HINSCAN: Efficient Structural Graph Clustering over Heterogeneous Information Networks

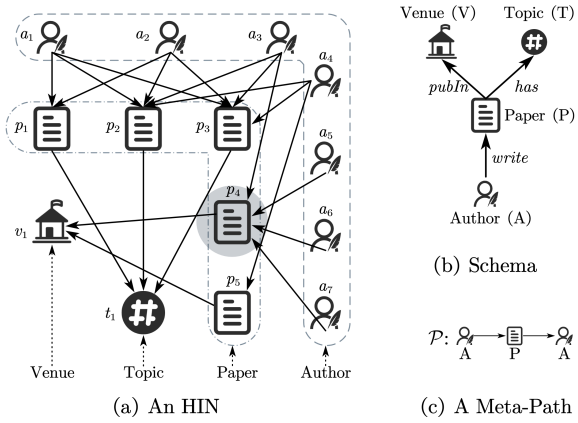
Long Yuan[§], Xiaotong Sun[§], Zi Chen[♮*], Xuemin Lin[‡], Peng Cheng[†], Longbin Lai[◦]
[§]*Nanjing University of Science and Technology, China,* [♮]*Nanjing University of Aeronautics and Astronautics, China,*
[‡]*Shanghai Jiao Tong University, China,* [†]*East China Normal University, China,*[◦]*Alibaba Group, China*
[§]{longyuan,xiaotong}@njust.edu.cn; [♮]zichen@nuaa.edu.cn; [‡]xuemin.lin@gmail.com;
[†]pcheng@sei.ecnu.edu.cn; [◦]longbin.lailb@alibaba-inc.com

***Abstract*—Structural graph clustering (SCAN) is one of the most popular graph clustering paradigms, and has attracted plenty of attention recently. Existing solutions assume that the input graphs is homogeneous, i.e., the vertices are of the same type. However, in many real applications, such as bibliographic networks and knowledge graphs, the input graphs is heterogeneous information networks which consist of multi-typed and interconnected objects, which makes SCAN cannot be applied to cluster. Therefore, in this paper, we study the SCAN problem over heterogeneous information networks. Based on the concept of meta-path, we propose two new structural graph clustering models first. Following these two new models, we design new algorithms to support the efficient clustering of a heterogeneous information network. We conduct extensive experiments on six real heterogeneous information networks, and the results demonstrate the effectiveness of our new models and the efficiency of our proposed clustering algorithms.**

I. Introduction

Graph clustering aims to identify clusters of "similar" vertices that are densely connected within the group and sparsely connected to those outside [1]. This analytical technique finds application across a broad spectrum of fields, encompassing, such as social and biological network analysis [2], load balancing in distributed systems [3], natural language processing [4, 5], and recommendation systems [6]. Structural Clustering Algorithm for Networks (SCAN) is one of the well-known graph clustering methods. Unlike traditional graph clustering methods that enforce non-overlapping cluster boundaries, SCAN allows for the possibility of a vertex being affiliated with multiple clusters. Moreover, it distinguishes between *hub* vertices—those acting as bridges between disparate clusters—and *outlier* vertices, which demonstrate weak ties to any cluster. SCAN has proven effective in clustering both biological data and social media data [5, 7–9].

While the effectiveness of SCAN in detecting hubs, outliers, and clusters has been demonstrated in various applications, SCAN assumes that the input graphs are homogeneous. However, many real-world graphs are heterogeneous information networks (HINs), such as bibliographic networks, social media networks, and knowledge network [10–12], which comprise multiple types of objects and links representing different relationships. Fig. 1 (a) illustrates an HIN derived from DBLP network, delineating the interrelations among various types, including author, paper, venue, and topic. Specifically, it encompasses seven distinct authors labeled a_1 to a_7, five papers labeled p_1 to p_5, one topic labeled t_1, and one venue labeled v_1. The directional connections signify the semantic associations between these entities. For instance, authors a_1 and a_2 co-authored paper p_1, which is centered around topic t_1. Fig. 1 (b) shows the network schema which defines the types of entities (such as authors, papers, venues, and topics) and the allowed relationships between them, providing a structural blueprint for the network [10]. Since existing SCAN methods mainly focus on homogeneous graphs, they are not applicable to HINs. Consequently, in this paper, we study the SCAN problem over HINs and aim to devise a new structural clustering method to cluster the vertices of the same type within a given HIN effectively and efficiently.



(a) An HIN (b) Schema (c) A Meta-Path

Fig. 1. DBLP Bibliographic Network

Challenges. However, designing a new structural clustering model that effectively captures the unique characteristics of HINs while preserving the clustering efficacy of SCAN in identifying diverse vertex roles is challenging. Revisiting the SCAN model, its success is rooted in a rigorous set of definitions that determine the roles of vertices. Given a homogeneous graph G, SCAN deems two vertices connected by an edge structurally similar when the number of their common neighbors normalized by the geometric mean of their degrees (structural similarity) surpasses a predefined threshold ϵ ($0 < \epsilon \leq 1$). A vertex u is categorized as a core vertex if it possesses at least μ ($\mu \geq 2$) structurally similar neighbors.

* Zi Chen is the corresponding author.

Clusters within G originate from core vertices and expand to encompass all vertices connected to cores through structurally similar edges. Any vertex not included in the clusters is classified as a hub vertex if its neighbors belong to two or more clusters; otherwise, it is classified as an outlier vertex. It is evident that the foundational principles of SCAN hinge on the connectivity between nodes and the existence of common neighbors. Yet, in HINs, even the vertices of the same type may not connect directly, let alone having common neighbors and the formulation of more intricate definitions upon that.

Our Approach. Based on the above analysis, the key to devising SCAN over HINs is to precisely depict the connection between two vertices of the same type and the corresponding common neighbors between them. On the other hand, meta-path, defined as a sequence of vertex and edge types based on the network schema of a HIN, represents semantically valid paths between specific node types. Prior research across various graph analytics tasks over HINs, such as similarity search [10], relationship prediction [13], and recommendation [14], demonstrates meta-path not only improves interpretability but is also essential for uncovering complex, multi-relational patterns within HINs. Clearly, meta-path is the backbone for structuring meaningful interactions between vertices within HINs. Inspired by this, in this paper, we introduce the meta-path neighbor (Definition 3.4 in Section III) in which two vertices u and v are considered to be connected if there exists a path instance of the given meta-path between u and v. Following the meta-path neighbor, we propose the non-independent common meta-path neighbor which is a meta-path neighbor of both u and v. Although direct and intuitive, it may lead to the so-called "single-vertex failure" issue. Consider the HIN shown in Fig. 1 (a) and assume the given meta-path is $\mathcal{P} = (APA)$ (Fig. 1 (c)), it is easy to verify that there exist three paths $a_5 \to p_4 \to a_6$, $a_5 \to p_4 \to a_7$, $a_6 \to p_4 \to a_7$ following the pattern of $\mathcal{P}$. Therefore, a_7 is a non-independent common meta-path neighbor of a_5 and a_6. However, revisiting these three edges, we can observe that p_4 appears in all of these three edges. If p_4 disappears, the connection among a_5, a_6, a_7 disappears. To avoid the "single-vertex failure" issue in non-independent common neighbor and further improve the robustness of the model, we propose the independent common neighbor which requires the three path instances connected u, v, and w do not share any more vertex except u, v, and w. Based on meta-path neighbor and non-independent common neighbor/independent common neighbor, we define the corresponding meta-path structural similarity, and core vertex, hub vertex, and outlier vertex accordingly. The case studies on real HINs in our experiment (Exp-4,5,6 in Section VI) demonstrate the effectiveness of our proposed model in identifying different meaningful roles of vertices in HINs such as DBLP, IMDB, and product network. However, as the newly proposed models are more complex than SCAN for homogeneous networks, the following challenges need to be addressed to make our proposed model practically applicable: (1) How to cluster an input HIN for a given meta-path and parameters ϵ and μ following the definition of the new model? (2) Is it possible to further improve the clustering performance by careful algorithm design?

Contribution. In this paper, we answer the above questions and make the following contributions:

- We propose a new structural clustering model named HINSCAN (Two variants: non-independent HINSCAN and independent HINSCAN for different users' requirement) based on meta-path for clustering the HINs. To the best of our knowledge, this is the first work for structural clustering on HINs.
- We propose efficient algorithms to conduct the clustering for the given parameters. For the non-independent HINSCAN, we first propose a transformation-clustering framework via transforming the input HINSCAN to a homogeneous graph named meta-path transformed graph and further improve the performance by introducing a new search-join transformation paradigm. For the independent HINSCAN, we design an efficient algorithm to verify the independence regarding the path instances of the given meta-path for three vertices, and further improve the whole clustering performance by introducing the lower bound and upper bound of the structurally similar neighbors of each vertex to prune unnecessary computation.
- We conduct extensive experiments to evaluate the effectiveness and efficiency of our proposed methods on real HINs. The results demonstrate that HINSCAN is able to obtain meaningful clustering results on HINs and our proposed algorithms can conduct the clustering efficiently for the given parameters.

II. Related Work

Graph data analysis is crucial for uncovering complex relationships and patterns in interconnected data, enabling insights and solutions across a wide range of fields, from social networks to biological systems and beyond [15–24]. Graph clustering, a pivotal concern in graph data analysis, has garnered substantial research interest [1, 25]. Originating with the introduction of the structural graph clustering framework (SCAN) in [26], the field has since witnessed a plethora of advancements. For online algorithms with fixed SCAN parameters, SCAN++ [27], pSCAN [28] and anySCAN [29] accelerate SCAN by eliminating unnecessary structural similarity computation. Among these, pSCAN emerges as the foremost online algorithm for SCAN on homogeneous graphs, demonstrating superior efficiency relative to contemporaries. GPUSCAN [30, 31] harnesses GPU technology to accelerate SCAN execution, while SparkSCAN [32] adapts SCAN for the Spark computing framework. pmSCAN [33] tackles large-scale graphs exceeding main memory constraints, and ppSCAN [34] enhances performance through parallelized pruning algorithms and vectorized instructions. Index-based algorithms, exemplified by GS*-Index [35], introduce space-efficient indexing alongside an index-driven query processing mechanism. Building on this, [36] offers a parallelized GS*-Index and an approximate construction algorithm utilizing

locality-sensitive hashing. [37] focuses on maintaining structural similarity amidst graph updates, proposing an approximate algorithm with theoretical guarantees. [38] contributes a novel structural clustering paradigm for directed graphs, coupled with an index-driven strategy for efficient clustering. However, all preceding methods pertain exclusively to homogeneous graphs and the corresponding techniques cannot be directly extended to support HINs. Therefore, in this paper, we propose a structural clustering model for HINs and design efficient algorithms to conduct the clustering following the new model. Note that [12] uses edge-joint meta-path neighbors to define the truss model on HINs, which inspires the definition of our independent common meta-path neighbors. Despite the existence of alternative graph clustering methods for HINs, as seen in [39–42], none are grounded in structural clustering, thereby lacking the capability of SCAN to effectively identify clusters, hubs, and outliers. A comprehensive survey on this topic can be found in [43].

III. Preliminaries

Let $G = (V, E)$ be a graph, where $V(G)$ is the set of vertices and $E(G)$ is the set of edges, respectively. For a vertex $v \in V(G)$, we use $\mathsf{nbr}(v, G)$ to denote the set of neighbors of v, i.e., $\mathsf{nbr}(v, G) = \{u | (u, v) \in E(G)\}$. The degree of a vertex $v \in V(G)$, denoted by $\mathsf{deg}(v, G)$ is the number of neighbors of v, i.e., $\mathsf{deg}(v, G) = |\mathsf{nbr}(v, G)|$. For simplicity, we omit G in the notations when the context is self-evident.

Definition 3.1: (Heterogeneous Information Network [10]) An heterogeneous information networks (HIN) is a directed graph $\mathcal{G} = (V, E, \psi, \phi, \mathcal{A}, \mathcal{R})$ where: (1) V is the set of vertices. (2) E is the set of edges. (3) $\psi : V \to \mathcal{A}$ is a vertex type mapping function that maps each vertex $v \in V$ to a vertex type $\psi(v) \in \mathcal{A}$. (4) $\phi : E \to \mathcal{R}$ is an edge type mapping function that maps each edge $e \in E$ to an edge type $\phi(e) \in \mathcal{R}$. Here, $\mathcal{A}$ is the set of vertex types, and $\mathcal{R}$ is the set of edge types.

Accordingly, if a graph G has vertices and edges of the same type, we call it a homogeneous graph. For the ease of presentation, we use G to denote a homogeneous graph and $\mathcal{G}$ to denote an HIN thereafter when the context is self-evident.

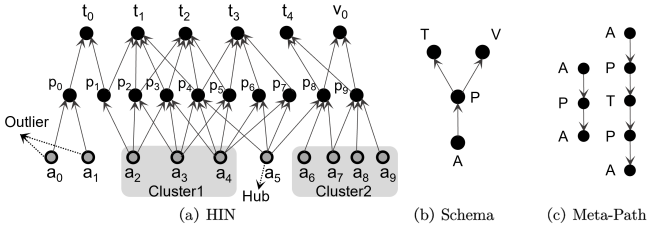
Definition 3.2: (HIN Schema [10]) Given an HIN $\mathcal{G} = (V, E)$ with mappings $\psi : V \to \mathcal{A}$ and $\phi : E \to \mathcal{R}$, its schema $T_{\mathcal{G}}$ is a directed graph defined over vertex types $\mathcal{A}$ and edge types from $\mathcal{R}$, i.e., $T_{\mathcal{G}} = (\mathcal{A}, \mathcal{R})$.

The HIN schema describes all permissible edge types between vertex types. If there is an edge R from vertex type A_1 to vertex type A_2, the inverse edge R^{-1} naturally exists from A_2 to A_1. We use lowercase letters (e.g. a_1) to denote vertices, and uppercase letters (e.g., A_1) to denote vertex types.

Definition 3.3: (Meta-path [10]) A meta-path $\mathcal{P}$ is a path defined on an HIN schema $T_{\mathcal{G}} = (\mathcal{A}, \mathcal{R})$, and is represented as $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \cdots \xrightarrow{R_l} A_{l+1}$, where l is the length of $\mathcal{P}$, $A_i \in \mathcal{A}$, and $R_i \in \mathcal{R} (1 \le i \le l)$.

For simplicity, we also use vertex type names to denote a meta-path, i.e., $\mathcal{P} = (A_1 A_2 \cdots A_{l+1})$, if there are no multiple edges between the same pair of vertex types. We also use $|\mathcal{P}|$ to denote the length of $\mathcal{P}$, $\mathcal{P}[i]$ to denote i-th vertex type of $\mathcal{P}$, where $0 \le i < |\mathcal{P}|$, $\mathcal{P}[i \dots j]$ to denote the sub-path of $\mathcal{P}$ from position i to j, where $0 \le i < j < |\mathcal{P}|$, i.e., $\mathcal{P}[i \dots j] = (A_i A_{i+1} \cdots A_j)$. A path $p = a_1 \to a_2 \cdots \to a_{l+1}$ between vertices a_1 and a_{l+1} is called a path instance of $\mathcal{P}$, if for all i, the vertex a_i and edge $e_i = (a_i, a_{i+1})$ satisfy $\psi(a_i) = A_i$ and $\phi(e_i) = R_i$. For example, in Fig. 2, assume the given meta-path $\mathcal{P}_1 = (APA)$, the path $a_0 \to p_0 \to a_1$ is a path instance of $\mathcal{P}_1$. Since the meta-path provides the meta-level explainable semantics of HINs [10] and has been successfully used in various graph analytics tasks over HINs such as similarity search [10], relationship prediction [13], and recommendation [14].

Definition 3.4: (Meta-path Neighbor) Give a meta-path $\mathcal{P}$ on an HIN $\mathcal{G}$, for a vertex $u \in V(\mathcal{G})$, a vertex $v \in V(\mathcal{G})$ is a meta-path neighbor of u regarding $\mathcal{P}$ if there exists a path instance of $\mathcal{P}$ between u and v.



(a) HIN (b) Schema (c) Meta-Path

Fig. 2. A HIN $\mathcal{G}$ with Schema and Meta-path ($\epsilon = 0.80, \mu = 3$)

Definition 3.5: (Meta-path Structural Neighborhood) Given a meta-path $\mathcal{P}$ on a HIN $\mathcal{G}$, for a vertex $u \in V(\mathcal{G})$, the structural neighborhood of u, denoted by $N_{\mathcal{P}}(u)$, is defined as $N_{\mathcal{P}}(u) = \{v \in V(\mathcal{G}) | v \text{ is a meta-path neighbor of } u\} \cup \{u\}$.

Definition 3.6: (Meta-path Structural Similarity) Given a meta-path $\mathcal{P}$ on a HIN $\mathcal{G}$, for two vertices $u, v \in V(\mathcal{G})$, the meta-path structural similarity between u and v is defined as:

$$\sigma_{\mathcal{P}}(u, v) = \frac{|N_{\mathcal{P}}(u) \cap N_{\mathcal{P}}(v)|}{\sqrt{|N_{\mathcal{P}}(u)||N_{\mathcal{P}}(v)|}} \tag{1}$$

Example 3.1: Consider the HIN $\mathcal{G}$ 1 shown in Fig. 2. Assume the given meta-path $\mathcal{P}_2 = (APTPA)$. For a_0, a_1 is a meta-path neighbor of a_0 as there exists a path $a_0 \to p_0 \to t_0 \to p_0 \to a_1$ between a_0 and a_1, and a_2 is also a meta-path neighbor of a_0 due to $a_0 \to p_0 \to t_0 \to p_1 \to a_2$, and thus $N_{\mathcal{P}}(a_0) = \{a_1, a_2\}$. Similarly, $N_{\mathcal{P}}(a_2) = \{a_0, a_1, a_2, a_3, a_4, a_5\}$, $N_{\mathcal{P}}(a_3) = \{a_2, a_3, a_4, a_5\}$. The meta-path similarity between a_2 and a_3 is $\sigma_{\mathcal{P}}(a_2, a_3) = |\{a_2, a_3, a_4, a_5\}|/\sqrt{6 \times 4} \approx 0.82$.

Given a meta-path structural similarity threshold ϵ ($0 < \epsilon \le 1$), the ϵ meta-path neighborhood for a vertex u is defined as follows:

Definition 3.7: (ϵ Meta-path Neighborhood) Given a meta-path $\mathcal{P}$ on a HIN $\mathcal{G}$ and a meta-path structural similarity threshold ϵ, for a vertex $u \in V(\mathcal{G})$, the ϵ meta-path neighborhood of u, denoted by $N_\epsilon[u]$, is defined as the subset of

$N_{\mathcal{P}}[u]$ in which every vertex v satisfies $\sigma_{\mathcal{P}}(u,v) \geq \epsilon$, i.e., $N_{\mathcal{P},\epsilon}[u]$=$\{v \in N_{\mathcal{P}}[u] \mid \sigma_{\mathcal{P}}(u,v) \geq \epsilon\}$.

When the number of ϵ meta path neighbors of a vertex is large enough, it becomes a core vertex, which is defined as:

Definition 3.8: (Core) Given a meta-path $\mathcal{P}$ on an HIN $\mathcal{G}$, a meta-path structural similarity threshold ϵ ($0 < \epsilon_c \leq 1$), and an integer μ ($\mu \geq 2$), a vertex u is a core if $|N_{\mathcal{P},\epsilon}[u]| \geq \mu$.

Given a core vertex u, the structurally reachable vertices of u is defined as:

Definition 3.9: (Structurally Reachable) Given two vertices u and v in $\mathcal{G}$, v is structurally reachable from u if there exists a sequence of vertices v_1, v_2, ..., $v_l \in V(\mathcal{G})$ ($l \geq 2$) such that: (1) $v_1 = u$, $v_l = v$; (2) $v_1, v_2, \ldots, v_{l-1}$ are core vertices; and (3) $v_{i+1} \in N_{\mathcal{P},\epsilon}[v_i]$ for each $1 \leq i \leq l-1$.

Definition 3.10: (Cluster) A cluster $C \in V(\mathcal{G})$ is a non-empty subset of $V(G)$ such that:

- (Connectivity) For any two vertices v_1, $v_2 \in C$, there exists 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 structure reachable from u also belong to C.

For ease of presentation, we refer to a vertex in a cluster that is not a core vertex as a non-core vertex. After identifying all the clusters $\mathbb{C}$ in $\mathcal{G}$, we can determine the set of hubs and outliers in $\mathcal{G}$, which are defined as follows:

Definition 3.11: (Hub and Outlier) Given the set of clusters in an HIN $\mathcal{G}$, a vertex u that is not in any cluster is a hub vertex if its meta-path neighbors belong to two or more clusters. Otherwise it is an outlier vertex.

Example 3.2: Reconsider $\mathcal{G}$ in Fig. 2, it is easy to verify a_2, a_7 are core vertices as $|N_{\mathcal{P},\epsilon}(a_2)| = 3 \geq 3$ and $|N_{\mathcal{P},\epsilon}(a_7)| = 4 \geq 3$. Following the core vertex, we can find two clusters: $C_1 = \{a_2, a_3, a_4\}$ and $C_2 = \{a_6, a_7, a_8, a_9\}$, a_5 is a hub vertex as it is not in any cluster but its meta-path neighbors, such as a_4 and a_6, belong to two clusters C_1 and C_2. a_0 and a_1 are outlier vertices as they are not belong to any cluster.

Following Definition 3.6, the meta-path structural similarity between two vertices u and v is determined by their common meta-path neighbors w. Although this definition is direct and intuitive, it may include vertices that are weakly engaged in the cluster due to the path instances from u and v to their common meta-path neighbors w may share the same vertices and causes the "single-vertex failure" issue as discussed in Section I. To address this issue, we define the *independent* HINSCAN model based on independent common meta-path neighbors:

Definition 3.12: (Independent Common Meta-path Neighbors) Given a meta-path $\mathcal{P}$ on an HIN $\mathcal{G}$, for two vertices $u, v \in V(\mathcal{G})$, the independent common meta-path neighbors of u and v, denoted by $\mathcal{I}_{\mathcal{P}}(v,w)$, is the set of vertices $w \in N_{\mathcal{P}}(u) \cap N_{\mathcal{P}}(v)$ such that there exist three independent path instances p_1, p_2, p_3 of $\mathcal{P}$ connecting (u,v), (u,w) and (v,w) and p_1, p_2, p_3 do not share any vertex except u, v, and w.

Definition 3.13: (Independent Meta-path Structural Similarity) Given a meta-path $\mathcal{P}$ on an HIN $\mathcal{G}$, for two vertices $u, v \in V(\mathcal{G})$, the independent meta-path structural similarity between u and v is defined as:

$$\sigma^{\mathcal{I}}_{\mathcal{P}}(u,v) = \frac{|\mathcal{I}_{\mathcal{P}}(u,v)|}{\sqrt{|N_{\mathcal{P}}(u)||N_{\mathcal{P}}(v)|}} \tag{2}$$

Example 3.3: Consider the vertex a_0 and a_1 in $\mathcal{G}$ shown in Fig. 2. We have $N_{\mathcal{P}}(a_0) \cap N_{\mathcal{P}}(a_1) = \{a_0, a_1, a_2\}$. However, we can't find three independent path instances connecting $(a_0, a_1), (a_0, a_2)$ and (a_1, a_2), because they must be connected through t_0. For this reason, $\mathcal{I}_{\mathcal{P}}(a_0, a_1) = \{a_0, a_1\}$, and $\sigma^{\mathcal{I}}_{\mathcal{P}}(a_0, a_1) = |\{a_0, a_1\}|/\sqrt{3 \times 3} \approx 0.67$, which is different from $\sigma_{\mathcal{P}}(a_0, a_1) = |\{a_0, a_1, a_2\}|/\sqrt{3 \times 3} = 1$ if meta-path structural similarity is used.

Clearly, the "single-vertex failure" issue is avoided in Definition 3.13. Following Definition 3.13, we can further define the counterparts of ϵ meta-path neighborhood, core, cluster, hub, and outlier in the similar way as Definition 3.7 $\sim$ Definition 3.11. For the sake of differentiation, we refer to the clustering model based on meta-path structural similarity as the *non-independent* HINSCAN model.

Problem statement. Given a HIN $\mathcal{G}$, a meta-path $\mathcal{P}$, two parameters ϵ ($0 < \epsilon \leq 1$), and μ ($\mu \geq 2$), in this paper, we aim to efficiently compute all clusters in $\mathcal{G}$ and identify the corresponding role of each vertex following the non-independent HINSCAN model and the independent HINSCAN model. As we aim to cluster the vertices in $\mathcal{G}$ with the same type, the first and last vertices of $\mathcal{P}$ must be of the same type.

IV. Non-independent HINSCAN Algorithms

A. A Transformation-Clustering Framework

To conduct the clustering, a direct approach involves first computing the meta-path structural similarity between every pair of vertices and then identifying the role of each vertex according to Definition 3.8, Definition 3.10, and Definition 3.11. However, this approach entails significant unnecessary computation, as shown by the results of the state-of-the-art SCAN algorithm for homogeneous graphs, pSCAN [44]. By avoiding unnecessary computations, pSCAN achieves significant speedup on performance and has demonstrated success across numerous applications. This motivates us to explore whether, given the impressive performance of pSCAN, we can design a new algorithm for non-independent HINSCAN that fully exploits the techniques proposed in pSCAN. Following this line of thought, we define:

Definition 4.1: (Meta-path Transformed Graph) Given a meta-path $\mathcal{P}$ on an HIN $\mathcal{G}$, the meta-path transformed graph $G_{\mathcal{P}} = (V, E)$ is a homogeneous graph consisting of the vertices with the target clustering vertex type and two vertices u and v have an edge if there is a path instance of $\mathcal{P}$ between u and v in $\mathcal{G}$.

For the ease of presentation, given a vertex $u \in G_{\mathcal{P}}$, we use $N(u)$ to denote $\mathsf{nbr}(u, G_{\mathcal{P}}) \cup \{u\}$, and use $\sigma(u,v)$ to denote

the structural similarity [44] between u and v in $G_{\mathcal{P}}$, i.e., $\sigma(u,v) = \frac{|N[u] \cap N[v]|}{\sqrt{|N[u] \cup N[v]|}}$. We have:

Lemma 4.1: *Given a meta-path $\mathcal{P}$ on an* HIN *$\mathcal{G}$, let $G_{\mathcal{P}}$ be the meta-path transformed graph G', for two vertices $u, v \in V(G')$, $\sigma_{\mathcal{P}}(u,v) = \sigma(u,v)$.*

Meanwhile, it can be easily verified that for a vertex $u \in V(\mathcal{G})$, if $|N_{\mathcal{P},\epsilon}[u]| \geq \mu$, then, the number of vertices $v \in N(u)$ with $\sigma(u,v) \geq \epsilon$ in $G_{\mathcal{P}}$ is also not less than μ. This implies that a core vertex in $\mathcal{G}$ is also a core vertex in $G_{\mathcal{P}}$. Similar conclusions can be drawn regarding non-core vertices, hub vertices, and outlier vertices. Therefore, we can identify the roles of vertices in $\mathcal{G}$ based on $G_{\mathcal{P}}$. Following this idea, our two-phase non-independent HINSCAN algorithm works as follows: first, we transform $\mathcal{G}$ into $G_{\mathcal{P}}$. Then, we perform clustering on $G_{\mathcal{P}}$ using pSCAN, and the identified roles regarding the vertices in $G_{\mathcal{P}}$ correspond exactly to what we need. The pseudocode of our algorithm is shown in Algorithm 1.

Algorithm 1: NIHINScan$(\mathcal{G}, \mathcal{P}, \epsilon, \mu)$

```
1  𝕍 ← {u ∈ V(𝒢) | ψ(u) = 𝒫[0]}; G_𝒫 ← ∅;
2  for each vertex u ∈ 𝕍 do
3  |  𝒮 ← ∅, ℱ[·] ← ∅;
4  |  findMPN(𝒢, 𝒫, 0, u, 𝒮, ℱ[·]);
5  |  for each v ∈ 𝒮 do
6  |  |  if (u, v) ∉ G_𝒫 then add edge (u, v) into G_𝒫;
7  pSCAN(G_𝒫, ϵ, μ) [44];
8  Procedure findMPN(𝒢, 𝒫, i, u, 𝒮, ℱ[·])
9  for each v ∈ nbr(u, 𝒢) do
10 |  if 𝒫[i] = ψ(v) and v ∉ ℱ[i] then
11 |  |  if i + 1 < |𝒫| then
12 |  |  |  findMPN(𝒢, 𝒫, i + 1, v, 𝒮, ℱ[·]); ℱ[i].add(v);
13 |  |  else
14 |  |  |  𝒮.add(v); ℱ[i].add(v);
```

Algorithm. Algorithm 1 begins by constructing the meta-path transformed graph $G_{\mathcal{P}}$ (lines 1-6) and then directly invokes pSCAN [44] to identify the roles of the vertices in $G_{\mathcal{P}}$ (line 7). Specifically, it first retrieves the vertices aiming to cluster with type $\mathcal{P}[0]$ and stores them in $\mathbb{V}$ (line 1). Obviously, the vertices in $\mathbb{V}$ constitute the vertices of $G_{\mathcal{P}}$. Then, it iterates over each vertex $u \in \mathbb{V}$ (line 2), obtains the meta-path neighbors of u through the findMPN and stores them in $\mathcal{S}$ (line 4). After that, it adds edge between u and its meta-path neighbors v in $G_{\mathcal{P}}$ (line 5-6). When the meta-path transformed graph $G_{\mathcal{P}}$ is built, pSCAN is invoked to identify the roles of the vertices in $G_{\mathcal{P}}$. For the procedure findMPN, staring from u (line 4), it recursively visits i-th vertex type of $\mathcal{P}$ (line 9) and selects the neighbors v of u with $\mathcal{P}[i] = \psi(v)$ (line 10). $\mathcal{F}[i]$ is used to store the visited vertices with type $\mathcal{P}[i]$ during exploration. When all the $|\mathcal{P}|$ vertex types are explored, the meta-path neighbors of u are stored in $\mathcal{S}$ (line 14).

Example 4.1: Consider the HIN $\mathcal{G}$ in Fig. 2, Fig. 3 shows its corresponding meta-path transformed graph $G_{\mathcal{P}}$, where $\mathcal{P} = (APTPA)$. The vertices in $\mathcal{G}$ with vertex type A constitute the vertices of $G_{\mathcal{P}}$, and the edges can be obtained by procedure findMPN. Take a_0 as an example, we get the vertex of type P among the neighbors of a_0, which is p_0. Then, we explore p_0 to get its neighbors of type T, which is t_0. After that, we explore t_0 and get p_0, p_1. Finally, we explore p_0, p_1 in turn and get a_1, a_2. Then, the edges (a_0, a_1) and (a_0, a_2) are added. For the given parameters $\epsilon = 0.8, \mu = 3$, the clustering result based on pSCAN is also illustrated. It is easy to verify that the result is consistent with the results shown in Fig. 2.

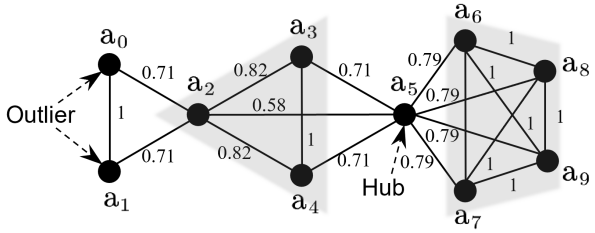


Fig. 3. Meta-path Transformed Graph $G_{\mathcal{P}}$ and Clustering Result of pSCAN

The correctness of Algorithm 1 can be directly proved following Lemma 4.1. For the time complexity of Algorithm 1, we have:

Theorem 4.1: *Given a meta-path $\mathcal{P}$ on an* HIN *$\mathcal{G}$, two parameter ϵ and μ, the time complexity of Algorithm 1 to conduct the clustering is $O(|V(G_{\mathcal{P}})| \cdot \mathsf{deg}_{\max}^{|\mathcal{P}|} + |E(G_{\mathcal{P}})|^{1.5})$, where $|V(G_{\mathcal{P}})|/|E(G_{\mathcal{P}})|$ denotes the number of vertices/edges in $G_{\mathcal{P}}$ respectively, $\mathsf{deg}_{\max}$ denotes the maximum degree of vertices in $\mathcal{G}$.*

B. A Search-Join Transformation Algorithm

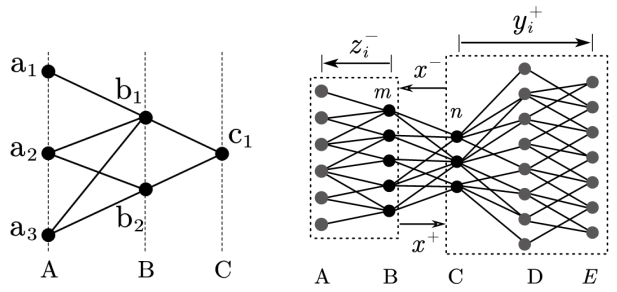
By introducing the meta-path transformed graph, Algorithm 1 can fully leverage the optimization of pSCAN. However, transforming the input graph $\mathcal{G}$ into $G_{\mathcal{P}}$ is time-consuming, as demonstrated in Theorem 4.1. We also evaluate the time required for these two phases on five real datasets used in our experiments, and the results are presented in Table I. As shown in Table I, the time used for transforming $\mathcal{G}$ into $G_{\mathcal{P}}$ is times significantly longer than the time used for clustering on average. On dataset Foursquare, the transformation phase takes even two orders of magnitude more time than the clustering phase. Therefore, to accelerate the performance of non-independent HINSCAN, it is crucial to improve the efficiency of constructing the meta-path transformed graph.

TABLE I
TIME CONSUMED AT EACH PHASE OF ALGORITHM 1 (*us*)

Phase	TMDB	IMDB	DBLP	Foursquare	DBpedia
Transform	85,986	415,418	4,826,255	3,024,440	3,620,805
Clustering	17,689	2705	288,636	10,451	339,808

A Search-Join Transformation Paradigm. Revisit Algorithm 1, findMPN obtains the vertex pairs connecting by a path instance of the given meta-path $\mathcal{P}$ by exploring the vertices in $\mathcal{G}$ directly following the direction of $\mathcal{P}$. However, this approach may involve lots of redundant computation. Consider a HIN with meta-path $\mathcal{P} = (ABC)$ shown in Fig. 4 (a). If we follow the meta-path direction directly, $A \rightarrow B \rightarrow C$, we need to gather all vertices of type A (e.g., $\{a_1, a_2, a_3\}$)

and then perform the same search operation on them one by one. For instance, starting from a_1, we first find all its neighbors of type B, and then for each vertex of type B, we find all its neighbors of type C. This process involves exploring vertices $\{a_1, b_1, c_1\}$. Similarly, for a_2, we explore $\{a_2, b_1, b_2, c_1\}$, and for a_3, we explore $\{a_3, b_1, b_2, c_1\}$. It's evident that b_1 and b_2 and their corresponding neighbors are explored multiple times, causing additional overhead. Alternatively, we can get such vertex pairs by dividing $\mathcal{P} = (ABC)$ into two sub-meta-path $\mathcal{P}_1 = (BA)$ and $\mathcal{P}_2 = (BC)$ first. Then, we search the neighbors of vertices with vertex type B following the direction of $B \to A$ and $B \to C$, and join the neighbors together to obtain the final vertex pairs. This requires us to explore the vertices of type B along two sub-meta-paths, involving vertices $\{b_1, a_1, a_2, a_3, c_1\}$ and $\{b_2, a_2, a_3, c_1\}$. Moreover, we can search the vertex pairs in the reverse order of $\mathcal{P}$, $C \to B \to A$. Similar to the above, this involves exploring vertices $\{c_1, b_1, b_2, a_1, a_2, a_3\}$. From this example, we can observe that different search methods could affect the number of explored vertices, and thus the performance of the meta-path transformed graph construction. This motivates us to investigate whether we can find a search method that minimizes vertex exploration as much as possible to improve the performance of meta-path transformed graph construction.



(a) HIN ($\mathcal{P} = ABC$) (b) A Conceptual Example

Fig. 4. A Search-Join Transformation Paradigm

Generally, in Fig. 4 (b), A, B, C, D, E represent five types of vertices, respectively. Assume that there are m vertices of type B and n vertices of type C. Let x^-/x^+ denote the total number of vertices that need to be explored from $C \to B$ / $B \to C$. Additionally, y_i^+ represents the number of vertices that need to be explored from each vertex i in C to its (C, D, E)-path-neighbors in E. z_i^- represents the number of vertices that need to be visited from each vertex i in B to its (B, A)-path-neighbors in A. Assuming that the average time complexity of visiting a vertex is $O(\alpha)$, the time complexity of building the meta-path transformed graph by $C \to B \to A$ and $C \to D \to E$ is: $t_1 = O(\alpha \cdot x^- + \sum_{i=1}^{n} \alpha \cdot y_i^+ + \sum_{i=1}^{m} \alpha \cdot \beta \cdot z_i^-)$, where $0 \le \beta \le n$ denotes that each vertex in B may be connected to β vertices in C, And the time complexity of building the meta-path transformed graph by $B \to A$, $B \to C \to D \to E$ is: $t_2 = O(\alpha \cdot x^+ + \sum_{i=1}^{n} \alpha \cdot \gamma \cdot y_i^+ + \sum_{i=1}^{m} \alpha \cdot z_i^-)$, where $0 \le \gamma \le m$ denotes that each vertex in C may be connected to γ vertices in B. It is easy to verify that $x^+ \gtrapprox x^-$, and if $m \gg n$, we have $\sum_{i=1}^{n} \gamma \cdot y_i^+ \gg \sum_{i=1}^{m} \beta \cdot z_i^-$, which means $t_1 > t_2$. Therefore, starting the search with vertex types that have fewer vertices tends to be more efficient. Based on this observation, we define:

Definition 4.2: (Pivot Type) Given a meta-path $\mathcal{P}$ on an HIN $\mathcal{G}$, the pivot type $\psi_{\mathcal{P}}$ of $\mathcal{P}$ regarding $\mathcal{G}$ is the vertex type in $\mathcal{P}$ such that the number of vertices in $\mathcal{G}$ belonging to that type is minimum.

Algorithm 2: NIHINScan$^+(\mathcal{G}, \mathcal{P}, \epsilon, \mu)$

```
1  ψ_P, idx ← getPivotType(G, P);
2  V ← {u ∈ V(G) | ψ(u) = ψ_P}; G_P ← ∅;
3  P_{f+} ← P[idx ... |P|]; P_{f-} ← P[0 ... |idx|];
4  for each u ∈ V do
5  |  M ← ∅; S_{f+} ← ∅, S_{f-} ← ∅, F_{f+}[·] ← ∅, F_{f-}[·] ← ∅;
6  |  findMPN(G, P_{f+}, 0, u, S_{f+}, F_{f-}[·]);
7  |  findMPN(G, P_{f-}, 0, u, S_{f-}, F_{f-}[·]);
8  |  for each v ∈ S_{f+} do
9  |  |  for each w ∈ S_{f-} do
10 |  |  |  M ← M ∪ {(v, w)};
11 |  |  for each (v, w) ∈ M do
12 |  |  |  add edge (v, w) into G_P;
13 pSCAN(G_P, ε, μ) [44];
14 Procedure getPivotType(G, P)
15 i ← 0; cnt ← ∞; ψ ← ∅; idx ← 0
16 while i < |P| do
17 |  if cnt > |{u ∈ V(G) | ψ(u) = P}| then
18 |  |  cnt ← |{u ∈ V(G) | ψ(u) = P}|;
19 |  |  ψ ← P[i]; idx ← i
20 |  i++;
21 return ψ, idx;
```

Algorithm. According to Definition 4.2, our new search-join transformation based non-independent HINSCAN algorithm is shown in Algorithm 2. Different from Algorithm 1, it first retrieves the pivot type $\psi_{\mathcal{P}}$ of $\mathcal{P}$ based on Definition 4.2 (line 1) and computes the vertices in $\mathcal{G}$ with type $\psi_{\mathcal{P}}$ (line 2). Moreover, based on the pivot type $\psi_{\mathcal{P}}$, the meta-path $\mathcal{P}$ is divided into two sub-meta-path $\mathcal{P}_{f^+}$ and $\mathcal{P}_{f^-}$ (line 3). Then, it iterates vertices in $\mathbb{V}$ (line 4). For each vertex u, two searches following $\mathcal{P}_{f^+}$ and $\mathcal{P}_{f^-}$ are conducted and the meta-path neighbors of u regarding $\mathcal{P}_{f^+}$/$\mathcal{P}_{f^-}$ are stored in $\mathcal{S}_{f^+}$/$\mathcal{S}_{f^-}$ (line 6-7). After that, the meta-path neighbors of u stored in $\mathcal{S}_{f^+}$ and $\mathcal{S}_{f^-}$ are visited and the corresponding edge (v, w) is added into $G_{\mathcal{P}}$ (line 8-12). When $G_{\mathcal{P}}$ is built, pSCAN is invoked to identify the roles of the vertices (line 13). Procedure getPivotType is used to find the pivot type of $\mathcal{P}$ regarding $\mathcal{G}$ following Definition 4.2. As the pseudocode is self-evident, we omit the explanation.

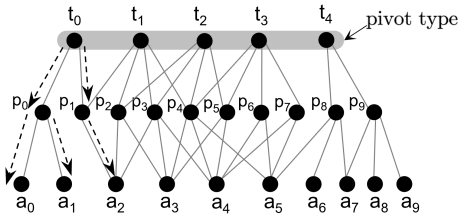


Fig. 5. Example of pivot type

Example 4.2: Consider the HIN $\mathcal{G}$ in Fig. 2, Fig. 5 shows the pivot type we choose to build the transformed graph $G_{\mathcal{P}}$, which is type T. Type T divides the meta-path into two sub-meta-paths, i.e. $\mathcal{P}_1 = (APT), \mathcal{P}_2 = (TPA)$. For each sub-meta-path, we traverse each type along the path start from T, and then join the obtained vertex set of type A. In the end, the edges in $G_{\mathcal{P}}$ is obtained. Take t_0 as an example, for the sub-meta-path $\mathcal{P}_1$, we first get the vertex of type P among the neighbors of t_0, which is p_0, p_1. Then, we explore p_0, p_1 in turn and get the set of type A, which is $\{a_0, a_1, a_2\}$. Similarly, for sub-meta-path $\mathcal{P}_2$, we can also get the set $\{a_0, a_1, a_2\}$. Then, we join these two set, and finally we have an edges (a_0, a_1), (a_0, a_2), and (a_1, a_2).

V. Independent HINSCAN Algorithms

A. A Verification-Based Approach

Following Definition 3.13, it is clear that we can still conduct the clustering based on the meta-path transformed graph. However, as the independent meta-path structural similarity is defined based on independent common meta-path neighbors, and while the independence among the path instances is not considered when constructing the meta-path transformed graph. Therefore, we need to verify the independence when retrieving the independent common meta-path neighbors, and compute the independent meta-path structural similarity accordingly. Following this idea, our baseline algorithm for independent HINSCAN is shown in Algorithm 3.

Algorithm 3: IHINScan$(\mathcal{G}, \mathcal{P}, \epsilon, \mu)$

1 construct $G_{\mathcal{P}}$ based on $\mathcal{G}$ (line 1-12 of Algorithm 2);
2 **for *each*** $u \in V(G_{\mathcal{P}})$ **do**
3 **for *each*** $v \in \mathsf{nbr}(u, G_{\mathcal{P}})$ **do**
4 $\mathcal{I}_{\mathcal{P}}(u, v) \leftarrow \mathsf{findIMPN}(N[u], N[v])$;
5 $\sigma^{\mathcal{I}}_{\mathcal{P}}(u, v) \leftarrow |\mathcal{I}_{\mathcal{P}}(u, v)| / \sqrt{|N[u]| \cdot |N[v]|}$;
6 **if** $\sigma^{\mathcal{I}}_{\mathcal{P}}(u, v) \geq \epsilon$ **then**
7 $\mathcal{N}_{\mathcal{P},\epsilon}[v].\mathsf{add}(u)$; $\mathcal{N}_{\mathcal{P},\epsilon}[u].\mathsf{add}(v)$;
8 $\mathbb{C} \leftarrow \emptyset$; $V' \leftarrow V(G_{\mathcal{P}})$; $C \leftarrow \emptyset$;
9 **for *each*** $u \in V(G_{\mathcal{P}})$ **do**
10 **if** u is explored **then continue**;
11 **if** $|\mathcal{N}_{\mathcal{P},\epsilon}[u]| \geq \mu$ **then**
12 $Q \leftarrow \emptyset$; $Q.\mathsf{push}(u)$; mark u as explored; $C \leftarrow C \cup \{u\}$; $V' \leftarrow V' \setminus \{u\}$
13 **while** $Q \neq \emptyset$ **do**
14 $v \leftarrow Q.\mathsf{pop}()$;
15 **for *each*** $w \in \mathcal{N}_{\mathcal{P},\epsilon}[v]$ **do**
16 **if** w is unexplored **then**
17 $C \leftarrow C \cup \{w\}$; mark w as explored;
18 **if** $w \notin Q \wedge |\mathcal{N}_{\mathcal{P},\epsilon}[u]| \geq \mu$ **then**
19 $Q.\mathsf{push}(w)$;
20 **else**
21 mark w as non-core vertex;
22 $\mathbb{C} \leftarrow \mathbb{C} \cup \{C\}$; $C \leftarrow \emptyset$;
23 mark all non-core vertex as unexplored;
24 **for *each*** $u \in V'$ **do**
25 **if** u's neighbors belong to different clusters **then**
26 mark u as hub;
27 **else**
28 mark u as outlier;

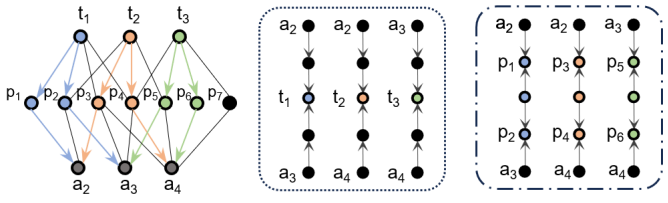
Algorithm. Following a similar framework as Algorithm 2, IHINScan first constructs $G_{\mathcal{P}}$ based on $\mathcal{G}$ (line 1). Then, it iterates each vertex in $V(G_{\mathcal{P}})$ (line 2), and for each neighbor of v of u (line 3), it first finds the independent common meta-path neighbors by procedure findIMPN (line 4) and computes the independent meta-path structural similarity $\sigma^{\mathcal{I}}_{\mathcal{P}}(u, v)$ following Definition 3.13 (line 5). If $\sigma^{\mathcal{I}}_{\mathcal{P}}(u, v) \geq \epsilon$ (line 6), u and v are added into $\mathcal{N}_{\epsilon}[v]$ and $\mathcal{N}_{\epsilon}[u]$, respectively (line 6-7). After the independent meta-path structural similarity regarding all connected vertex pairs in $G_{\mathcal{P}}$ is obtained, it computes the clusters in $G_{\mathcal{P}}$ based on Definition 3.10. Specifically, for each core vertex v in $G_{\mathcal{P}}$ (line 14), it explores the vertices $w \in \mathcal{N}_{\mathcal{P},\epsilon}[v]$ (line 15) and put them in the same cluster (line 17). If w is also a core vertex (line 18), w is further pushed in to Q (line 19). The procedure continues until Q is empty (line 13). When all the clusters are found, it computes the hub vertices and outlier vertices based on Definition 3.11, respectively (line 24-28).

Algorithm 4: findIMPN$(\mathcal{N}, u, v, \mathcal{P})$

1 $\mathcal{I}_{\mathcal{P}} \leftarrow \emptyset$;
2 **for** $w \in \mathcal{N}$ **do**
3 $\mathcal{S}_{\mathsf{cand}} \leftarrow \{(u, v, \mathcal{P}), (u, w, \mathcal{P}), (v, w, \mathcal{P})\}$;
4 **if** $\mathsf{verify}(\mathcal{S}_{\mathsf{cand}})$ **then** $\mathcal{I}_{\mathcal{P}}.\mathsf{add}(w)$;
5 **return** $\mathcal{I}_{\mathcal{P}}$;
6 **Procedure** $\mathsf{verify}(\mathcal{S}_{\mathsf{cand}})$
7 $i \leftarrow 1$;
8 **while** $i \leq |\mathcal{S}_{\mathsf{cand}}|$ **do**
9 $\mathcal{N}'_i \leftarrow \emptyset$, $\mathcal{N}''_i \leftarrow \emptyset$;
10 $\mathsf{findMPN}(\mathcal{G}, \mathcal{S}_{\mathsf{cand}}[i].\mathcal{P}[0 \ldots \lceil \frac{|\mathcal{P}|}{2} \rceil], 0, \mathcal{S}_{\mathsf{cand}}[i].u, \mathcal{N}'_i, \emptyset)$;
11 $\mathsf{findMPN}(\mathcal{G}, \mathcal{S}_{\mathsf{cand}}[i].\mathcal{P}[\lceil \frac{|\mathcal{P}|}{2} \rceil \ldots |\mathcal{P}|], 0, \mathcal{S}_{\mathsf{cand}}[i].v, \mathcal{N}''_i, \emptyset)$;
12 $\mathcal{N}_i \leftarrow \mathcal{N}'_i \cap \mathcal{N}''_i$; i++;
13 $\mathcal{N}_{\mathsf{cand}}[\cdot] \leftarrow$ sort $\mathcal{N}_1, \cdots, \mathcal{N}_{|\mathcal{S}_{\mathsf{cand}}|}$ in increasing order of size;
14 **return** $\mathsf{enum}(0, \emptyset, \mathcal{N}_{\mathsf{cand}}[\cdot])$;
15 **Procedure** $\mathsf{enum}(i, \mathcal{S}, \mathcal{N}_{\mathsf{cand}}[\cdot])$
16 **if** $i \geq |\mathcal{S}_{\mathsf{cand}}|$ **then**
17 $\mathcal{S}'_{\mathsf{cand}} \leftarrow \emptyset$;
18 **for** $j \leftarrow 1$ *to* $|\mathcal{S}_{\mathsf{cand}}|$ **do**
19 $\mathcal{P}_l \leftarrow \mathcal{S}_{\mathsf{cand}}[j].\mathcal{P}[0 \ldots \lceil \frac{|\mathcal{P}|}{2} \rceil]$;
20 $\mathcal{P}_r \leftarrow \mathcal{S}_{\mathsf{cand}}[j].\mathcal{P}[\lceil \frac{|\mathcal{P}|}{2} \rceil \ldots |\mathcal{P}|]$;
21 **if** $|\mathcal{P}_l| \geq 2$ **then**
22 $\mathcal{S}'_{\mathsf{cand}} \leftarrow \mathcal{S}'_{\mathsf{cand}} \cup \{(\mathcal{S}_{\mathsf{cand}}[j].u, \mathcal{S}[j], \mathcal{P}_l\}$;
23 **if** $|\mathcal{P}_r| \geq 2$ **then**
24 $\mathcal{S}'_{\mathsf{cand}} \leftarrow \mathcal{S}'_{\mathsf{cand}} \cup \{(\mathcal{S}_{\mathsf{cand}}[j].v, \mathcal{S}[j], \mathcal{P}_r]\}$;
25 **if** $\mathcal{S}'_{\mathsf{cand}} = \emptyset$ **or** $\mathsf{verify}(\mathcal{S}'_{\mathsf{cand}})$ **then return** true;
26 **else return** false;
27 **else**
28 **for** $v \in \mathcal{N}_{\mathsf{cand}}[i] \wedge v \notin \mathcal{S}[0 \ldots i-1]$ **do**
29 $\mathcal{S}[i] \leftarrow v$;
30 **if** $\mathsf{enum}(i+1, \mathcal{S}, \mathcal{N}_{\mathsf{cand}}[\cdot])$ **then return** true;
31 **return** false;

Procedure findIMPN (Algorithm 4) is used to verify whether the vertices in $\mathcal{N}$ are the independent common meta-path neighbors between two vertices u and v regarding $\mathcal{P}$. To achieve this goal, the sub-routine verify is invoke with parameters $(u, v, \mathcal{P})$, $(u, w, \mathcal{P})$, and $(v, w, \mathcal{P})$ for each vertex $w \in \mathcal{N}$ (line 2-3). If w passes the verification, then w is added into $\mathcal{I}_{\mathcal{P}}$ (line 4). The independent common meta-path neighbors are

returned in line 5. The sub-routine verify checks whether there exists three independent path instances of $\mathcal{P}$ connecting (u, v), (u, w) and (v, w) and they do not share any vertex except u, v, and w following Definition 3.12. Intuitively, if there exist three independent path instances, then there must exists three different vertices x_1, x_2, x_3 with vertex type $\mathcal{P}[\lceil \frac{|\mathcal{P}|}{2} \rceil]$ at $\lceil \frac{|\mathcal{P}|}{2} \rceil$ position of these path instances. Recursively, there must further exist six different vertex pairs (u, x_1), (x_1, v), (v, x_2), (x_2, w), (w, x_3), and (x_3, u) with independent path instances similarly. The sub-routine verify conduct the above recursive verification until the length of divided meta-path is 2. Specifically, for each pair $(u, v, \mathcal{P}) \in \mathcal{S}_{\mathsf{cand}}$ (line 8), it computes common neighbors of u / v starting from u / v following the meta-path $\mathcal{P}[0, \ldots, \lceil \frac{|\mathcal{P}|}{2} \rceil]$ / $\mathcal{P}[\lceil \frac{|\mathcal{P}|}{2} \rceil, \ldots, |\mathcal{P}|]$, and store them in $\mathcal{N}_i$ where $1 \leq i \leq |\mathcal{S}_{\mathsf{cand}}|$ (line 9-12). Then, it sorts them in the increasing order of their size (line 13). After that, by retrieving each vertex in $\mathcal{N}_i$, it checks whether exists a combination of the retrieved vertices such that all the vertices are different by $\mathcal{S}$ through procedure enum (line 14, line 28-30). If such a combination exists, it further splits each pair $(u, v, \mathcal{P}) \in \mathcal{S}_{\mathsf{cand}}$ into two pairs and repeats the above procedure recursively until $|\mathcal{P}| = 2$ is achieved (line 16-26).



(a) Subnetwork of $\mathcal{G}$ (b) Step1 (c) Step2

Fig. 6. A running example Algorithm 4 for a_2, a_3, and a_4

Example 5.1: Consider the HIN $\mathcal{G}$ in Fig. 2 and assume the given meta-path is $\mathcal{P}_2 = (APTPA)$, Fig. 6 shows the steps to verify whether there exist three independent path instances among a_2, a_3, and a_4. At step 1, take a_2 and a_3 as an example, $\mathcal{P}_2$ are divided into two sub-meta-path $\mathcal{P}_2' = (APT)$ and $\mathcal{P}_2'' = (TPA)$ and $\mathcal{P}[\lceil \frac{|\mathcal{P}|}{2} \rceil] = T$. And we have $\mathcal{N}_1' = \{t_0, t_1, t_2\}$ and $\mathcal{N}_1'' = \{t_1, t_2, t_3\}$. Thus, $\mathcal{N}_{\mathsf{cand}}[1] = \{t_1, t_2\}$. $\mathcal{N}_{\mathsf{cand}}[2] = \{t_1, t_2\}$ and $\mathcal{N}_{\mathsf{cand}}[3] = \{t_1, t_2, t_3\}$ can be obtained similarly. Then we find 3 different vertices $t_1 \in \mathcal{N}_{\mathsf{cand}}[1], t_2 \in \mathcal{N}_{\mathsf{cand}}[2]$ and $t_3 \in \mathcal{N}_{\mathsf{cand}}[3]$ by trying different combinations of vertices from $\mathcal{N}_{\mathsf{cand}}[1]$, $\mathcal{N}_{\mathsf{cand}}[2]$ and $\mathcal{N}_{\mathsf{cand}}[3]$ following the procedure enum in Algorithm 4. Therefore, we continue to step 2. At step 2, due to the existence of t_1, t_2 and t_3, we has to verify whether there exist 6 different vertices with type P, which connect pairs $(t_1, a_2), (t_1, a_3), (t_2, a_2), (t_2, a_4), (t_3, a_3)$ and (t_3, a_4), respectively. This is similar to the previous process of verifying t_1, t_2 and t_3 at step 1. Finally, we find three independent path instances, i.e. $(a_2, p_1, t_1, p_2, a_3), (a_2, p_3, t_2, p_4, a_4)$ and $(a_3, p_5, t_3, p_6, a_4)$, which means a_4 is an independent meta-path neighbor of a_2 and a_3.

Theorem 5.1: *Given a meta-path $\mathcal{P}$ on an* HIN *$\mathcal{G}$, two parameters ϵ and μ, the time complexity of Algorithm 3 to conduct the clustering is $O(|V(G_{\mathcal{P}})| \cdot \mathsf{deg}_{\mathsf{max}}^{|\mathcal{P}|} + |E(G_{\mathcal{P}})|^{1.5} \cdot \nu)$, where $|V(G_{\mathcal{P}})|/|E(G_{\mathcal{P}})|$ denotes the number of vertices/edges in $G_{\mathcal{P}}$ respectively, $\mathsf{deg}_{\mathsf{max}}$ denotes the maximum degree of vertices in $\mathcal{G}$, ν denotes the time used for the procedure* verify.

Proof: Algorithm 3 follows the same framework of Algorithm 1. Therefore, the time complexity to construct $G_{\mathcal{P}}$ is $O(|V(G_{\mathcal{P}})| \cdot \mathsf{deg}_{\mathsf{max}}^{|\mathcal{P}|})$. Following [44], the total number of common meta-path neighbors for all vertex pairs connecting by an edge can be bounded by $O(|E(G_{\mathcal{P}})|^{1.5})$, and each common meta-path neighbor needs to invoke verify once. Therefore, the time complexity of clustering is $O(|E(G_{\mathcal{P}})|^{1.5} \cdot \nu)$. Therefore, the theorem holds. □

B. A pSCAN Enhanced Pruning Algorithm

Algorithm 5: IHINScan$^+(\mathcal{G}, \mathcal{P}, \epsilon, \mu)$

```
1  construct G_P based on G (line 1-12 of Algorithm 2);
2  pSCAN(G_P, ε, μ);
3  C ← {v | v ∈ V(G_P) ∧ lb(v) ≥ μ}, C' ← ∅;
4  for each u ∈ C do
5      N ← {w ∈ nbr(u, G_P) | σ(u,w) = ∅ ∨ σ(u,w) ≥ ε}, i ← 0;
6      while ub(u) ≥ μ ∧ lb(u) + |N| − i ≥ μ ∧ ub(u) − |N| + i < μ ∧ i < |N| do
7          v ← N[i];
8          if σ(u,v) ≥ ε then
9              σ(u,v) ← findIMPN(N[u], N[v]) / sqrt(|N[u]| · |N[v]|);
10             if σ(u,v) < ε then
11                 lb(u) ← lb(u) − 1; lb(v) ← lb(v) − 1;
12                 ub(u) ← ub(u) − 1; ub(v) ← ub(v) − 1;
13         else
14             σ(u,v) ← findIMPN(N[u], N[v]) / sqrt(|N[u]| · |N[v]|);
15             if σ(u,v) ≥ ε then
16                 lb(u) ← lb(u) + 1; lb(v) ← lb(v) + 1;
17             else ub(u) ← ub(u) − 1; ub(v) ← ub(v) − 1;
18     i ← i + 1;
19     if lb(u) ≥ μ then C' ← C' ∪ {u};
20 for each u ∈ C' do
21     for each v ∈ nbr(u, G_P) do
22         if lb(v) ≥ μ ∧ σ(u,v) ≥ ε then u, v in the same cluster;
23 ℂ ← the set of clusters of core vertices;
24 for each C ∈ ℂ do
25     for each u ∈ C do
26         for each v ∈ nbr(u, G_P) do
27             if lb(v) < μ ∧ v ∉ C then
28                 if σ(u,v) ≥ ε ∨ σ(u,v) = ∅ then
29                     σ(u,v) ← findIMPN(N[u], N[v]) / sqrt(|N[u]| · |N[v]|);
30                 if σ(u,v) ≥ ε then C ← C ∪ {v};
31 for each vertex u not in the cluster do
32     line 25-28 of Algorithm 3;
```

Based on Theorem 5.1, it is clear that invoking the procedure findIMPN to compute the independent common meta-path neighbors is the most time-consuming part of Algorithm 3. Therefore, to improve the performance of independent HINSCAN, the key is to avoid the unnecessary invocation of findIMPN. To achieve this goal, we propose a pSCAN enhanced pruning algorithm in which the lower bound and upper bound of $|N_{\mathcal{P},\epsilon}[u]|$ for each vertex u is used to prune the unnecessary computation of independent common meta-

path neighbors. Before introducing the algorithm, we have the following lemma:

Lemma 5.1: *Given a meta-path $\mathcal{P}$ on an* HIN $\mathcal{G}$*, for two vertices u, v, $\sigma^{\mathcal{I}}_{\mathcal{P}}(u, v) \leq \sigma_{\mathcal{P}}(u, v)$.*

Following Lemma 5.1, it is clear that $\sigma_{\mathcal{P}}(u, v)$ is an upper bound of $\sigma^{\mathcal{I}}_{\mathcal{P}}(u, v)$. Moreover, based on Definition 3.10, we can derive that if a vertex is not a core vertex in non-independent HINSCAN, it cannot be a core vertex in independent HINSCAN. Considering that pSCAN is very fast as verified in our experiments, we can perform pSCAN on $G_{\mathcal{P}}$ similarly as Algorithm 2 first, and only the vertices in the clusters need to be verified according to Lemma 5.1. Following this idea, a direct approach is that we use Algorithm 3 to identify the roles of vertices in the clusters and the time-consuming invocation of findIMPN for the hub vertices and outlier vertices can be avoided.

However, this approach still does not fully exploit the pruning potential of pSCAN. Revisit the procedure of pSCAN in the above direct approach, it maintains the lower bound and upper bound of the number of ϵ meta-path neighbors for each vertex to further prune the unnecessary computation during the clustering in essential. Obviously, the above direct approach overlooked such bounds when identifying the roles of the vertices in the clusters. This inspires us to think that whether it is possible to further prune the invocations of findIMPN by reusing these bounds to identifying the roles of the vertices in the clusters. For the ease of presentation, we use $\mathsf{lb}(u)$ and $\mathsf{ub}(u)$ to indicate the lower bound and upper bound of the number of ϵ meta-path neighbors that pSCAN maintained for a vertex u, $\mathcal{N}$ be the set of neighbors w of u in $G_{\mathcal{P}}$ such that $\sigma(u, w) \geq \epsilon$ or $\sigma(u, w)$ is not computed by pSCAN, and we have the following lemma regarding a independent core vertex:

Lemma 5.2: *Given a meta-path $\mathcal{P}$ on an* HIN $\mathcal{G}$ *and two parameters ϵ and μ, for a vertex $u \in V(\mathcal{G})$, if u is a core vertex of independent* HINSCAN*, then (1) $\mathsf{ub}(u) \geq \mu$, (2) $\mathsf{lb}(u) + |\mathcal{N}| - i \geq \mu$, (3) $\mathsf{ub}(u) - |\mathcal{N}| + i < \mu$, where $0 \leq i \leq |\mathcal{N}|$.*

Algorithm. Based on the above analysis, our pSCAN enhanced pruning algorithm for independent HINSCAN is shown in Algorithm 5. It first constructs $G_{\mathcal{P}}$ based on $\mathcal{G}$ in the same way as Algorithm 2 (line 1). Then, pSCAN is used to prune the hub vertices and outlier vertices as analyzed above (line 2). For the vertices in the clusters, it first retrieves the core vertices in non-independent HINSCAN, namely $\mathsf{lb}(v) \geq \mu$ and stores them in $\mathcal{C}$ (line 3). Then, for each core vertex of non-independent HINSCAN $u \in \mathcal{C}$ (line 4), $\mathcal{N}$ stores the neighbors of u in $G_{\mathcal{P}}$ that $\sigma(u, w)$ is not computed or not less than ϵ (line 5). For each $v \in \mathcal{N}$, it iteratively updates $\sigma(u, v)$ by findIMPN (line 9, 14). Based on the updated $\sigma(u, v)$, it maintains $\mathsf{lb}(u)$, $\mathsf{lb}(v)$, $\mathsf{ub}(u)$, and $\mathsf{ub}(v)$ accordingly (line 10-12, line 16-17). If $\mathsf{lb}(u) \geq \mu$, u is a core vertex of non-independent HINSCAN clearly, and is stored in $\mathcal{C}'$ (line 19). When all the neighbors in $\mathcal{N}$ of u are verified and $\mathsf{lb}(u) \geq \mu$, then u is a core vertex of independent HINSCAN and recorded in $\mathcal{C}'$ (line 19). After that, two adjacent core vertices in independent HINSCAN are mark in the same cluster (line 20-22). Until now, all the core vertices of non-independent HINSCAN have been identified. Then, it computes the non-core vertices in independent HINSCAN in each clusters (line 24-30). Specifically, for each core vertex $u \in C$ in independent HINSCAN, it iterates it neighbors v in $G_{\mathcal{P}}$ that is not a core vertex of non-independent HINSCAN (line 27). If $\sigma(u, v) \geq \epsilon$ and is not updated or it is not computed before, it computes the independent meta-path structural similarity by findIMPN (line 28-29). If the independent meta-path structural similarity between u and v is not less than ϵ, it is a non-core vertex of non-independent HINSCAN in C (line 30). After all the vertices in the clusters have been determined, the hub vertices and outlier vertices are adjusted similarly as Algorithm 3.

TABLE II
DATASETS USED IN EXPERIMENTS

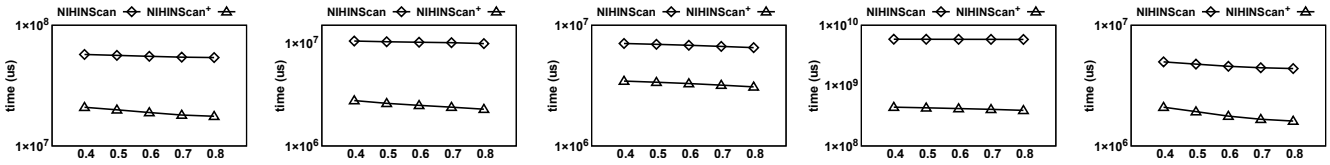
Datasets	n	m	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{R}\rvert$	#Meta-Paths
TMDB	71,978	113,581	7	12	37
IMDB	854,616	3,898,144	4	3	12
DBLP	2,056,444	6,607,065	4	3	11
Foursquare	4,472,122	10,200,000	4	3	8
DBpedia	5,900,558	17,961,887	413	637	50

VI. EXPERIMENTS

This section presents our experimental results. All the experiments are conducted on a machine with an Intel Xeon 2.4GHz CPU and 256GB main memory, running Ubuntu 20.04.6 LTS, 64 bit.

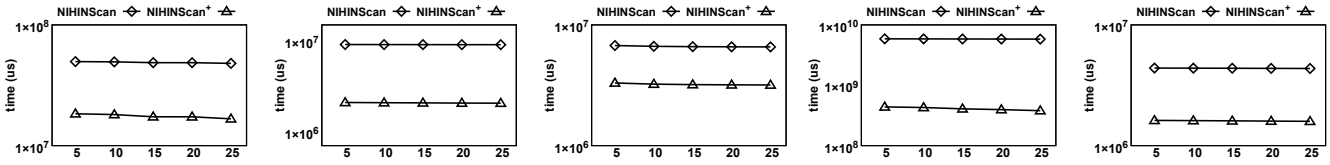
Dataset. We evaluate the algorithms on five real heterogeneous information network: TMDB, IMDB, DBLP, Foursquare, and DBPedia. TMDB is a movie knowledge graph including entities like movies, actors, casts, crews and companies. IMDB contains the movie rating records since 2000, and it has four types of vertices (authors, directors, writers and movies). DBLP includes publication records in computer science areas, and the vertex types are authors, papers, venues and topics. Foursquare contains the check-in records in US, which has four types of vertices, including records, users, venues, and categories. DBPedia contains the data extracted from Wikipedia infoboxes using the mapping-based extraction. The details are shown in Table II.

Meta-path Configuration. For the first four datasets, we collect all the possible symmetric meta-paths with lengths less than four according to the corresponding schema network. For the remaining one, since it does not have the pre-defined schema, we construct the schema by following the method mentioned in [11]. Afterwards, by traversing the schema network, we collect 10 meta-paths, where 5 meta-paths are with lengths two and four respectively. Related information is reported in Table II. For each dataset, we obtain top-5 meta-paths of the highest frequencies (i.e., the number of meta-path instances) from all meta-paths, then perform NIHINSCAN and IHINSCAN for each meta-path, and report the average result in the following experiment.



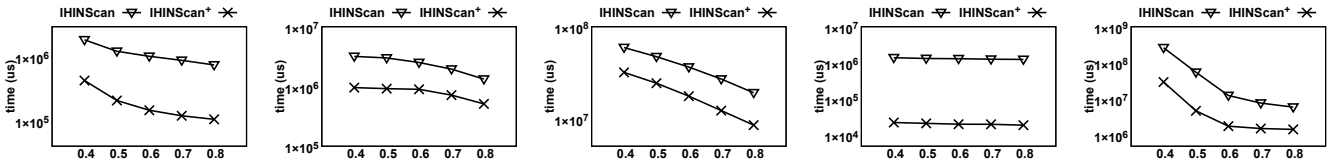
(a) TMDB (b) IMDB (c) DBLP (d) Foursquare (e) DBPedia

Fig. 7. Performance of NIHINScan and NIHINScan$^+$ when varying ϵ



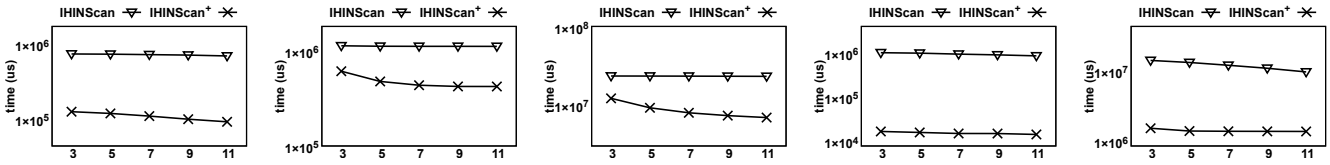
(a) TMDB (b) IMDB (c) DBLP (d) Foursquare (e) DBPedia

Fig. 8. Performance of NIHINScan and NIHINScan$^+$ when varying μ



(a) TMDB (b) IMDB (c) DBLP (d) Foursquare (e) DBPedia

Fig. 9. Performance of IHINScan and IHINScan$^+$ when varying ϵ



(a) TMDB (b) IMDB (c) DBLP (d) Foursquare (e) DBPedia

Fig. 10. Performance of IHINScan and IHINScan$^+$ when varying μ

Algorithm. We evaluate the following algorithms:

- NIHINScan: Algorithm 1 (Section IV-A).
- NIHINScan$^+$: Algorithm 2 (Section IV-B).
- IHINScan: Algorithm 3 (Section V-A).
- IHINScan$^+$: Algorithm 5 (Section V-B).
- PathSimCluster: HINSCAN clustering model by replacing Definition 3.6 with PathSim [10]

All algorithms are implemented in C++ and compiled with GNU GCC 10.5.0 and -O3 level optimization. The time cost of algorithms is measured as the amount of elapsed wall-clock time during the execution. In the experiments, we first evaluate the efficiency of the non-independent HINSCAN and independent HINSCAN model, then the effectiveness of the proposed algorithms. To evaluate the effectiveness of our model, as SCAN [26] suggests that an ϵ value between 0.4 and 0.8 is normally sufficient to achieve a good clustering result for SCAN, we set μ as 3 and tune the value of ϵ between 0.4 and 0.8 to obtain a good clustering result.

Exp-1: Performance of NIHINScan **and** NIHINScan$^+$**.** In this experiment, we evaluate the clustering performance of NIHINScan and NIHINScan$^+$ by varying ϵ and μ, respectively. For ϵ, we vary its value from 0.4 to 0.8 in increments of 0.1. For μ, we vary its values from 5 to 25 in increments of 5. The running times are reported in Fig. 7 and Fig. 8.

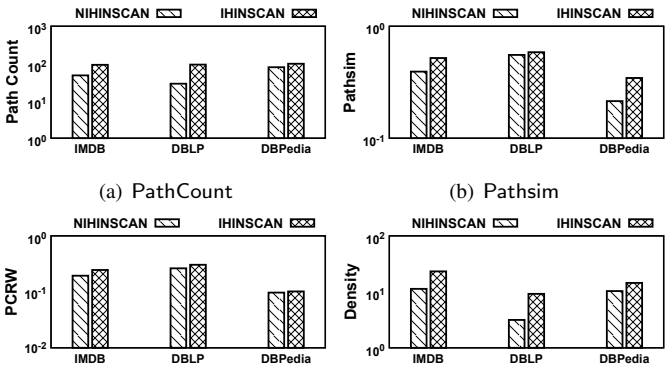
The results show that NIHINScan$^+$ runs much faster than NIHINScan. This performance improvement is primarily due to the optimized algorithm's efficient use of explored vertex information, which minimizes redundant operations. Moreover, as μ and ϵ vary, although the running time is shortened, the change is very small. This is because, in the non-independent HINSCAN model, constructing the meta-path transformed graph often takes up a large part of the time, whereas pscan takes relatively little time. Since varying μ or ϵ only affects pscan, the overall running time remains largely unchanged.

Exp-2: Performance of IHINScan **and** IHINScan$^+$**.** In this experiment, we evaluate the clustering performance of IHINScan and IHINScan$^+$ by varying ϵ and μ. For ϵ, we vary its value from 0.4 to 0.8 in increments of 0.1. For μ, we vary its values from 5 to 25 in increments of 5. The processing times for all five datasets are reported in Fig. 9 and Fig. 10.

The results indicate that IHINScan$^+$ significantly outperforms IHINScan. This is mainly because findIMPN (Algorithm 4) is a time-consuming procedure, while IHINScan$^+$ makes full use of the information of pSCAN in the previous stage, thereby avoiding a lot of unnecessary time for executing

findIMPN compared with IHINScan. In addition, as μ and ϵ increase, both algorithms exhibit significant reductions in running time. This is mainly because in the independent HINSCAN model, the clustering phase takes much more time duo to the time-consuming findIMPN procedure, and larger value of μ or ϵ generally means more computations can be pruned, and thus the whole running time is reduced.

Exp-3: Cohesiveness Analysis. To measure the cohesiveness, we compute and compare the similarity and density of the clusters on large datasets. Specifically, for each cluster in NIHINSCAN and IHINSCAN, we compute the average similarity value of the two vertices in each $\mathcal{P}$-pair using metrics PathCount [45], PathSim [10], and PCRW [46]. To measure the density of clusters, we follow the density measure in [12], which is the number of $\mathcal{P}$-pairs over the number of vertices (here, all the vertices are with the target type) as density. We report the average results in Fig. 11. Clearly, the clusters in IHINSCAN achieve higher similarity values than those of NIHINSCAN. Therefore, IHINSCAN can obtain clusters with vertices of higher similarity values. Besides, the clusters in NIHINSCAN have a lower density than those of IHINSCAN. Therefore, the IHINSCAN model can obtain clusters with vertices that tend to be more densely connected.



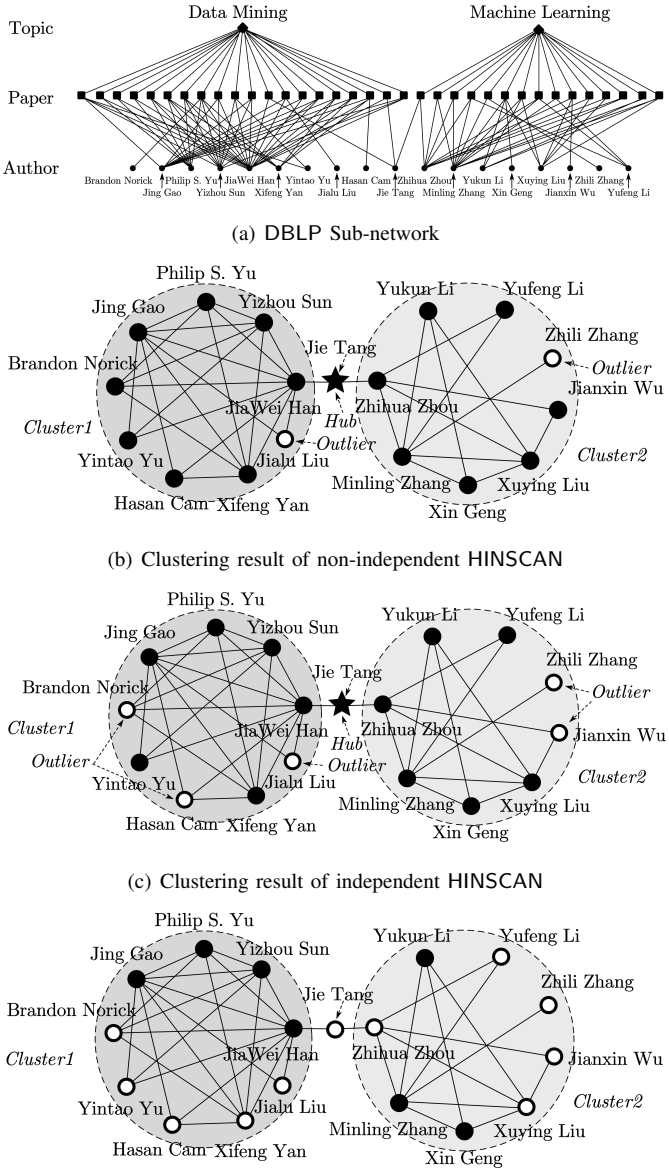
(a) PathCount (b) Pathsim

(c) PCRW (d) Density

Fig. 11. Cohesiveness Analysis of NIHINSCAN and IHINSCAN

Exp-4: Case Study on DBLP Network. DBLP is a bibliographic HIN that captures the collaboration relationships between researchers in the computer science. It comprises three types of vertices: Author (A), Paper (P), and Topic (T). Fig. 12 illustrates the HINSCAN clustering results on the DBLP sub-network centered on "Philip S. Yu" (Fig. 12 (a)).

As shown in Fig. 12 (b), the non-independent HINSCAN model categorizes the researchers into two clusters, corresponding to the data mining and the machine learning areas. "Jie Tang" is identified as a hub due to his close collaborations with researchers in both fields, whereas "Jialu Liu" and "Zhili Zhang" are considered outliers because they lack close cooperation within their respective fields. The clustering result of the independent HINSCAN model is shown in Fig. 12 (c). This model further identifies "Brandon Norick", "Hasan Cam", and "Jianxin Wu" as outliers, in addition to those identified in Fig. 12 (b). This is because, although these



(a) DBLP Sub-network

(b) Clustering result of non-independent HINSCAN

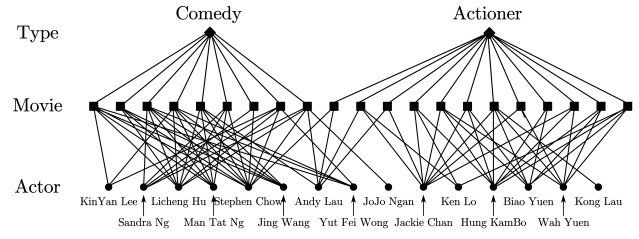
(c) Clustering result of independent HINSCAN

(d) Clustering result of PathSimCluster ($\epsilon = 0.52$)

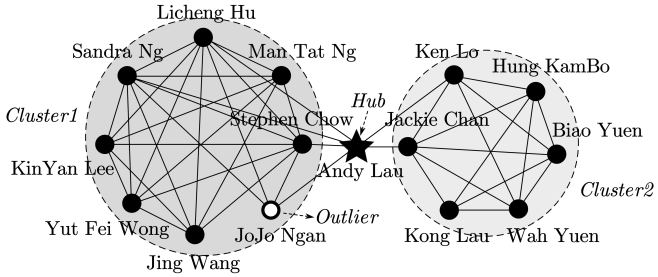
Fig. 12. Case Study on DBLP ($\mu = 3$, $\epsilon = 0.64$, $\mathcal{P} = (APA)$)

individuals have collaborated with many researchers in their areas, these connections are typically established through one or two papers, which do not constitute strong collaborative relationships. Comparing the results in Fig. 12 (b) and Fig. 12 (c), it is evident that the independent HINSCAN model identifies denser clusters than the non-independent HINSCAN model. Regarding PathSimCluster (Fig. 12 (c)), "Jie Tang" is incorrectly identified as an outlier, which implies PathSim is not a suitable alternative to structural similarity measure regarding SCAN on HINs.

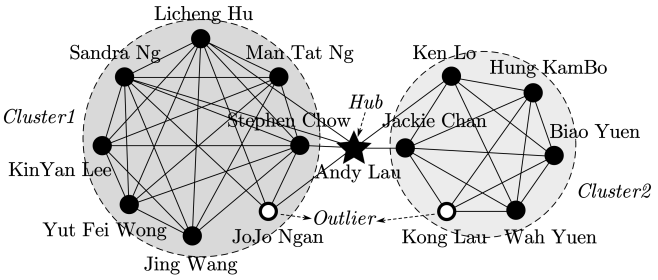
Exp-5: Case Study on IMDB Network. IMDB is a HIN that records the relationships between movies and actors. It contains three types of vertices, namely Actor (A), Movie (M), and Type (T). Fig. 13 presents the HINSCAN clustering results on the IMDB sub-network centered on "Stephen Chow" (Fig. 13 (a)).



(a) IMDB Sub-network



(b) Clustering result of non-independent HINSCAN



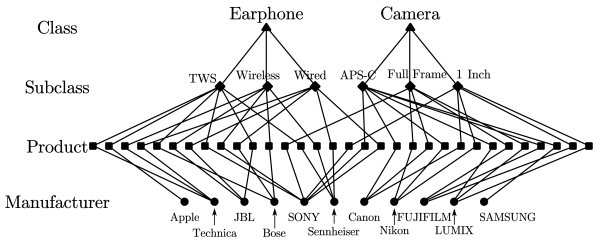
(c) Clustering result of independent HINSCAN

Fig. 13. Case Study on IMDB ($\mu = 3$, $\epsilon = 0.75$, $\mathcal{P} = (AMA)$)

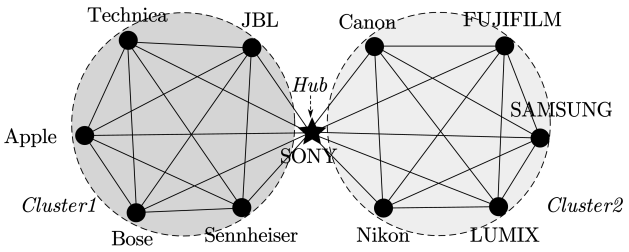
As shown in Fig. 13 (b), the non-independent HINSCAN model divides the actors into two clusters, corresponding to the comedy movie and the action movie genres. "Andy Lau" is identified as a hub due to his close collaborations with actors in both genres, whereas "JoJo Ngan" is considered an outlier because he lacks close cooperation with other actors in comedy movies. Fig. 13 (c) shows the clustering result of the independent HINSCAN model, which reveals a denser clustering outcome and additionally considers "Kong Lau" as an outlier.

Exp-6: Case Study on Amazon Product Network. Amazon Product Network is an HIN that records electronic products and their manufacturers. The network contains four types of vertices: Manufacture (M), Product (P), Subclass (S), and Class (C). Fig. 14 shows the HINSCAN clustering results on the Amazon Product sub-network centered on "SONY" (Fig. 14 (a)).

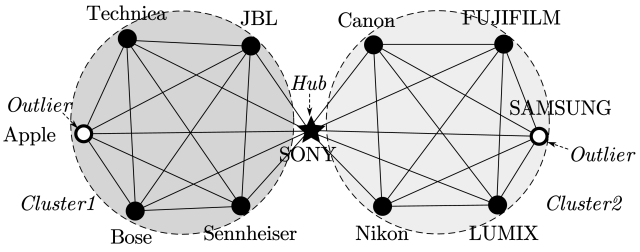
As shown in Fig. 14 (b), the non-independent HINSCAN model divides the manufacturer into two clusters, one for headphone manufacturers and the other for camera manufacturers. "SONY" is identified as a hub because it produces a significant number of both headphone and camera products. Fig. 14 (c) presents the clustering results of the independent HINSCAN model, revealing that "Apple" and "SAMSUNG" are considered outliers, unlike in the non-independent HINSCAN model. This distinction arises because, although both manufacturers produce headphones and cameras, the variety of their products is limited, resulting in weak connections established through these products.



(a) Amazon Product Sub-network



(b) Clustering result of non-independent HINSCAN



(c) Clustering result of independent HINSCAN

Fig. 14. Case Study on APDB ($\mu = 3$, $\epsilon = 0.75$, $\mathcal{P} = (MPSPM)$)

VII. Conclusion

In this paper, we explore the structural clustering problem in HINs. We first propose two models, non-independent HINSCAN and independent HINSCAN, for structural clustering in HINs. For the non-independent HINSCAN, we first propose a transformation-clustering framework and further improve the performance by introducing a new search-join transformation paradigm. For the independent HINSCAN, we design an efficient algorithm to verify the independence regarding the path instances, and further improve the whole clustering performance by introducing the lower bound and upper bound. The experimental results on real datasets demonstrate the effectiveness and efficiency of our proposed algorithms.

Acknowledge. Long Yuan is supported by National Key R&D Program of China 2022YFF0712100, NSFC62472225. Zi Chen is supported by NSFC6240071854, BK20241381, JSTJ-2023-XH055.

REFERENCES

[1] S. E. Schaeffer, "Graph clustering," *Computer science review*, vol. 1, no. 1, pp. 27–64, 2007.

[2] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the national academy of sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[3] W. Fan, R. Jin, M. Liu, P. Lu, X. Luo, R. Xu, Q. Yin, W. Yu, and J. Zhou, "Application driven graph partitioning," in *Proceedings of SIGMOD*, 2020, pp. 1765–1779.

[4] C. Biemann, "Chinese whispers-an efficient graph clustering algorithm and its application to natural language processing problems," in *Proceedings of TextGraphs: the First Workshop on Graph Based Methods for Natural Language Processing*, 2006, pp. 73–80.

[5] V.-S. Martha, Z. Liu, L. Guo, Z. Su, Y. Ye, H. Fang, D. Ding, W. Tong, and X. Xu, "Constructing a robust protein-protein interaction network by integrating multiple public databases," in *BMC bioinformatics*, vol. 12, no. 10, 2011, pp. 1–10.

[6] A. Bellogín and J. Parapar, "Using graph partitioning techniques for neighbour selection in user-based collaborative filtering," in *Proceedings of RecSys*, 2012, pp. 213–216.

[7] Y. Ding, M. Chen, Z. Liu, D. Ding, Y. Ye, M. Zhang, R. Kelly, L. Guo, Z. Su, S. C. Harris *et al.*, "atbionet–an integrated network analysis tool for genomics and biomarker discovery," *BMC genomics*, vol. 13, no. 1, pp. 1–12, 2012.

[8] M. Schinas, S. Papadopoulos, G. Petkos, Y. Kompatsiaris, and P. A. Mitkas, "Multimodal graph-based event detection and summarization in social media streams," in *Proceedings of SIGMM*, 2015, pp. 189–192.

[9] M. Schinas, S. Papadopoulos, Y. Kompatsiaris, and P. A. Mitkas, "Visual event summarization on social media using topic modelling and graph-based ranking algorithms," in *Proceedings of ICMR*, 2015, pp. 203–210.

[10] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu, "Pathsim: Meta path-based top-k similarity search in heterogeneous information networks," *PVLDB*, vol. 4, no. 11, pp. 992–1003, 2011.

[11] Y. Zhou, Y. Fang, W. Luo, and Y. Ye, "Influential community search over large heterogeneous information networks," *Proc. VLDB Endow.*, vol. 16, no. 8, pp. 2047–2060, 2023.

[12] Y. Yang, Y. Fang, X. Lin, and W. Zhang, "Effective and efficient truss computation over large heterogeneous information networks," in *Proceedings of ICDE*, 2020, pp. 901–912.

[13] Y. Sun, J. Han, C. C. Aggarwal, and N. V. Chawla, "When will it happen?: relationship prediction in heterogeneous information networks," in *Proceedings of WSDM*, 2012, pp. 663–672.

[14] X. Yu, X. Ren, Y. Sun, B. Sturt, U. Khandelwal, Q. Gu, B. Norick, and J. Han, "Recommendation in heterogeneous information networks with implicit user feedback," in *Proceedings of RecSys*, 2013, pp. 347–350.

[15] B. Liu, L. Yuan, X. Lin, L. Qin, W. Zhang, and J. Zhou, "Efficient (α, β)-core computation in bipartite graphs," *VLDB J.*, vol. 29, no. 5, pp. 1075–1099, 2020.

[16] D. Ouyang, L. Yuan, L. Qin, L. Chang, Y. Zhang, and X. Lin, "Efficient shortest path index maintenance on dynamic road networks with theoretical guarantees," *Proc. VLDB Endow.*, vol. 13, no. 5, pp. 602–615, 2020.

[17] L. Meng, Y. Shao, L. Yuan, L. Lai, P. Cheng, X. Li, W. Yu, W. Zhang, X. Lin, and J. Zhou, "A survey of distributed graph algorithms on massive graphs," *ACM Computing Surveys*, vol. 57, no. 2, pp. 1–39, 2024.

[18] Z. Chen, Y. Zhao, L. Yuan, X. Lin, and K. Wang, "Index-based biclique percolation communities search on bipartite graphs," in *Proceedings of ICDE*. IEEE, 2023, pp. 2699–2712.

[19] J. Zhang, L. Yuan, W. Li, L. Qin, Y. Zhang, and W. Zhang, "Label-constrained shortest path query processing on road networks," *The VLDB Journal*, vol. 33, no. 3, pp. 569–593, 2024.

[20] L. Yuan, X. Li, Z. Chen, X. Lin, X. Zhao, and W. Zhang, "I/o efficient label-constrained reachability queries in large graphs," *PVLDB*, vol. 17, no. 10, pp. 2590–2602, 2024.

[21] L. Yuan, K. Hao, X. Lin, and W. Zhang, "Batch hop-constrained st simple path query processing in large graphs," in *Proceedings of ICDE*. IEEE, 2024, pp. 2557–2569.

[22] Q. Liu, M. Zhao, X. Huang, J. Xu, and Y. Gao, "Truss-based community search over large directed graphs," in *Proceedings of SIGMOD*, 2020, pp. 2183–2197.

[23] Y. Gao, X. Qin, B. Zheng, and G. Chen, "Efficient reverse top-k boolean spatial keyword queries on road networks," *IEEE TKDE*, vol. 27, no. 5, pp. 1205–1218, 2015.

[24] Y. Gao, T. Zhang, L. Qiu, Q. Linghu, and G. Chen, "Time-respecting flow graph pattern matching on temporal graphs," *IEEE TKDE*, vol. 33, no. 10, pp. 3453–3467, 2021.

[25] Y. Zhou, H. Cheng, and J. X. Yu, "Graph clustering based on structural/attribute similarities," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 718–729, 2009.

[26] X. Xu, N. Yuruk, Z. Feng, and T. A. J. Schweiger, "SCAN: a structural clustering algorithm for networks," in *Proceedings of SIGKDD*, 2007, pp. 824–833.

[27] H. Shiokawa, Y. Fujiwara, and M. Onizuka, "Scan++ efficient algorithm for finding clusters, hubs and outliers on large-scale graphs," *PVLDB*, vol. 8, no. 11, pp. 1178–1189, 2015.

[28] L. Chang, W. Li, X. Lin, L. Qin, and W. Zhang, "pscan: Fast and exact structural graph clustering," in *Proceedings of ICDE*, 2016, pp. 253–264.

[29] W. Zhao, G. Chen, and X. Xu, "Anyscan: An efficient anytime framework with active learning for large-scale network clustering," in *Proceedings of ICDM*, 2017, pp. 665–674.

[30] T. R. Stovall, S. Kockara, and R. Avci, "GPUSCAN: gpu-based parallel structural clustering algorithm for networks," *IEEE TPDS*, vol. 26, no. 12, pp. 3381–3393, 2015.

[31] L. Yuan, Z. Zhou, X. Lin, Z. Chen, X. Zhao, and F. Zhang, "Gpuscan^{++}: Eficient structural graph clustering on gpus," *CoRR*, vol. abs/2311.12281, 2023.

[32] Q. Zhou and J. Wang, "Sparkscan: a structure similarity clustering algorithm on spark," in *National Conference on Big Data Technology and Applications*, 2015, pp. 163–177.

[33] J. H. Seo and M. H. Kim, "pm-scan: an I/O efficient structural clustering algorithm for large-scale graphs," in *Proceedings of CIKM*, 2017, pp. 2295–2298.

[34] Y. Che, S. Sun, and Q. Luo, "Parallelizing pruning-based graph structural clustering," in *Proceedings of ICPP*, 2018, pp. 77:1–77:10.

[35] D. Wen, L. Qin, Y. Zhang, L. Chang, and X. Lin, "Efficient structural graph clustering: An index-based approach," *PVLDB*, vol. 11, no. 3, pp. 243–255, 2017.

[36] B. Ruan, J. Gan, H. Wu, and A. Wirth, "Dynamic structural clustering on graphs," in *Proceedings of SIGMOD*, 2021, pp. 1491–1503.

[37] T. Tseng, L. Dhulipala, and J. Shun, "Parallel index-based structural graph clustering and its approximation," in *Proceedings of SIGMOD*, 2021, pp. 1851–1864.

[38] L. Meng, L. Yuan, Z. Chen, X. Lin, and S. Yang, "Index-based structural clustering on directed graphs," in *Proceedings of ICDE*, 2022, pp. 2831–2844.

[39] Y. Sun, Y. Yu, and J. Han, "Ranking-based clustering of heterogeneous information networks with star network schema," in *Proceedings of SIGKDD*, 2009, pp. 797–806.

[40] Y. Huang and X. Gao, "Clustering on heterogeneous networks," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 4, no. 3, pp. 213–233, 2014.

[41] X. Li, Y. Wu, M. Ester, B. Kao, X. Wang, and Y. Zheng, "Semi-supervised clustering in attributed heterogeneous information networks," in *Proceedings of World Wide Web*, 2017, pp. 1621–1629.

[42] X. Li, B. Kao, Z. Ren, and D. Yin, "Spectral clustering in heterogeneous information networks," in *Proceedings of AAAI*, vol. 33, no. 01, 2019, pp. 4221–4228.

[43] C. Shi, Y. Li, J. Zhang, Y. Sun, and S. Y. Philip, "A survey of heterogeneous information network analysis," *IEEE TKDE*, vol. 29, no. 1, pp. 17–37, 2016.

[44] L. Chang, W. Li, L. Qin, W. Zhang, and S. Yang, "pscan: fast and exact structural graph clustering," *IEEE TKDE*, vol. 29, no. 2, pp. 387–401, 2017.

[45] Y. Sun, R. Barber, M. Gupta, C. C. Aggarwal, and J. Han, "Co-author relationship prediction in heterogeneous bibliographic networks," in *2011 International Conference on Advances in Social Networks Analysis and Mining*, 2011, pp. 121–128.

[46] N. Lao and W. W. Cohen, "Relational retrieval using a combination of path-constrained random walks," *Mach. Learn.*, vol. 81, no. 1, p. 53–67, Oct. 2010. [Online]. Available: https://doi.org/10.1007/s10994-010-5205-8