1$), and an integer μ, SC-Index costs $O(hm)$ space, where m is the number of edges in its similarity graph.*

Proof. Obviously, the space cost of G is $O(hm)$, as each edge retains an h-dimensional similarity vector. Besides, the CTS of each vertex within a $(\mu, \boldsymbol{\epsilon})$-core is determined by the edges it contains, and it is smaller than the number of its neighbors in G, as each edge contributes at most h minimum values to the threshold vectors. Thus, the overall space cost of SC-Index is $O(hm)$. □

Lemma 3. *Given an SC-Index, Algorithm 3 completes in $O(m)$ time, where m is the number of edges between vertices of the SSC in the similarity graph.*

Proof. Since the core determination of each vertex u through CTS involves at most c_u comparisons, where $c_u = |\Phi[u]| \leq |N_G[u]|$, the computation cost for core determination is $O(\sum_{u \in SSC} |c_u|)$, bounded by $O(m)$. Besides, during the expansion, the algorithm needs to perform similarity comparisons for all the edges between vertices of the SSC. Hence, Algorithm 3 completes in $O(m)$ time. □

Lemma 4. *Given an HIN, Algorithm 4 costs $O(\alpha n + \beta m + \gamma^{h-1} m)$ time, where γ is the maximum number of different similarity values among h dimensions.*

Proof. Akin to the online algorithm, the computation of s-neighbors and structural similarities completes in $O(\alpha n + \beta m)$ time. In *CTSMono*, each edge in G is gradually removed, incurring a cost of $O(m)$. Additionally, we iteratively update $\boldsymbol{\epsilon}$ and execute *CTSMono* at most γ^{h-1} times. Hence, the lemma holds. □

Lemma 5. *Algorithm 6 correctly computes all the CTSs for vertices with the target type.*

Proof. During the iterative update of the threshold $\boldsymbol{\epsilon}$, Algorithm 6 obtains $(\mu, \boldsymbol{\epsilon})$-core for all possible similarity values. According to the definitions of CTS and search area, each new threshold added to CTS is always the maximum one during the computation process, guaranteeing the uniqueness and completeness of the computed CTS. Therefore, the lemma holds. □

Lemma 6. *Given an HIN $\mathcal{H}$, Algorithm 6 completes in $O(\alpha n + \beta m + \gamma^e m)$ time, where $e = max\{h - 2, 0\}$.*

Proof. The first two steps are the same as those of Algorithm 4. For the construction of CTS, only a subset of edges within the $(\mu, \boldsymbol{\epsilon})$-core can enable the connected vertex to be included in the anchored vertex set, which costs at most $O(m)$. In addition, each edge in G is removed at most once by employing search area in *CTSDual*, and repeating no more than γ^{h-2} times. □

B REVIEW OF EXISTING OPTIMIZATION OF SIMILARITY COMPUTATION

We provide a brief review of existing optimization methods tailored for some specific similarity measures, which can be used in our solution, to name a few:

- **Jaccard similarity:** For instance, Chang [6] proposed pruning rules and Adaptive Structure-Similar Checking for efficiently computing Jaccard similarity against a threshold. Techniques such as MinHash [5] and k-partition MinHash [45], which are part of Locality Sensitive Hashing (LSH), allow for fast estimation of Jaccard similarity with bounded error [90].
- **Cosine similarity:** Tseng et al. [76] proposed using SimHash to approximate cosine similarity. When using k samples and satisfying $k \geq \pi ln(nm)/(2\delta^2)$, the exact cosine similarity outside the range $(\epsilon - \delta, \epsilon + \delta\sqrt{1-\epsilon^2})$ can be accurately classified as either above or below the threshold ϵ.
- **PathSim similarity:** To compute PathSim between vertices of the same type with a specified meta-path, matrix multiplication can serve as an efficient alternative to path-based searches, as shown by Lin et al. [47].

C ADDITIONAL EXPERIMENTS

• **Similarity of community members.** We compare the community member similarity for communities under different models. After obtaining each community, we use the Jaccard, PathSim, and Cosine similarity metrics to calculate the average similarity between each pair of vertices within the community. Table 5 reports the average of all results, where the results on DBpedia are skipped since CSSH cannot be applied to it. We observe that on all datasets, the similarity of communities obtained by SSCS is significantly higher than that of other methods. Besides, under the SSC model, using the Jaccard method slightly outperforms the PathSim method. Thus, by explicitly considering structural similarity, our SSC model is able to identify communities with higher member similarity.

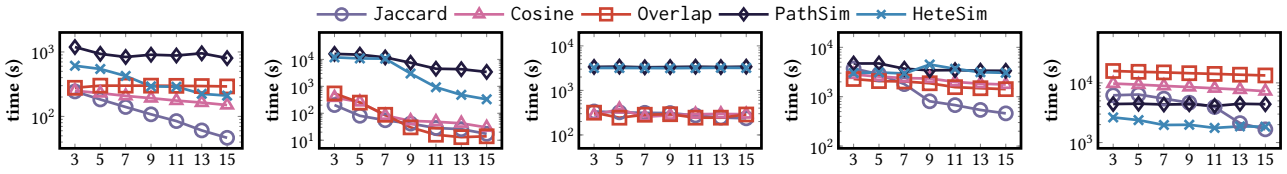
Table 4: Community analysis under CD methods in DBLP.

Algorithm	Similarity	Diameter	CC	Community coverage
SCAN-$\mathcal{P}$	0.572	2.144	0.404	60%
k-Btruss	0.171	2.180	0.524	18%
SSCD	0.652	2.667	0.533	49%

• **Comparing CD solutions.** In this experiment, we compare CD methods (i.e., SSCD, SCAN-$\mathcal{P}$, and k-Btruss), including similarity, diameter, CC, and community coverage, where the community coverage is the ratio of vertices that are assigned to the communities over all the vertices with the target type. As shown in Table 4, our SSCD exhibits higher similarity and achieves higher CC values than other methods. Besides, the communities of SSCD contain fewer vertices. The smaller diameters observed in SCAN-$\mathcal{P}$ are due to the meta-path $\mathcal{P}$="PVP", where papers are segregated into distinct communities solely based on their venues and are fully connected within each community, resulting in a diameter of 1. Although k-Btruss can identify dense subgraphs in HINs and achieve a smaller diameter, the similarity between its members is

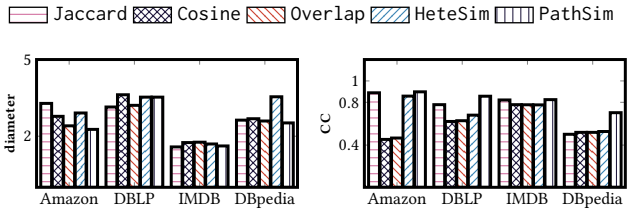
Table 5: Similarity of community members under different CS algorithms. (The best and second best results are marked in bold and underline respectively; We use "–" to denote the cases where the results are unavailable.)

Algorithm	Amazon			DBLP			IMDB			DBpedia		
	Jaccard	PathSim	Cosine	Jaccard	PathSim	Cosine	Jaccard	PathSim	Cosine	Jaccard	PathSim	Cosine
CSH	0.172	0.359	0.222	0.147	0.375	0.229	0.324	0.523	0.389	0.400	0.605	0.435
CSSH	0.169	0.356	0.219	0.148	0.423	0.222	0.398	0.583	0.452	–	–	–
SSCS-Jaccard	**0.348**	**0.362**	**0.43**	0.552	0.545	0.608	**0.898**	**0.806**	**0.929**	**0.957**	**0.862**	**0.971**
SSCS-PathSim	0.345	0.361	0.411	**0.608**	**0.579**	**0.683**	0.876	0.769	0.91	0.838	0.637	0.862
CSH-E	0.073	0.132	0.128	0.05	0.12	0.094	0.226	0.344	0.294	0.284	0.778	0.304
CSH-V	0.06	0.064	0.106	0.048	0.118	0.087	0.276	0.347	0.348	0.285	0.778	0.304



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

Figure 14: SSC search efficiency under different similarity measures by varying $\mu(\epsilon = 0.2)$.



(a) Diameter (b) CC

Figure 15: Diameters and CC values under different similarity measures.

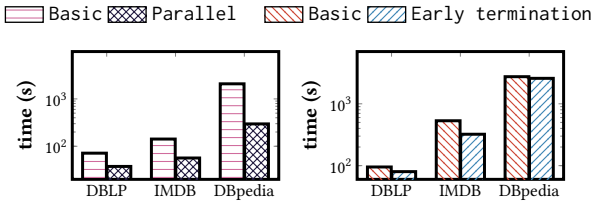
significantly lower than that of other methods. Clearly, the community coverage of k-Btruss is 18%, much lower than those of the other two methods, indicating that 82% of all the vertices are not assigned to any community found by k-Btruss.

• **Similarity measures analysis.** We compare the influence of different similarity measures on SSC search, analyzing their effects on both community quality and search efficiency. Besides Jaccard and PathSim similarity, we select Cosine [76], Szymkiewicz–Simpson coefficient (i.e., Overlap coefficient) [68], and HeteSim [62] as additional similarity measures. Among these, the first two use the same parameter settings as Jaccard, while HeteSim, like PathSim, is based on path-based similarity, using the parameter settings of PathSim.

We compare the quality of communities obtained using SSCS under these similarity measures. As shown in Figure 15, the results from Cosine and overlap coefficient similarities are comparable to those of Jaccard in diameter, but their CC is lower than that of the Jaccard Similarity. This is because the similarity values obtained under these two measures are higher than those of Jaccard, resulting in more vertices being included. However, these additional vertices are not well-connected with other community members, leading to a lower CC value. The results of HeteSim are comparable to those of PathSim. However, in DBPedia, fluctuations occur because certain specified metapaths yield less results.

We also evaluate the runtime efficiency under different similarity measures, as shown in Figure 14. Overall, path-based similarities (PathSim and HeteSim) require more time compared to neighbor-based measures i.e., Jaccard, Cosine, and Overlap coefficient. This is because path-based similarity calculations involve traversing various paths, counting path instances, and computing ratios, which is more time-intensive. In Freebase, the metapaths limits the range of neighbors that need to be searched. While the Overlap coefficient yields higher similarity values between vertices, causing the community to contain a larger number of vertices, which results in longer computation times.

• **optimization for similarity computation.** We also test the optimization of similarity computation using parallel processing and early termination strategies. As shown in Figure 16(a), for the step of similarity computation, the parallelized one is 7× faster than the serial algorithm. As shown in Figure 16(b), by using the early termination strategy in the query, the query efficiency can be improved significantly.



(a) Parallel optimization (b) Early termination

Figure 16: Two different optimization for similarity computation.