Efficient Structural Clustering over Hypergraphs

Dong Pan*, Xu Zhou*, Lingwei Li*, Quanqing Xu†,
Chuanhui Yang†, Chenhao Ma‡, KenLi Li*
*College of Computer Science and Electronic Engineering, Hunan University, Changsha, China
† OceanBase, Ant Group, HangZhou, China ‡The Chinese University of Hong Kong, ShenZhen, China
Email:{pandong, zhxu, lilingwei, lkl}@hnu.edu.cn, {xuquanqing.xqq, rizhao.ych}@oceanbase.com, machenhao@cuhk.edu.cn

***Abstract*—Structural Graph Clustering is a well-known problem that aims to identify clusters and distinguish between special roles, such as hub and outlier. However, SCAN, the fundamental structural clustering model, is designed for pairwise graphs and fails to capture the unique structural information inherent in hypergraphs when clustering hypergraphs. Motivated by this, we propose a new structural clustering model, HSCAN, specifically for hypergraphs. We further design an Order-Index to accelerate fetching the key information of the HSCAN and a Lightweight Similarity Bucket Index to reduce the index cost. Next, we present an index-based sequential query algorithm with high performance and a parallel query algorithm to process large hypergraphs faster. Additionally, we provide the algorithms for constructing Order-Index and Lightweight Similarity Bucket Index. Extensive experiments on both real-world and synthetic datasets show that HSCAN performs better than existing models, and the two index-based query algorithms are up to three orders of magnitude faster than the existing algorithm.**

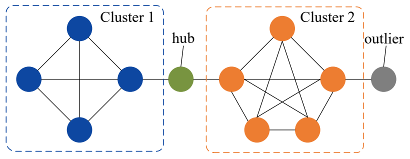
***Index Terms*—SCAN, graph algorithm, hypergraph, structural graph clustering**

I. INTRODUCTION

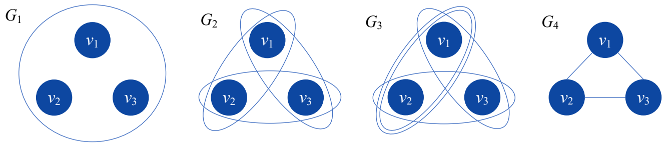
Hypergraph is a fundamental graph structure consisting of hyperedges in which each hyperedge connects an arbitrary number of nodes, which differs from pairwise graphs where each edge connects two vertices [1]–[3]. Recently, hypergraphs have attracted growing attention as they capture higher-order relationships among multiple entities [1], such as neural networks [4], academic networks [5], protein complex networks [6], and other real-life applications. There has been extensive research about hypercore maintenance [7], neighborhood-core decomposition [8], subgraph matching [9], and other problems [10]–[15]. In this paper, we focus on the structural graph clustering problem over hypergraphs.

Prior Works. *Structural Graph Clustering* [16] is a well-known problem that aims to cluster the vertices based on structural similarity (e.g., Cosine Similarity [17] or Jaccard Similarity [18]) and finds clusters, hubs, and outliers. SCAN [16] is the most fundamental structural clustering model. It calculates the structural similarity of each vertex pair using their neighborhoods. Then, SCAN constructs distinct clusters based on structural similarity scores. Other vertices are identified as hubs and outliers that do not belong to any cluster, while hubs bridge different clusters. An example of SCAN is shown in Fig. 1(a).

Application. Compared to other graph clustering methods, *Structural Graph Clustering* has unique outputs (similarity-based structural clusters, hubs, and outliers), which have a wide range of applications as follows:



(a) An example of SCAN



(b) Hypergraphs with different hyperedges

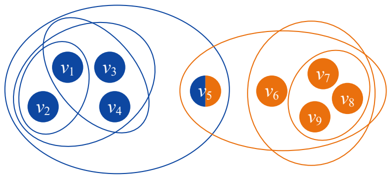
Fig. 1. Motivation.

- *Similarity-based structural clusters* have been widely applied to the analysis of biological data [19]–[22], social media data [23]–[26], and web data [27]–[29]. In addition, they are useful in image segmentation [30], image clustering [31], and fraud detection on blockchain data [32].
- *Hubs* can be treated as influential nodes and are meaningful in many fields, such as viral marketing [33], epidemiology [34], and graph compression [35]. *Outliers* can be isolated as noise and play a significant role in data mining [36]–[39].

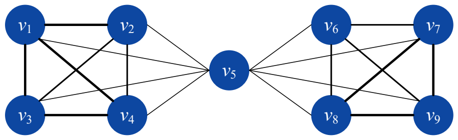
Motivation. *Why not apply SCAN to hypergraphs?* Although SCAN is an effective structural clustering model for undirected pairwise graphs, it is not effective for other types of graphs (e.g., directed graphs [40] and uncertain graphs [41], [42]). When transforming hypergraphs into pairwise graphs to apply the SCAN model, some similar hypergraphs with different structural relationships are transformed into the same pairwise graph, e.g., G_1, G_2, and G_3 shown in Fig. 1(b) are all converted to G_4. **This indicates that a significant amount of information within the hyperedges is lost when applying SCAN due to the unique nature of hyperedges.**

The following example demonstrates the outcome of applying SCAN to hypergraphs.

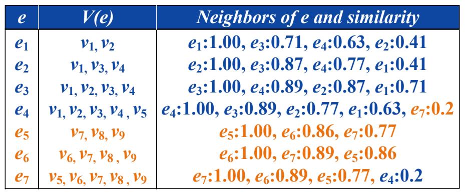
Example 1. *2(a) illustrates an example hypergraph. Fig. 2(b) depicts a pairwise graph converted from the example hypergraph, and Fig. 2(c) is the structural information of the example hypergraph. In the pairwise graph, for vertex v_5 and each of its neighbors v_i ($i \in [1-4] \cup [6-9]$), the cosine similarity score of their neighborhoods is 0.745.*



(a) Hypergraph



(b) Pairwise graph (each edge connects two vertices)



e	*V(e)*	*Neighbors of e and similarity*
e_1	v_1, v_2	e_1:1.00, e_3:0.71, e_4:0.63, e_2:0.41
e_2	v_1, v_3, v_4	e_2:1.00, e_3:0.87, e_4:0.77, e_1:0.41
e_3	v_1, v_2, v_3, v_4	e_3:1.00, e_4:0.89, e_2:0.87, e_1:0.71
e_4	v_1, v_2, v_3, v_4, v_5	e_4:1.00, e_3:0.89, e_2:0.77, e_1:0.63, e_7:0.2
e_5	v_7, v_8, v_9	e_5:1.00, e_6:0.86, e_7:0.77
e_6	v_6, v_7, v_8, v_9	e_6:1.00, e_7:0.89, e_5:0.86
e_7	v_5, v_6, v_7, v_8, v_9	e_7:1.00, e_6:0.89, e_5:0.77, e_4:0.2

(c) Structural information of Fig. 1(c)

Fig. 2. Example Graphs.

It indicates that v_5 is similar to all other vertex, and SCAN treats the entire graph as a cluster. It is inappropriate for hypergraphs, as the structural relationship between two groups $\{v_1, v_2, v_3, v_4, v_5\}$ and $\{v_5, v_6, v_7, v_8, v_9\}$ involves only two hyperedges that share a single common vertex. This weak connection contrasts sharply with the stronger internal structural relationships within each group.

Challenges. There are two main challenges in the structural clustering of hypergraphs as follows:

- *Challenge 1.* A new effective structural clustering model for hypergraphs. The new model should capture the unique characteristics of hypergraphs and extract the information carried by hyperedges.
- *Challenge 2.* An efficient approach for implementing structural clustering based on the new model. Considering the hypergraphs in real applications can be large, we need a new approach that minimizes space overhead while ensuring rapid implementation of structural clustering.

Our solution. To address *Challenge 1*, we design a new hyperedge-centric structural clustering model to facilitate extracting the structural information contained in hyperedges. Two hyperedges are considered neighbors if they share at least one common vertex. For such neighboring hyperedges, we use a single measure to compute the structural similarity between the sets of vertices contained in them. Additionally, we replace vertices with hyperedges in other definitions. Thus, we obtain a new structural clustering model that effectively captures the unique characteristics of hypergraphs. We can obtain vertex clusters directly by this new model. In particular, a vertex cluster is the set of vertices contained in hyperedges of its corresponding hyperedge cluster.

Why is a hyperedge-centric structural clustering model chosen? There are two main reasons leading to the new hyperedge-centered structural clustering model:

(1) The high-order relationships of vertices are important for improving the clustering quality in many novel clustering studies [43]–[46], and the unique structure to represent the high-order relationships of vertices in hypergraphs is hyperedge [47]. Thus, the hyperedge-centric model can gain better clustering results, as demonstrated by the experiments in Exp-1.

(2) A hyperedge represents a small vertex group (as a clique in pairwise graphs). Clustering these groups can be widely utilized in real-life applications, including named entity recognition systems [48], biomedical research literature analysis [49], among others.

In summary, a hyperedge-centric model can not only obtain vertex clusters with high-quality clustering results but also has a wide range of applications.

To tackle *Challenge 2*, considering that collecting cores and similar neighbors are two crucial tasks in structural clustering, we design an Order-Index to accelerate these collections. To reduce the cost of Order-Index, we present a Lightweight Similarity Bucket Index to organize key information into multiple buckets based on similarity instead of storing the similarity values. It provides approximate clustering results with the same query time complexity as Order-Index, which has a high clustering quality proved theoretically (approximate guarantee in Theorems 4 and 7) and experimentally (Exp-9) in this paper. Then, we present two query algorithms, sequential query algorithm and parallel query algorithm, based on the indexes proposed in this paper. These query algorithms efficiently handle the structural clustering within the new model. Additionally, we present the construction algorithms for building two indexes.

Contributions. Here are our main contributions:

- We propose a new structural clustering model, HSCAN, for clustering hypergraphs. HSCAN can capture the unique structural information of hypergraphs. To the best of our knowledge, this is the first work on structural clustering for hypergraphs (Section II).
- We design the Order-Index to boost query performance and introduce the Lightweight Similarity Bucket Index to reduce its cost (Section III).
- We propose two index-based query algorithms, sequential query algorithm, and parallel query algorithm, to efficiently handle the structural clustering within the new model (Section IV).
- We present the construction algorithms for building two proposed indexes efficiently (Section V).
- Extensive experiments on seven real-world datasets and a synthetic dataset demonstrate that HSCAN performs better than other clustering models (Section VI).

II. PRELIMINARIES

A. Problem Definition

In this section, we present the structural hypergraph clustering model. Table I summarizes frequently used notations and their meanings.

Notations in hypergraph. We consider an unweighted and undirected hypergraph $G{=}(V, E)$ on a finite set of vertices V, where $E \subset 2^V$ is a set of hyperedges. Each hyperedge $e \in E$ represents a set of $|e|$ vertices that interact. The hypergraph G can contain hyperedges consisting of only one vertex and may have repeated hyperedges. We denote the set of vertices contained in a certain hyperedge e as $V(e)$ and the set of hyperedges containing a specific vertex v as $E(v)$. Two hyperedges e_1 and e_2 are considered neighbors if they share at least one common vertex. We use $\mathcal{M}$ and $\mathcal{M}(G)$ to denote the set of such neighboring pairs.

Terminologies. e_1 connect e_2 if they share at least one common vertex, i.e., e_1 and e_2 are neighbors.

Definition 1 (Structural Similarity of Hyperedge)**.** *Given two hyperedges e_1 and e_2, the structural similarity between two hyperedges e_1 and e_2, denoted by $\sigma(e_1, e_2)$, is defined as the number of common vertices between hyperedge e_1 and e_2, normalized by the geometric mean of their cardinalities of the vertices they contain. That is*

$$\sigma(e_1, e_2) = \frac{|V(e_1) \cap V(e_2)|}{\sqrt{|V(e_1)| \cdot |V(e_2)|}}.$$

Definition 2 (Structural Neighborhood)**.** *The structural neighbor for a hyperedge e, denoted by $N[e]$, is defined as $N[e] = \{e' \in E | V(e) \cap V(e') \neq \emptyset\} \cup \{e\}$.*

Definition 3 (ϵ-Neighbor)**.** *The ϵ-neighborhood for a hyperedge e, denoted by $N_\epsilon[e]$, is defined as the subset of $N[e]$, in which every hyperedge e' satisfies $\sigma(e, e') \geq \epsilon$. That is, $N_\epsilon[e] = \{e' \in N[e] | \sigma(e, e') \geq \epsilon\}$.*

Definition 4 (Core Hyperedge)**.** *Given a similarity threshold $\epsilon(0 < \epsilon \leq 1)$ and an integer $\mu(\mu \geq 2)$, a hyperedge e is a core hyperedge if $|N_\epsilon[e]| \geq \mu$.*

Definition 5 (Structural Reachablity)**.** *Given two hyperedges e and e', e' is structurally reachable from e if there is a sequence of hyperedges $e_1, e_2, ...e_l \in E(l \geq 2)$ such that: (i) $e_1 = e, e_l = e'$; (ii) for all $1 \leq i \leq l-1$,e_i is core, and $e_{i+1} \in N_\epsilon[e_i]$.*

Definition 6 (Core Subgraph)**.** *Given the set of core hyperedges E_{core}, the core subgraph is a subgraph $G_{core} = (V_{core}, E_{core})$ of input graph G where $V_{core} = \{v \in V(e) | e \in E_{core}\}$.*

Definition 7 (Cluster)**.** *A cluster $C \in E$ is a non-empty subset of E such that:*

- *(Connectivity.) For any two hyperedges $e_1, e_2 \in C$, there exists a hyperedge $e \in C$ such that both e_1 and e_2 are structurally reachable from e.*
- *(Maximality.) For a core hyperedge $e \in C$, all hyperedges that are structurally reachable from e belong to C.*

Definition 8 (Cluster Subgraph)**.** *Given a cluster C, the cluster subgraph is a subgraph $G_C = (V_C, E_C)$ of the input graph G where (i) $E_C = \{e \in C\}$; (ii) $V_C = \{v \in V(e) | e \in C\}$.*

Definition 9 (Hub and Outlier)**.** *Given a hyperedge e that does not belong to any cluster, e is a hub if it has neighbors belonging to two or more different clusters. Otherwise,e is an outlier.*

Problem statement (HSCAN or (ϵ, μ)-HSCAN). Given a hypergraph $G = (V, E)$ and two parameters $0 < \epsilon \leq 1$ and $\mu \geq 2$, in this paper, we aim to efficiently compute the set of all clusters $\mathcal{C}_{\epsilon,\mu}$ in G and identify the corresponding role of each hyperedge.

TABLE I
FREQUENTLY-USED NOTATIONS.

Notation	Definition
$G = (V, E)$	A hypergraph with its vertex set and hyperedge set
$N[e]$	The set of structural neighbors for a hyperedge e
ϵ, μ	Two input parameters
$N_\epsilon[e]$	The set of ϵ-neighbors for a hyperedge e
$\mathcal{M}, \mathcal{M}(G)$	Set of neighboring hyperedges of G
$C, C_{\epsilon,\mu}$	A cluster for parameters ϵ, μ
$G_{core}, G_{core}^{\epsilon,\mu}$	A core subgraph for parameters ϵ, μ
$G_C, G_C^{\epsilon,\mu}$	The cluster subgraph of $C_{\epsilon,\mu}$
$\mathcal{C}, \mathcal{C}_{\epsilon,\mu}$	The set of clusters for parameters ϵ, μ
$\mathcal{G}_{core}, \mathcal{G}_{core}^{\epsilon,\mu}$	The union of core subgraphs for parameters ϵ, μ
$\mathcal{G}_C, \mathcal{G}_C^{\epsilon,\mu}$	The union of all cluster subgraphs of for parameters ϵ, μ

Theorem 1 (Nested property of HSCAN)**.** *Given two pairs of parameters ϵ_1,μ_1 and ϵ_2,μ_2, if $\epsilon_1 \leq \epsilon_2$ and $\mu_1 \leq \mu_2$, we have that for any cluster $C_{\epsilon_1,\mu_1} \in \mathcal{C}_{\epsilon_1,\mu_1}$, there exist a cluster $C_{\epsilon_2,\mu_2} \in \mathcal{C}_{\epsilon_2,\mu_2}$ such that $C_{\epsilon_2,\mu_2} \subseteq C_{\epsilon_1,\mu_1}$.*

Proof. The theorem directly follows the definitions of HSCAN. □

Example 2. *As shown in Fig. 2. Given two small parameters $\epsilon = 0.5$ and $\mu = 2$. With using HSCAN, we can obtain two clusters C_1 and C_2 in G and their corresponding cluster subgraphs are $G_{C_1} = \{\{v_1, v_2, v_3, v_4, v_5\}, \{e_1, e_2, e_3, e_4\}\}$ and $G_{C_2} = \{\{v_5, v_6, v_7, v_8, v_9\}, \{e_5, e_6, e_7\}\}$. C_1 and C_2 cannot form a large cluster together because the only connection between two clusters is $e_4 \leftrightarrow v_5 \leftrightarrow e_7$, and $\sigma(e_4, e_7) = \frac{1}{\sqrt{5 \cdot 5}} = 0.2 < 0.5$.*

B. Related Work

To the best of our knowledge, this paper is the first work focused on structural clustering on hypergraphs.

The concept of structural graph clustering and the fundamental model SCAN is first introduced by Xu et al. in [16]. Subsequent works have built upon this foundation to enhance efficiency and scalability. [50] proposes SCAN++ by reducing unnecessary similarity computations, while pSCAN [51] designs a new clustering paradigm to accelerate structural clustering. GPUSCAN [52] leverages GPU technology to implement SCAN more efficiently, and [53] presents an I/O-efficient algorithm designed to address the challenge of handling large graphs that cannot be fully loaded into main memory. Parallel and distributed approaches have also been explored, such as ppSCAN [54], which parallelizes pruning-based algorithms, and [55], which designs a distributed SCAN algorithm. Dong et al. [56] proposes an efficient index called GS*-index for SCAN.

Building on GS*-index, [57] presents a parallel algorithm based on the GS*-index and its approximation using locality-sensitive hashing. Ruan et al. [58] study the maintenance of computed structural similarity between vertices and propose an approximate algorithm for graph updates. Additionally, DSCAN [59] extends SCAN to directed graphs, while [60] explores structural clustering in uncertain graphs. Li et al. [61] prove the NP-hardness of manipulating SCAN and solve

Algorithm 1: GetSimNei_OI

```
Input  : A hyperedge $e$, a parameter $\epsilon$, a hypergraph
         $G = (V, E)$, and Neighbor-Order $NO$
Output: the set $S$ of $\epsilon$-neighbors of $e$
1 $S \leftarrow \emptyset$;
2 foreach $(e', \sigma(e, e'))$ in $NO_e$ do
3     if $\sigma(e, e') < \epsilon$ then
4         break;
5     $S \leftarrow S \cup \{e'\}$;
6 return $S$
```

Algorithm 2: GetCore_OI

```
Input  : Two parameters $\epsilon$ and $\mu$, a hypergraph
         $G = (V, E)$, and Core-Order $CO$
Output: Hash table $H$ of the set of core hyperedges
1 $H \leftarrow$ an empty hash table;
2 foreach $(e, \epsilon^{e,\mu}_{max})$ in $CO_\mu$ do
3     if $\epsilon^{e,\mu}_{max} < \epsilon$ then
4         break;
5     Add $e$ to $H$;
6 return $H$
```

it by identifying two key sub-problems: local promotion and global selection. Zhang et al. [62] introduce a bottom-k sketch to implement the approximate computation of Jaccard similarity, which can accelerate constructing and maintaining an index similar to the GS*-index. [63] extends the bottom-k sketch to d-hop neighbors to facilitate distance-based structural clustering.

There are also several other graph clustering works [64]–[72] on hypergraphs. However, they are not related to Structural Graph Clustering and cannot be utilized to process our HSCAN problem directly.

III. Structural Graph Clustering Indexes

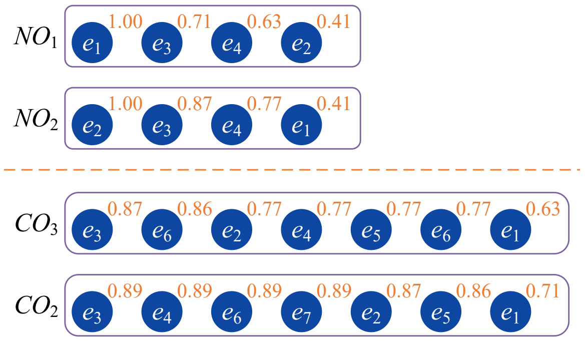
In this section, we first design the Order-Index (OI) to enhance performance. Then, to reduce the space and time cost of constructing the index, we further present the lightweight similarity-bucket index (LSBI).

A. Order-Index

Order-Index (OI) is extended from the best index GS*-index [56] of structural clustering pairwise graph. OI consists of Neighbor-Order (NO) and Core-Order (CO). NO is an adjacency table that stores (id, similarity) pairs instead of id. Moreover, for each $e \in E$, NO sorts its neighbors in descending order of similarity. NO is designed to fetch similar neighbors quickly. Differently, CO collects the max value of ϵ for each hyperedge $e \in E$ under each possible μ, such that $e \in G^{\epsilon,\mu}_{core}$. After that, CO stores the (id, ϵ) pairs in descending order of ϵ. Based on the above, the structure of OI is defined as follows:

Definition 10. *Given a hypergraph $G = (V, E)$, the Order-Index of G, denoted by OI, consists of Neighbor-Order and Core-Order. The Neighbor-Order for a hyperedge e, denoted by NO_e, contains all the pairs $(e', \sigma(e, e'))$. For two hyperedges $e_1, e_2 \in NO_e$, e_2 appears after e_1 if $\sigma(e, e_2) \leq \sigma(e, e_1)$. The Core-Order for a parameter μ, denoted by CO_μ, contains all the pairs $(e, \epsilon^{e,\mu}_{max})$ such that $\epsilon^{e,\mu}_{max} = max\{\epsilon | e \in \mathcal{G}^{\epsilon,\mu}_{core}\}$. For two hyperedges $e_1, e_2 \in NO_e$, e_2 appears before e_1 if $\epsilon^{e_1,\mu}_{max} \leq \epsilon^{e_2,\mu}_{max}$.*

Once Order-Index has been constructed, we can quickly obtain core hyperedges and similar neighbors via OI. The details for collecting ϵ-neighbors using OI are outlined in Algorithm 1. For a given hyperedge e and a parameter ϵ, we collects each pair $(e', \sigma(e, e'))$ from NO_e until $\sigma(e', e) < \epsilon$ (Lines 2-5). In addition, Algorithm 2 shows the pseudo-code



NO_1
e_1 1.00 e_3 0.71 e_4 0.63 e_2 0.41
NO_2
e_2 1.00 e_3 0.87 e_4 0.77 e_1 0.41
CO_3
e_3 0.87 e_6 0.86 e_2 0.77 e_4 0.77 e_5 0.77 e_6 0.77 e_1 0.63
CO_2
e_3 0.89 e_4 0.89 e_6 0.89 e_7 0.89 e_2 0.87 e_5 0.86 e_1 0.71

Fig. 3. Part of Order-Index (NO and CO) for hypergraph in Fig. 2(a).

of collecting core hyperedges. For two given parameters ϵ and μ, we iterate over each pair $(e, \epsilon^{e,\mu}_{max})$ of CO_μ and add e to a hash table when $\epsilon^{e,\mu}_{max} \geq \epsilon$ (Lines 2-5).

Theorem 2. *For a given hyperedge e and a parameter ϵ, the time complexity of Algorithm 1 is bounded by $O(|N_\epsilon[e]|)$. For two given parameters ϵ and μ, the time complexity of Algorithm 2 is bound by $O(|E(\mathcal{G}_C)|)$.*

Theorem 3. *For a given hypergraph G, the space complexity of Neighbor-Order is bounded by $O(\mathcal{M}(G))$, and the space complexity of Core-Order is also bounded by $O(\mathcal{M}(G))$.*

Example 3. *Fig. 3 shows a part of Order-Index (NO and CO) for hypergraph in Fig. 2(a). Given two parameters $\epsilon = 0.85$ and $\mu = 3$, we can obtain the set of ϵ-neighbors for hyperedge e_2 is $\{e_2, e_3\}$ from NO_2, and we can obtain the set of core hyperedges is $\{e_3, e_6\}$ from CO_3.*

B. Lightweight Similarity Bucket Index

To reduce the space cost of Order-Index, we propose a Lightweight Similarity Bucket Index: LSBI. LSBI comprises two sub-indexes: Similarity-Index (SI), and Core-Index (CI).

1) Similarity-Index

The Similarity-Index (SI) is primarily designed to obtain ϵ-neighbors of each hyperedge quickly. For each hyperedge $e \in E$, we pre-calculate its similarity to each neighbor, then sort these neighbors in descending order of similarity. We use τ buckets to store the sorted neighbors of e. These buckets divide the similarity scores into τ separate ranges. Each bucket represents a specific range of similarity scores, i.e., i^{th} corresponds to the range $[1 - \frac{i}{\tau}, 1 - \frac{i-1}{\tau})$. Each bucket stores the neighbors of e whose similarity with e falls within

Algorithm 3: GetSimNei_LSBI

```
Input : A hyperedge $e$, a parameter $\epsilon$, a hypergraph
        $G = (V, E)$, and the Similarity-Index $SI$
Output: the set $S$ of $\epsilon$-neighbors of $e$
1 $k \leftarrow \lceil (1-\epsilon) \cdot \tau \rceil$, $S \leftarrow \emptyset$;
2 if $\epsilon = 1 - \frac{k}{\tau}$ then
3     $S \leftarrow \bigcup_{i=1}^{k} SI_e[i]$;
4 else
5     $S \leftarrow \bigcup_{i=1}^{k-1} SI_e[i]$;
6 return $S$
```

the corresponding range. The hyperedges within a bucket are sorted in descending order by similarity score.

Definition 11 (Similarity-Index)**.** *Given a hypergraph $G = (V, E)$, the Similarity-Index of G, denoted by SI, consists of $|E|$ entries. The entry for a given hyperedge e, denoted by SI_e, consists of τ buckets. The i^{th} bucket in SI_e, denoted by $SI_e[i]$, contains all the hyperedges e' that satisfy $e' \in N[e]$ and $\sigma(e, e') \in [1-\frac{i}{\tau}, 1-\frac{i-1}{\tau})$. For two hyperedges $e_1, e_2 \in SI_e[i]$, e_2 appears after e_1 if $\sigma(e, e_2) \leq \sigma(e, e_1)$.*

Once SI has been constructed, obtaining the ϵ-neighbors of a specified hyperedge e for a given ϵ becomes straightforward. Moreover, SI can also expedite the identification of core hyperedges. The details for collecting ϵ-neighbors are outlined in Algorithm 3. Given the input parameters ϵ and μ, GetSimNei_LSBI algorithm firstly determines the bucket k such that ϵ falls within the range $[1-\frac{k}{\tau}, 1-\frac{k-1}{\tau})$ and initializes the set S as empty (Line 1). Then, we need to consider the following two cases:

Case 1: $\epsilon = 1 - \frac{k}{\tau}, k \in \{[1, \tau] \cap \mathbb{Z}\}$ (Lines 2-3). For a hyperedge e, all ϵ-neighbors are stored in $SI_e[i]$ for $i = 1, 2, \cdots, k$. Thus, it collects all the hyperedges from these buckets to obtain the ϵ-neighbors of e (Line 3). The hyperedge e is identified as a *core* hyperedge if $\sum_{i=1}^{k} |SI_e[i]| \geq \mu$.

Case 2: $\epsilon \in (1 - \frac{k}{\tau}, 1 - \frac{k-1}{\tau}), k \in \{[1, \tau] \cap \mathbb{Z}\}$ (Lines 4-5). For a hyperedge e, it is certain that all hyperedges stored in $SI_e[i]$ for $i = 1, 2, \cdots, k-1$ are ϵ-neighbors of e and the other ϵ-neighbors of e may only be stored in $SI_e[k]$. Then, it directly collects all the hyperedges from $SI_e[i]$ for $i = 1, 2, \cdots, k-1$ and e is identified as a core hyperedge if $\sum_{i=1}^{k-1} |SI_e[i]| \geq \mu$.

It is evident that we cannot obtain the exact clusters in Case 2. However, we can provide an approximation guarantee similar to that in [62] of the approximate solution for SCAN.

Theorem 4 (Approximation Guarantee for Approximate ϵ-neighbors)**.** *Given two parameters ϵ and μ, for the clusters $\mathcal{C}_{\epsilon,\mu}$ generated by executing HSCAN using approximate ϵ-neighbors, we have*

- *For every cluster $C_{\epsilon+\frac{1}{\tau},\mu} \in \mathcal{C}_{\epsilon+\frac{1}{\tau},\mu}$, there exists a cluster $C_{\epsilon,\mu} \in \mathcal{C}_{\epsilon,\mu}$ such that $C_{\epsilon+\frac{1}{\tau},\mu} \subseteq C_{\epsilon,\mu}$.*
- *For every cluster $C_{\epsilon-\frac{1}{\tau},\mu} \in \mathcal{C}_{\epsilon-\frac{1}{\tau},\mu}$, there exists a cluster $C_{\epsilon,\mu} \in \mathcal{C}_{\epsilon,\mu}$ such that $C_{\epsilon,\mu} \subseteq C_{\epsilon-\frac{1}{\tau},\mu}$.*

Proof. In Case 2, we directly get its approximate ϵ-neighbors of a hyperedge e from $SI_e[i]$ for $i = 1, 2, \cdots, k-1$. In fact, these hyperedges are e's $1 - \frac{k-1}{\tau}$-neighbors. Consequently, after executing the (ϵ, μ)-HSCAN, we actually obtain clusters $\mathcal{C}_{1-\frac{k-1}{\tau},\mu}$. Then we have $\epsilon - \frac{1}{\tau} < 1 - \frac{k-1}{\tau} < \epsilon + \frac{1}{\tau}$ because $\epsilon \in (1 - \frac{k}{\tau}, 1 - \frac{k-1}{\tau})$ in Case 2. Thus, according to Theorem 1, this theorem holds. □

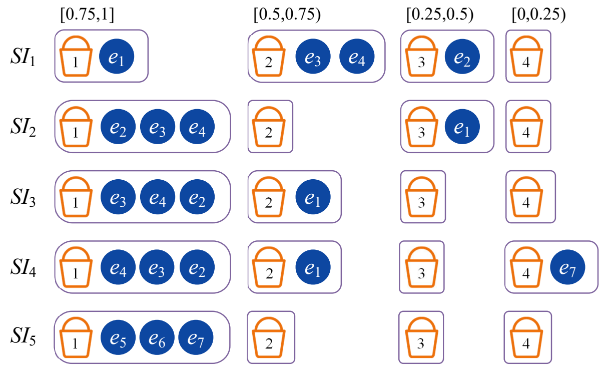


Fig. 4. Part of Similarity-Index (SI) for hypergraph in Fig. 2(a) with $\tau = 4$.

Theorem 5. *The time complexity of the GetSimNei_LSBI algorithm is bounded by $O(|N_\epsilon[e]|)$.*

Space complexity. The space cost of Similarity-Index (SI) is clearly given by $\sum_{i=1}^{|E|} N_\epsilon[e_i]$, which is bounded by $O(|\mathcal{M}(G)|)$, where $\mathcal{M}(G)$ is the set of structural neighbor pairs. Additionally, we provide the following guarantee regarding τ:

Theorem 6 (Space cost guarantee of Similarity-Index)**.** *The number of buckets in each entry of Similarity-Index, denoted by τ, is independent of the space complexity of Similarity-Index, i.e., the space cost of SI remains unchanged regardless of the value of τ.*

Example 4. *Given $\tau = 4$, Fig. 4 shows a part of the similarity index for the example graph in Fig. 2(a). Each entry of SI contains 4 buckets. Assume that $\epsilon = 0.75$ and $\mu = 2$ (Case 1), the ϵ-neighbors of e_3 are all hyperedges in $SI_3[1]$ and e_3 is a core hyperedge because $|SI_3[1]| = 3 > \mu$. Assume that $\epsilon = 0.7$ and $\mu = 2$ (Case 2), the hyperedges $\{e_4, e_3, e_2\}$ in $SI_4[1]$ are e_4's ϵ-neighbors and the hyperedges in $SI_4[3]$ and $SI_4[4]$ are not. The relationships between hyperedges in $SI_4[2]$ and e_4 need further detection.*

2) *Core-Index*

The Core-Index (CI) is designed to obtain the core hyperedges for HSCAN efficiently. Similar to SI, CI utilizes a series of buckets to divide the similarity score equally. To ensure that the same value of ϵ corresponds to the same bucket location in both the SI and CI structures, the number of buckets in the CI is kept as τ. However, CI does not maintain individual buckets for each hyperedge. Instead, it maintains a total of τ buckets. In the i^{th} bucket, which represents the similarity range $[1 - \frac{i}{\tau}, 1 - \frac{i-1}{\tau})$, CI stores an ordered set of $(e, |N_{1-\frac{i}{\tau}}[e]|)$ pairs, where e is a hyperedge and $|N_{1-\frac{i}{\tau}}[e]|$ is the number of its ϵ-neighbors with $\epsilon = 1 - \frac{i}{\tau}$. The pairs stored in each bucket are sorted in descending order according to $|N_{1-\frac{i}{\tau}}[e]|$. Considering the above, the structure of CI is defined as follows:

Algorithm 4: GetCore_LSBI

```
Input  : Two parameters $\epsilon$ and $\mu$, a hypergraph
         $G = (V, E)$, the Core-Index $CI$, and the
         Similarity-Index $SI$
Output: Hash table $H$ of the set of core hyperedges
1  $k \leftarrow \lceil(1-\epsilon)\cdot\tau\rceil$, $H \leftarrow$ an empty hash table;
2  if $\epsilon = 1 - \frac{k}{\tau}$ then
3      foreach $(e, cnt) \in CI_k$ do
4          if $cnt < \mu$ then break;
5          Add $e$ to $H$;
6  else
7      foreach $(e, cnt) \in CI_{k-1}$ do
8          if $cnt < \mu$ then break;
9          Add $e$ to $H$;
10 return $H$
```

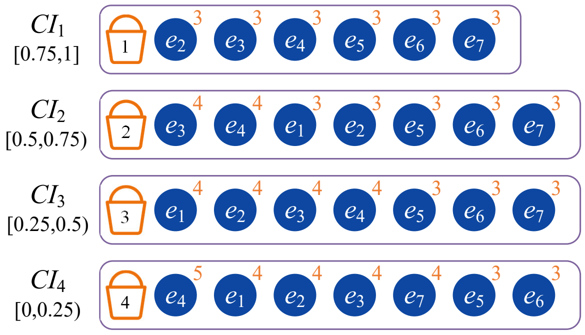


Fig. 5. Core-Index (CI) for hypergraph in Fig. 2(a) with $\tau = 4$.

Definition 12 (Core-Index). *Given a hypergraph $G = (V, E)$, the Core-Index of G, denoted by CI, consists of τ buckets. The i^{th} bucket in CI, denoted by CI_i, contains all the hyperedges e' that satisfy $|N_{1-\frac{i}{\tau}}[e]| \geq 2$. For two hyperedges $e_1, e_2 \in CI_i$, e_2 appears after e_1 if $|N_{1-\frac{i}{\tau}}[e_1]| \geq |N_{1-\frac{i}{\tau}}[e_2]|$.*

We can quickly obtain the core hyperedges using the CI. The details of collecting core hyperedges are shown in Algorithm 4. Given two input parameters, ϵ and μ, GetCore_LSBI algorithm first determines the bucket k such that ϵ falls within the range $[1-\frac{k}{\tau}, 1-\frac{k-1}{\tau})$ and initializes the hash table H as empty (Line 1). It then considers the following two cases:

Case 1: $\epsilon = 1 - \frac{k}{\tau}, k \in \{[1, \tau] \cap \mathbb{Z}\}$ (Lines 2-5). In this case, core hyperedges can be directly obtained from CI_k. Specifically, if the pair $(e, |N_{1-\frac{k}{\tau}}[e]|)$ in $CI_k[i]$ satisfies $|N_{1-\frac{k}{\tau}}[e]| < \mu$, then all the hyperedges stored in $CI_k[j]$ for $j = 1, 2, \cdots, i-1$ are identified as core hyperedges.

Case 2: $\epsilon \in (1-\frac{k}{\tau}, 1-\frac{k-1}{\tau}), k \in \{[1, \tau] \cap \mathbb{Z}\}$ (Lines 6-9). For a pair $(e, |N_{1-\frac{k}{\tau}}[e]|)$ stored in CI_k, we cannot ensure e is a core hyperedge, even if $|N_{1-\frac{k}{\tau}}[e]| \gg \mu$. However, for a pair $(e', |N_{1-\frac{k-1}{\tau}}[e']|)$ stored in CI_{k-1}, e' is certainly a core hyperedge if $|N_{1-\frac{k-1}{\tau}}[e']| \geq \mu$. Like collecting ϵ-neighbors, it directly collects the core hyperedges from CI_{k-1} and treats them as the entire core hyperedges. If the pair $(e, |N_{1-\frac{k-1}{\tau}}[e]|) \in CI_{k-1}$ satisfies $|N_{1-\frac{k-1}{\tau}}[e]| \geq \mu$, we identify e as a core hyperedges.

We can also provide an approximation guarantee for the approximate core hyperedges.

Theorem 7 (Approximation Guarantee for Approximate Core Hyperedges). *Given two parameters ϵ and μ, for the clusters $\mathcal{C}_{\epsilon,\mu}$ generated by executing HSCAN using approximate core, we can give the same approximation guarantee as that provided for approximate ϵ-neighbors.*

- *For every cluster $C_{\epsilon+\frac{1}{\tau},\mu} \in \mathcal{C}_{\epsilon+\frac{1}{\tau},\mu}$, there exists a cluster $C_{\epsilon,\mu} \in \mathcal{C}_{\epsilon,\mu}$ such that $C_{\epsilon+\frac{1}{\tau},\mu} \subseteq C_{\epsilon,\mu}$.*
- *For every cluster $C_{\epsilon-\frac{1}{\tau},\mu} \in \mathcal{C}_{\epsilon-\frac{1}{\tau},\mu}$, there exists a cluster $C_{\epsilon,\mu} \in \mathcal{C}_{\epsilon,\mu}$ such that $C_{\epsilon,\mu} \subseteq C_{\epsilon-\frac{1}{\tau},\mu}$.*

Proof. In Case 2, we directly obtain the approximate core hyperedges e from CI_{k-1} if $|N_{1-\frac{k-1}{\tau}}[e]| \geq \mu$. Essentially, we obtain the core hyperedges for $1 - \frac{k-1}{\tau}$ and μ. Since $\epsilon \in (1-\frac{k}{\tau}, 1-\frac{k-1}{\tau})$, it follows that $\epsilon - \frac{1}{\tau} \leq 1 - \frac{k-1}{\tau} \leq \epsilon + \frac{1}{\tau}$. Consequently, for any cluster $C_{\epsilon,\mu} \in \mathcal{C}_{\epsilon,\mu}$, there must exist two clusters $C_{\epsilon-\frac{1}{\tau},\mu} \in \mathcal{C}_{\epsilon-\frac{1}{\tau},\mu}$ and $C_{\epsilon+\frac{1}{\tau},\mu} \in \mathcal{C}_{\epsilon+\frac{1}{\tau},\mu}$ satisfy the following: (i) $G_{core}^{\epsilon+\frac{1}{\tau},\mu} \subseteq G_{core}^{\epsilon,\mu} \subseteq G_{core}^{\epsilon-\frac{1}{\tau},\mu}$ because $G_{core}^{\epsilon,\mu}$ is actually $G_{core}^{1-\frac{k-1}{\tau},\mu}$ and $\epsilon - \frac{1}{\tau} \leq 1 - \frac{k-1}{\tau} \leq \epsilon + \frac{1}{\tau}$. (ii) $G_C^{\epsilon+\frac{1}{\tau},\mu} \setminus G_{core}^{\epsilon+\frac{1}{\tau},\mu} \subseteq G_C^{\epsilon,\mu} \setminus G_{core}^{\epsilon,\mu} \subseteq G_C^{\epsilon-\frac{1}{\tau},\mu} \setminus G_{core}^{\epsilon-\frac{1}{\tau},\mu}$, because any non-core hyperedge in $G_C^{\epsilon,\mu}$ is an ϵ-neighbor of one of the core hyperedges and $\epsilon - \frac{1}{\tau} < \epsilon < \epsilon + \frac{1}{\tau}$. Thus, we can conclude that $G_C^{\epsilon+\frac{1}{\tau},\mu} \subseteq G_C^{\epsilon,\mu} \subseteq G_C^{\epsilon-\frac{1}{\tau},\mu}$, thereby proving this theorem. □

Theorem 8. *The time complexity of the GetCore_LSBI algorithm is $O(|E(\mathcal{G}_{\mathcal{C}})|)$.*

Space complexity. The space consumption of CI depends on the number of buckets and the number of hyperedges.

Theorem 9. *The space cost of Core-Index is $O(\tau \cdot |E|)$.*

Example 5. *Given $\tau = 4$, the Core-Index for the example hypergraph in Fig. 2(a) is shown in Fig. 5. The CI contains four buckets, each representing a different range of similarity scores, e.g., bucket CI_1, which corresponds to the similarity range $[0.75, 1]$, includes all the pairs $(e, |N_{0.75}[e]|)$. Assuming $\epsilon = 0.5$ and $\mu = 4$ (Case 1), we can easily obtain the set of core hyperedges $\{e_3, e_4\}$ from the CI_2. Assume that $\epsilon = 0.4$ and $\mu = 4$ (Case 2), it is clear that e_3 and e_4 are core hyperedges because $|N_{0.5}[e_3]| = 4 \geq \mu$ and $|N_{0.5}[e_4]| = 4 \geq \mu$. Additionally, only e_1 and e_2 have the potential to be identified as core hyperedges, as $|N_{0.25}[e_i]| = 3 < \mu$ ($i = 5, 6, 7$).*

C. Remark.

Effect of the similarity bucket. Suppose that the float type is used to store the similarity score. The technique of similarity bucket affects space overhead from two aspects: (1) CO stores core hyperedges of different μ and sorts them by ϵ. CI is designed based on similarity buckets, stores core hyperedges according to ϵ, and sorts them by μ. For a given hypergraph $G = (V, E)$, CO stores $\sum_{e \in E} |N[e]| - 1 = 2 \cdot |\mathcal{M}|$ pairs which need $16 \cdot 2 \cdot |\mathcal{M}|$ bytes, and CI stores $\tau \cdot |E|$ pairs which need $16 \cdot \tau \cdot |E|$ bytes. If $\tau < \frac{2 \cdot |\mathcal{M}|}{|E|}$, CI requires less space than CO.

(2) NO stores (id, similarity score) pairs, while SI only stores ids in similarity buckets. Thus, the similarity bucket technique on SI reduces the space overhead by half.

Relationships between our indexes and existing indexes. OI is directly extended by GS*-index [56] by changing the stored element from vertices to hyperedges. LSBI is motivated by BOTIN-index [62], and they both utilize the similarity bucket technique. BOTIN-index additionally uses bottom-k-sketch to obtain approximate Jaccard similarity, while LSBI not only uses similarity buckets to store information about core hyperedges but also uses similarity buckets to store information about neighbors and similarity scores (this part of BOTIN-index is the same as neighbor-order of GS*-index). Thus, LSBI can further reduce the space cost without reducing the accuracy of the approximate guarantee and the query efficiency.

Efficiency of queries using two indexes. It is noted that the performance of the query with OI and the query with LSBI is almost the same. Due to the space limitation, we provide the comparison in our technical report [73].

Exact and approximate queries using LSBI. LSBI collects approximate ϵ-neighbors and core hyperedges by default in Case 2, while it also supports fetching exact ϵ-neighbors and core hyperedges in Case 2, which requires additional time for detection. We compare the clustering results of approximate query and exact query in Exp-9, Section VI.D. The result indicates that the approximate query can obtain similar clustering results. Moreover, due to space limitations, the process of exact query and the efficiency comparison between approximate query and exact query are in our technical report.

IV. Index-based Query Algorithms

In this section, we present two query algorithms for HSCAN based on any index that speeds up core hyperedges and similar neighbor fetching, e.g., OI (Section III-A) and LSBI (Section III).

A. Sequential Query Algorithm

In this subsection, we propose the sequential query algorithm (SQuery).

Algorithm 5 provides the pseudo-core of the SQuery. Given the input parameters ϵ and μ, along with the index I and the hypergraph G, the algorithm first initializes the clustering result $\mathcal{C}$, the labels of each hyperedge $Label$, and a queue Q as empty(Line 1). It then retrieves all the core hyperedges stored in a hash table. Next, for each unlabeled core hyperedges e, it initializes a new cluster $C = \{e\}$ to record the cluster containing e and inserts e into queue Q (Line 4). It assigns a new label l to the new cluster C and labels the hyperedges in C with l (Line 5). Then, it iterates over the hyperedges in Q to expand the cluster C using a BFS process, where only hyperedges that are both core hyperedges and neighbors of the hyperedges in Q are added into the Q (Lines 6-14). Specifically, it firstly removes a hyperedge e_1 from queue Q and collects its ϵ-neighbors(Lines 7-8). Then, it iterates over each hyperedge e_2 in e_1's ϵ-neighbors, adding e_2 into C and labeling e_2 with l (Lines 10-12). If e_2 is a core hyperedge, it

Algorithm 5: Sequential query algorithm: SQuery

```
Input  : Parameters $\epsilon$ and $\mu$, an Index $I$, and
         hypergraph $G = (V, E)$
Output: Clustering result and labels of each hyperedge
1  $\mathcal{C} \leftarrow \emptyset$, $Q \leftarrow$ an empty queue, $Label \leftarrow$ an empty set;
2  $H \leftarrow$GetCore($\epsilon$,$\mu$,$G$,$I$);
3  foreach unlabeled $e \in H$ do
4      $C \leftarrow \{e\}$, $Q.inqueue(e)$;
5      Assign a new label $l$ to $C$ and label $e$ with $l$;
6      while $Q$ is not empty do
7          $e_1 \leftarrow Q.dequeue()$;
8          $N_\epsilon[e_1] \leftarrow$ GetSimNei($e_1$,$\epsilon$,$G$,$I$);
9          foreach $e_2 \in N_\epsilon[e_1]$ do
10             if $Label[e_2]$ is null then
11                 $C \leftarrow C \cup \{e_2\}$;
12                 Label $e_2$ with $l$;
13                 if $e_2 \in H$ then
14                     $Q.inqueue(e_2)$;
15     $\mathcal{C} \leftarrow \mathcal{C} \cup C$;
16 foreach $e \in E \setminus H$ do
17     if $e$ is unlabeled then
18         Label $e$ as hub or outlier according to its
             neighbor and Definition 9 ;
19 return $\mathcal{C}$, $Label$
```

inserts e_2 into Q (Lines 13-14). The BFS iteration continues until the queue Q is empty (Line 6). The completed cluster C is then added to the $\mathcal{C}$ (Line 15). Finally, SQuery identifies the hubs and outliers and returns the results (Lines 16-19).

Theorem 10. *The time complexity of SQuery is $O(|\mathcal{M}(G)|)$ with using OI or LSBI, where $\mathcal{M}(G)$ is the set of neighboring hyperedges of G.*

Due to space limitations, the proofs of Theorem 10 and the following theorems can be found in our technical report [73].

Example 6. *Consider the example hypergraph shown in Fig. 2(a), along with the Similarity-Index in Fig. 4 and the Core-Index in Fig. 5. With $\epsilon = 0.75$ and $\mu = 3$, we can obtain the set of core hyperedges $\{e_2, e_3, e_4, e_5, e_6, e_7\}$ using Algorithm 4. Starting with e_2, we expand a cluster C and add an edge e_2 into Q. Subsequently, edges e_3 and e_4 are added to C and Q because they are both e_2's ϵ-neighbors and core hyperedges. Since there is no hyperedge that is the ϵ-neighbors of e_3 or e_4, it has C=$\{e_2, e_3, e_4\}$. Similarly, we obtain the remaining cluster $\{e_5, e_6, e_7\}$. Finally, the edge e_1 is identified as an outlier hyperedge, and the entire query process finishes.*

B. Parallel Query Algorithm

In this subsection, we propose the parallel query algorithm (PQuery). Unlike SQuery, PQuery separates the process of clustering Core hyperedges from the process of clustering non-core hyperedges for better parallelism. The pseudo-code of the parallel query algorithm is shown in Algorithm 6. First,

Algorithm 6: Parallel query algorithm: PQuery

```
Input  : Parameters $\epsilon$ and $\mu$, an Index $I$, and
         hypergraph $G = (V, E)$
Output: Clustering result and labels of each hyperedge
1  $\mathcal{C} \leftarrow \emptyset$, $H \leftarrow$GetCore($\epsilon$,$\mu$,$G$,$I$);
2  $Label \leftarrow \{e \in E | Label[e] = e\}$;
3  foreach $e \in H$ in parallel do
4      $N_\epsilon[e] \leftarrow$GetSimNei($e_1$,$\epsilon$,$G$,$I$);
5      foreach $e' \in N_\epsilon[e]$ in parallel do
6          if $e'.id < e.id$ and $e' \in H$ then
7              $CC_e \leftarrow Label.path\_compress\_find(e)$;
8              $CC_{e'} \leftarrow Label.path\_compress\_find(e')$;
9              $Label.union(CC_e, CC_{e'})$;
10 foreach $e \in H$ in parallel do
11     $P_e \leftarrow Label.path\_compress\_find(e)$;
12 foreach $e \in H$ in parallel do
13     foreach $e' \in N_\epsilon[e]$ in parallel do
14         if $e' \notin H$ then
15             $Label.union(e', e)$ ;
16 foreach $e \in E \setminus H$ in parallel do
17     if $e$ is unlabeled then
18         Label $e$ as hub or outlier according to its
           neighbor and Definition 9 ;
19 return $\mathcal{C}$, $Label$
```

PQuery initializes the clustering result $\mathcal{C}$ as an empty set and stores all core hyperedges into a hash table H (Line 1). Then, it initializes the labels of hyperedges as a union-find disjoint set [74], an efficient data structure to find the connected component to which a vertex belongs and merge different connected components (Line 2). Lines 3-11 perform clustering core hyperedges. For each core hyperedge e, PQuery iterates over hyperedge e' that is both a core hyperedge e' and a ϵ-neighbor of e (Lines 3-6). To prevent duplicate calculation, PQuery skips the hyperedge e' with a larger id than e (Line 6). Next, it finds the connected components CC_e and $CC_{e'}$ to which e and e' belong, respectively (Lines 7-8). Then, PQuery merges two connected components (Line 9). Lines 10-11 utilize the path_compress_find to ensure that hyperedges within the same connected component use the same label. After clustering all core hyperedges, PQuery assigns non-core hyperedges to clusters (Lines 12-15) and classifies the remaining hyperedges as a hub or outlier (Lines 16-18).

Theorem 11. *For a single thread, the time complexity of PQuery is $O((\mathcal{M}(G)+|E|)\cdot\alpha(|E|))$, where $\alpha(\cdot)$ is the inverse Ackermann function and $\mathcal{M}(G)$ is the set of neighboring hyperedges of G.*

V. Index Construction

In this section, we propose the algorithms to construct two indexes presented in Section III.

Algorithm 7: OI-Construction

```
Input  : A hypergraph $G = (V, E)$
Output: The index OI for $G$
1  $NO \leftarrow \emptyset$, $CO \leftarrow \emptyset$ ;
2  foreach hyperedge $e \in E$ do
3      $N[e] \leftarrow \{e' | e' \in \bigcup_{v \in V(e)} E(v)\}$, $cnt = 0$;
4      foreach $e' \in N[e]$ do
5          Compute $\sigma(e, e')$ using hash table;
6          Add pair $(e', \sigma(e, e'))$ into sorted $NO_e$;
7      foreach $\mu \in [2, |N[e]|]$ do
8          $\epsilon_{max}^{e,\mu} \leftarrow NO_e[\mu].\sigma(e, e')$;
9          Add pair $(e, \epsilon_{max}^{e,\mu})$ into sorted $CO_\mu$;
10 return $OI(NO, CO)$
```

A. Similarity Computation

Before the construction of indexes, we need to obtain the pairs of neighboring hyperedges and compute the similarity score between hyperedges. The state-of-the-art structural clustering algorithm pSCAN [51] utilizes a sort-and-merged-based set intersection approach for similarity computations and only computes the similarity score once for a pair of neighbors. These techniques can be applied to our work, but we introduce an alternative method for computing similarity. For each hyperedge e, its neighbor set is given by $\{e' | e' \in \bigcup_{v \in V(e)} E(v)\}$. For each neighboring hyperedge pair (e_1, e_2), we compute $\sigma(e_1, e_2)$ according to Definition 1, and use the temporary hash table to accelerate the similarity computation. In particular, we hash the vertices in $V(e_1)$ and look up the vertices of $V(e_2)$ in the temporary hash table when computing the similarity between e_1 and e_2. Thus, a single similarity computation is bounded by $O(|V(e_1)| + |V(e_2)|)$.

B. OI construction algorithm

The pseudo-code for constructing Order-Index is shown in Algorithm 7. OI-Construction initializes NO and CO as empty (Line 1). Then, for each hyperedge e, OI-Construction iterates each of its neighbors, computes the structural similarity score, and adds them into NO_e (Lines 4-6). Next, for each hyperedge e, OI-Construction iterates each possible μ, sets $\epsilon_{max}^{e,\mu}$ as the $\sigma(e, e')$ stored in $NO_e[\mu]$, and inserts the pair $(e, \epsilon_{max}^{e,\mu})$ into CO_μ (Lines 7-9).

Theorem 12. *The time complexity of OI-construction is bounded by $O(\mathcal{M}(G)\cdot(|V|+\log|E|^2))$, where $\mathcal{M}(G)$ is the set of neighboring hyperedges of G.*

C. LSBI construction algorithm

Algorithm 8 presents the pseudo-code for the LSBI-Construction algorithm. Given a hypergraph $G = (V, E)$, LSBI-construction first initializes SI and CI as empty sets (Line 1). Lines 2-13 construct both the Similarity-Index and Core-Index. For each $e \in E$, LSBI-construction fetches its neighbors (Lines 2-3). Then, for each e's neighbor e', the similarity between them is computed only if e has a smaller id than e', thereby preventing duplicate calculations (Line 5). The similarity score is then calculated (Line 6). Subsequently,

Algorithm 8: LSBI-Construction

```
Input  : A hypergraph $G = (V, E)$
Output: The index LSBI for $G$
1  $SI \leftarrow \emptyset$, $CI \leftarrow \emptyset$ ;
2  foreach hyperedge $e \in E$ do
3      $N[e] \leftarrow \{e' | e' \in \bigcup_{v \in V(e)} E(v)\}$, $cnt = 0$;
4      foreach $e' \in N[e]$ do
5          if $e.id < e'.id$ then
6              Compute $\sigma(e, e')$ using hash table;
7              $k \leftarrow \lceil (1 - \sigma(e, e')) \cdot \tau \rceil$;
8              Add $e'$ into sorted set $SI_e[k]$;
9              Add $e$ into sorted set $SI_{e'}[k]$;
10     for $i$ from $1$ to $\tau$ do
11         $cnt = cnt + |SI_e[i]|$;
12         if $cnt < 2$ then continue;
13         Add $(e, cnt)$ into sorted set $CI_i$;
14 return LSBI(SI,CI)
```

e' is added into k^{th} bucket in SI_e, and e is added into k^{th} bucket in $SI_{e'}$, where k is determined based on the similarity score (lines 7-9). Once SI_e has been constructed, Lines 10-13 insert the pairs $(e, |N_{1-\frac{i}{\tau}}[e]|)$ into i^{th} bucket in CI for $i = 1, 2, \cdots, \tau$.

Theorem 13. *The time complexity of LSBI-Construction algorithm (Algorithm 8) is bounded by $O(|E| \cdot \tau \cdot \log |E| + |\mathcal{M}(G)| \cdot (\log |E| + |V|))$, where $\mathcal{M}(G)$ is the set of neighboring hyperedges of G.*

VI. Experiments

A. Experimental Settings

Datasets. We use seven real hypergraphs in the experiment. All the real hypergraphs are collected from [8], [47]. Additionally, the synthetic dataset rpah is generated by the Random-Preferential-Attachment-Hypergraph model [75]. The details of all datasets are shown in Table II.

Competitors. Our empirical studies are conducted against the following designs:

- *Clustering quality.* We compare the clustering result of HSCAN with two prior structural clustering models: SCAN [16] and WSCAN [76]. Referring to the previous works [77]–[79], the measure is Modularity [70].
- *Query performance.* We primarily compare the following three methods: (i) pHSCAN: the hypergraph-extended version of the state-of-the-art exact structural clustering algorithm [51], (ii) SQuery: the sequential query algorithm in Section IV-A, and (iii) PQuery: the parallel query algorithm in Section IV-B.
- *Index cost.* We primarily compare OI and LSBI with the following aspects: (i) Space cost, (ii) Construction cost, (iii) Considering that the query from LSBI is approximate, referring to the previous works [62], [63], we use Adjusted Rand Index (ARI) [80] to compare approximate query from LSBI and exact query from OI.

TABLE II
Datasets: $|V|$#nodes, $|E|$#hyperedges, $\overline{d(v)}$#avg. degree of a node, $\overline{|V(e)|}$#avg. cardinality of a hyperedge.

Graph	Type	$\|V\|$	$\|E\|$	$\overline{d(v)}$	$\overline{\|V(e)\|}$
enron	employee-email	4,423	5,734	6.8	5.2
contact	person-clique	242	12,704	127	2.4
congress	person-bill	1,718	83.105	426.2	8.8
drug	NDCcode-substance	5,311	112,405	39.1	1.9
ubuntu	users-posts	125,602	192,947	2.8	1.8
dblp	author-publication	1,836,596	2,170,260	4	3.4
aminer	author-publication	27,850,748	17,120,546	2.3	3.7
rpah	Synthetic	91,002,133	104,003,058	3.4	3

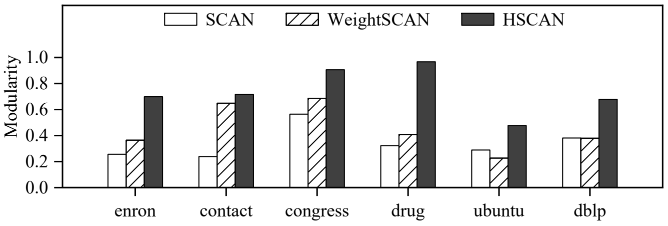


Fig. 6. Maximum modularity score.

- *Other hypergraph clustering models.* We compare our structural clustering model HSCAN with a classic hypergraph clustering model LOUV [71] and the state-of-the-art hypergraph clustering model PIC [72].

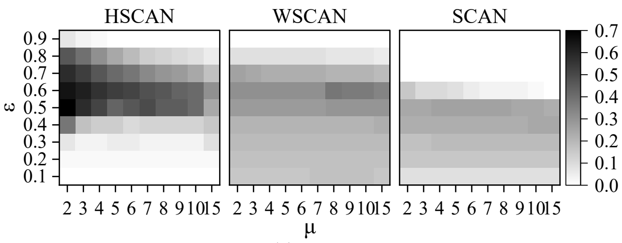
Environment and parameters. The code and datasets are available at https://github.com/pardon-hnu/Hyper-SCAN. We store all the graphs in memory and implement the algorithms in C++. The experiments are conducted on a server with an Intel(R) Xeon(R) Gold 5218R CPU @ 2.10GHz and 255 GB memory. The server has 2 physical CPUs, and each CPU contains 20 cores. The number of buckets, τ, is set to 10 by default.

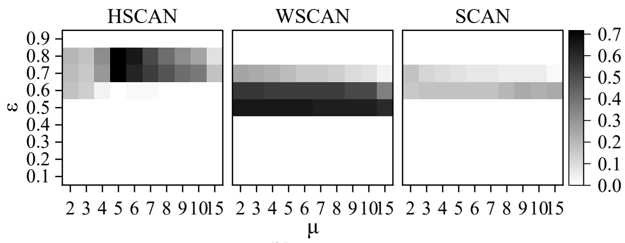
Remark. Note that we only show results on partial datasets in some experiments (Exp-2, Exp-4, Exp-5, and Exp-8) due to space limitations. Other results are similar to results shown in this paper and can be found in our technical report [73].

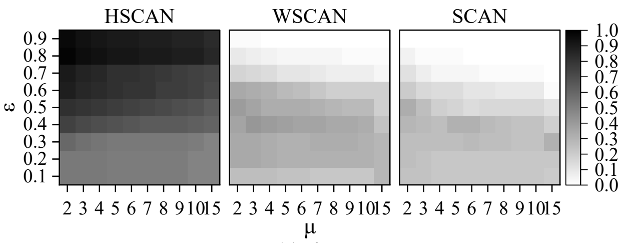
B. Clustering Quality

Referring to the previous works [63], [77]–[79], we use modularity, a key measure for evaluating clustering quality, to compare the clustering results between HSCAN and SCAN on hypergraphs. Modularity does not require comparison with the ground truth, making it widely applicable. The computation of modularity for hypergraphs is referred to [70], and it represents the quality of vertex clusters of the given hypergraph. Each vertex cluster of HSCAN is $\bigcup_{e \in C} V(e)$ directly obtained by its corresponding hyperedge set C. Additionally, we set $\epsilon \in \{0.1, 0.2, \cdots, 0.9\}$ and $\epsilon \in \{2, 3, \cdots, 9, 10, 15\}$.

Exp-1: Maximum Modularity. For the aminer and rpah datasets, we generated clustering results under various parameters, but computing the modularity measures and the ARI in Exp-10 took over 24 hours. Consequently, we excluded these two hypergraphs from the evaluation and used the remaining datasets. Fig. 6 shows the maximum modularity score of the clustering results, where the maximum modularity score means the best modularity of clustering results under varying parameters. The results indicate that HSCAN achieves a higher maximum modularity score than SCAN and WSCAN,


(a) enron.


(b) contact.


(c) drug.

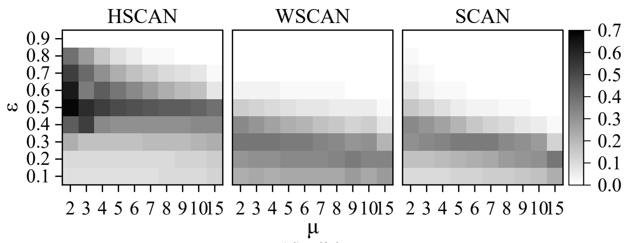

(d) dblp.

Fig. 7. Two-dimensional heatmap (measure: modularity score).

suggesting its superior suitability for even clustering vertices in hypergraphs.

Exp-2: Tuning parameters. In this experiment, we evaluate the impact of ϵ and τ on the modularity of the clustering results. Fig. 7 is a two-dimensional heatmap of each dataset that shows modularity scores under all parameter pairs. As shown in Fig. 7, we have the following observations: (1) it is clear that the impact of ϵ is more significant than τ. This is because ϵ directly affects both the compactness within clusters and the looseness between clusters. (2) The modularity scores show a trend of increasing first and then decreasing with increasing ϵ. The trend is particularly evident in some figures (e.g., Fig. 7(a) and Fig. 7(d)). This is likely due to the existence of an optimal ϵ for each hypergraph. When ϵ is much smaller than it, dissimilar vertices and hyperedges are included in clusters, leading to poor clustering quality. When ϵ is much larger than it, there are too few vertices and hyperedges within the clusters, and the clustering quality is also poor. (3) The modularity scores also show a trend of increasing first and then decreasing with increasing μ. The trend is particularly evident in some figures (e.g., Fig. 7(b) and Fig. 7(d)). This is likely due to the existence of an optimal μ for each hypergraph, and the optimal μ is 2 in many hypergraphs. When μ is smaller than it, the compactness within the clusters decreases. When μ is larger than it, the clusters have fewer vertices and hyperedges, and the clustering quality deteriorates. (4) HSCAN does not outperform SCAN and WSCAN under any pair of parameters. This is because HSCAN majors in hyperedges, and the optimal ϵ mentioned in (2) and the optimal μ mentioned in (3) of the three models are not the same in most cases. Additionally, according to the knee method, we find the ridge or knee-like area in the heatmap and obtain the recommended parameters ranges: $\epsilon \in [0.5, 0.8]$ and $\mu \in [2, 5]$ for HSCAN.

C. Query Performance

Referring to the previous work [56], we set $\epsilon \in \{0.2, 0.3, \cdots, 0.8\}$ and $\mu \in \{2, 5, 10, 15\}$. The default values of ϵ and μ are 0.6 and 5, respectively.

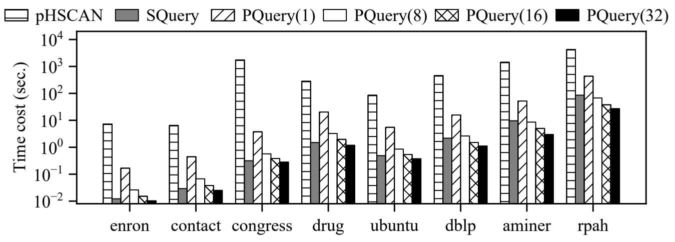


Fig. 8. Average query time on all datasets.

Exp-3: Average query time on all datasets. We compute the average results to evaluate query efficiency. We set the number of threads t of PQuery as $\{1, 8, 16, 32\}$ and denote this as PQuery(t). Fig. 8 reports the average running time of pHSCAN, SQuery, and PQuery with different numbers of threads. SQuery is up to three orders of magnitude faster than pHSCAN. The running time of PQuery decreases with increasing number of threads. PQuery(1) is slower than SQuery. PQuery(8) is up to 1.27× faster than SQuery and PQuery(16) is up to 2.27× faster than SQuery on large datasets such as aminer, while they are not greater than SQuery on small datasets. PQuery(32) is up to 3.2× faster than SQuery. To summarize, PQuery is more suitable for large hypergraphs, but it cannot offer a great improvement compared to SQuery in small hypergraphs.

This phenomenon occurs for the following reasons: (1) To implement and improve the parallelism, PQuery introduces extra computations such as checking labels (Lines 10-11 in Algorithm 5) and path_compress_find (Lines 7-8 in Algorithm 5). Thus, PQuery has a higher time complexity than SQuery. (2) Some unavoidable atomic operations in PQuery affect parallelism, such as the Union operation. (3) PQuery needs more cores to achieve higher parallelism and thus gain performance beyond SQuery. Additionally, PQuery performs better in large hypergraphs where the extra computation is a lower percentage of the total computation.

Exp-4: Impact of ϵ. In this experiment, we evaluate the

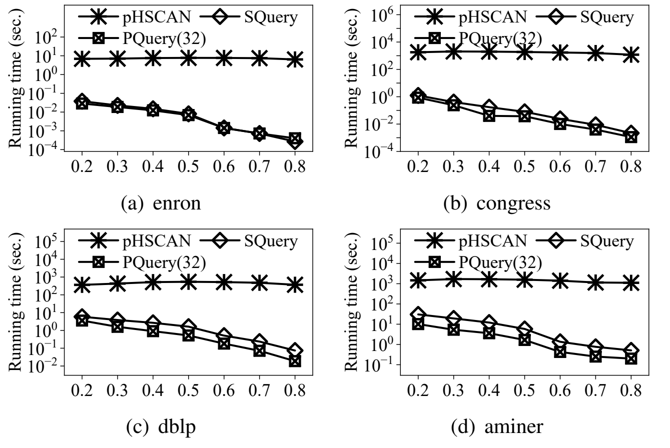


Fig. 9. Impact of ϵ: query time.

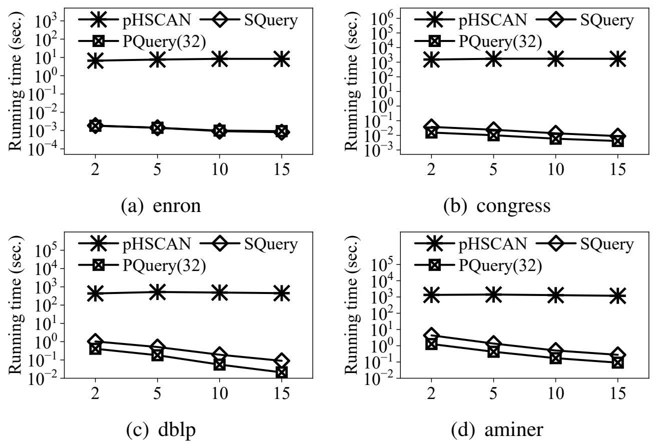


Fig. 10. Impact of μ: query time.

impact of ϵ on query time. We use three algorithms: pHSCAN, SQuery, and PQuery(32). As shown in Fig. 9, the query time of SQuery and PQuery(32) decreases as ϵ grows. This is because the clusters have fewer hyperedges when ϵ increases. In addition, the performance gap between PQuery(32) and SQuery is minimally affected by epsilon and more strongly affected by hypergraph size.

Exp-5: Impact of μ. In this experiment, we evaluate the impact of μ on query time. We also use three algorithms: pHSCAN, SQuery, and PQuery(32). Since there is no clustering for rpah under the default ϵ, we set ϵ to 0.3 for the rpah dataset. The result is shown in Fig. 10. As μ increases, the query times for SQuery and PQuery decrease, but not as significantly as the decrease observed when ϵ increases. This is because a small increase in μ does not significantly reduce the number of edges in the clusters. Like ϵ, the performance gap between PQuery(32) and SQuery is minimally affected by epsilon and strongly affected by hypergraph size.

D. Index Cost

Exp-6: Space cost of index. In this experiment, we compare the space cost of LSBI to that of OI. Fig. 11 shows the space cost of these two indexes, indicating that LSBI requires only $25\% \sim 31\%$ of the space needed by OI. Moreover, both CI and SI cost less space than NO and CO, respectively, which indicates the great effectiveness of the similarity bucket technique. Additionally, the space overhead ratio of LSBI to

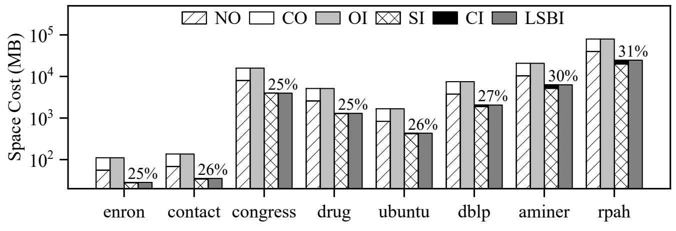


Fig. 11. Space cost of each index.

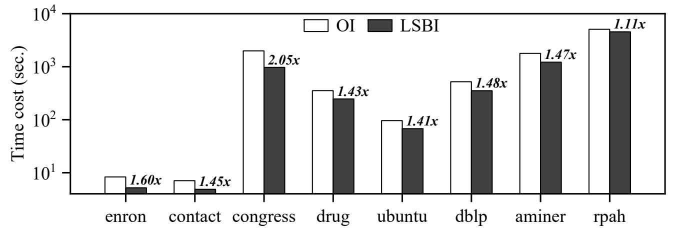


Fig. 12. Construction cost of each index.

OI decreases on sparser hypergraphs (e.g., dblp, aminer, and rpah). This is because $\tau \cdot |E|$ is closer to $\mathcal{M}(G)$ on these sparse hypergraphs.

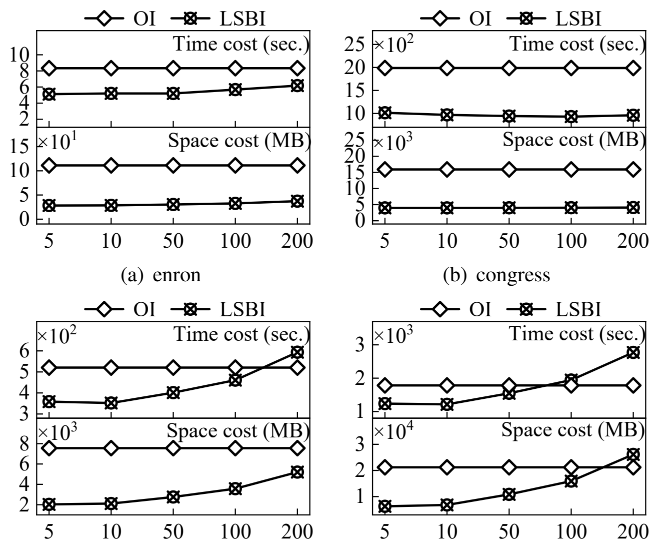
Exp-7: Construction cost of index. In this experiment, we evaluate the cost of constructing OI and LSBI. Fig. 12 reports the running time of OI-Construction and LSBI-Construction. We observe that LSBI-Construction can be up to 2.05× faster than OI-Construction, and the increase in running time is affected by both increasing hypergraph size and density.

Exp-8: Impact of τ. In this experiment, we evaluate the impact of τ on index cost. Fig. 13 shows the construction and space costs of OI and LSBI on each dataset with varying $\tau \in \{5, 10, 50, 100, 200\}$. The cost of OI is constant because it is independent of τ, and the missing data on rpah is due to memory limitations. The space and construction overhead of LSBI increase with τ, but very slowly for small hypergraphs. This is because the only hypergraph-related parameter affected by τ in the space and construction complexity is $|E|$. LSBI has a lower index cost than OI in most cases, but there is a risk that the index cost of LSBI exceeds that of OI when $\tau > 100$. This indicates that we can set τ in the range $[10, 100]$ considering the approximate guarantee and the index cost.

Exp-9: Evaluate the approximate query. In this experiment, we use Adjusted Rand Index (ARI) [80] with exact results serving as the ground truth to compare the approximate query from LSBI and the exact query from OI. We set $\epsilon \in \{0.25, 0.35, 0.45, 0.55, 0.65, 0.75\}$ for approximate query and evaluate the ARI score. Fig. 14 reports the max, mean, and median values of ARI scores. This illustrates that the clustering results obtained from the LSBI-based approximate query are almost identical to those from the OI-based exact query. When we need to reduce the index cost, we can use LSBI to obtain consistent query performance and similar clustering results.

E. Comparison with Other Models

In this subsection, we compare our structural hypergraph clustering model HSCAN with two modularity-based hypergraph clustering models: LOUV [71] (code provided by [81]) and PIC [72]. LOUV is a classic model, and PIC is the state-of-the-art hypergraph clustering model. Table III shows the



(a) enron (b) congress (c) dblp (d) aminer

Fig. 13. Impact of τ: index cost.

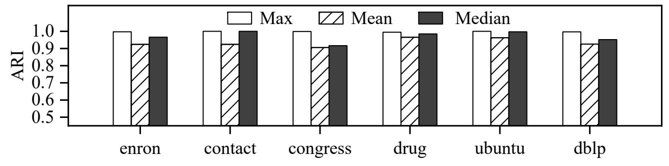
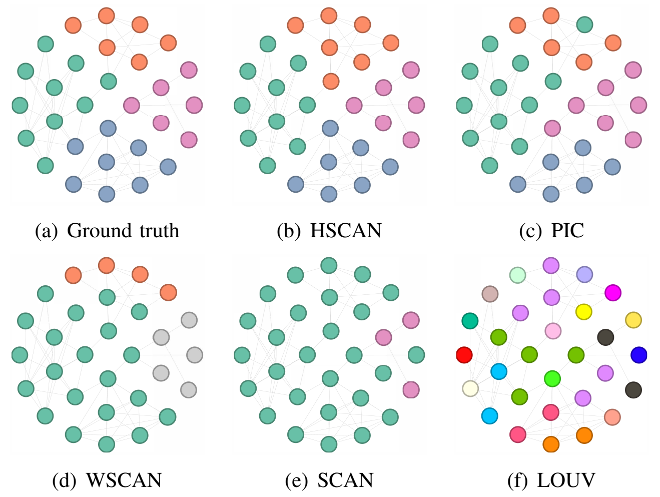


Fig. 14. ARI.

TABLE III
COMPARISON BETWEEN HSCAN, LOUV, AND PIC.

Dataset	Modularity			Clustering Time (sec.)		
	LOUV	PIC	HSCAN	LOUV	PIC	HSCAN
enron	0.48	**<u>0.76</u>**	0.70	0.41	0.07	**<u>0.01</u>**
contact	0.59	**<u>0.80</u>**	0.72	0.07	0.36	**<u>0.02</u>**
congress	0.47	0.59	**<u>0.91</u>**	4.09	1.17	**<u>0.32</u>**
drug	0.71	0.78	**<u>0.97</u>**	1.04	**<u>0.21</u>**	1.49
ubuntu	0.21	**<u>0.91</u>**	0.48	3.66	1.31	**<u>0.49</u>**
dblp	0.05	**<u>0.78</u>**	0.68	81.39	53.00	**<u>2.17</u>**
aminer	\	\	\	1005.08	810.985	**<u>9.59</u>**
rpah	\	\	\	5968.92	\	**<u>85.85</u>**



(a) Ground truth (b) HSCAN (c) PIC (d) WSCAN (e) SCAN (f) LOUV

Fig. 15. Case study

comparison between three models in modularity and time cost. '\' denotes no results due to time out (over 24 hours) in clustering or modularity computation. The top-1 score for each dataset is highlighted with bold&underline. On modularity, HSCAN and PIC perform better than each other on different datasets, but both are better than LOUV. On clustering efficiency, HSCAN outperforms other models on all datasets except for the drug, and HSCAN can be up to $85\times$ faster than PIC. Thus, HSCAN delivers high-quality clustering results in a short amount of time.

F. Case Study

In this subsection, we provide a case study based on a real dataset with ground-truth MAG. MAG is downloaded from [67], and collected from Microsoft-Academic-Graph [82]. Each node in MAG represents an author, and each hyperedge in MAG represents a paper with some coauthors who published this paper together in a conference. We cluster it with HSCAN and other compared clustering models in this paper and show the results on 30 vertices in Fig. 15 due to the limited space. We use the clustering results with the best modularity for structural clustering models (HSCAN, SCAN, and WSCAN). There are four ground-truth clusters, as shown in Fig. 15(a), corresponding to four conferences: *STOC* (in green), *VLDB* (in orange), *ICML* (in blue), and *CVPR* (in violet). Each author belongs to one cluster, which is determined by the conference in which he/she has published the most papers. There is a line between two vertices if they share at least one hyperedge.

We observe that HSCAN produces the most accurate clusters among all models, and PIC is close to HSCAN. WSCAN and SCAN struggle to effectively differentiate between distinct groups. LOUV generates many small-sized clusters. This is because both HSCAN and PIC fully consider the unique structure of the hypergraph, i.e., HSCAN obtains clusters centered on the hyperedges, while PIC designs a new model based on the relevance of the hyperedges.

VII. CONCLUSION

In this paper, we present a new and effective structural clustering model for hypergraphs, along with a comprehensive index-based approach for the new model, verified with extensive experiments. The index-based approach concludes with two indexes, two query algorithms, and construction algorithms for two indexes. In further work, we will investigate FPGA-based parallel algorithms for HSCAN to further accelerate the query.

ACKNOWLEDGMENT

Xu Zhou and Quanqing Xu are the corresponding authors of the research. The research was supported by the Creative Research Groups Program of the National Natural Science Foundation of China (Grant No. 62321003), the National Natural Science Foundation of China (Grant Nos. U23A20317, 62172146), the Natural Science Foundation of Hunan Province (Grant No. 2023JJ10016), the Key R&D Program of Hunan Province (Grant No. 2023GK2002), and the Ant Group through CCF-Ant Research Fund (Grant No. CCF-AFSG RF20230101).

References

[1] C. Berge, *Hypergraphs - combinatorics of finite sets*, ser. North-Holland mathematical library. North-Holland, 1989, vol. 45.

[2] A. Bretto, "Hypergraph theory," *An introduction. Mathematical Engineering. Cham: Springer*, vol. 1, 2013.

[3] W. Zhang, Z. Yang, D. Wen, W. Li, W. Zhang, and X. Lin, "Accelerating core decomposition in billion-scale hypergraphs," *Proc. ACM Manag. Data*, vol. 3, no. 1, pp. 6:1–6:27, 2025.

[4] Y. Feng, H. You, Z. Zhang, R. Ji, and Y. Gao, "Hypergraph neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 3558–3565.

[5] X. Ouvrard, J.-M. L. Goff, and S. Marchand-Maillet, "Networks of collaborations: Hypergraph modeling and visualisation," *arXiv preprint arXiv:1707.00115*, 2017.

[6] E. Ramadan, A. Tarafdar, and A. Pothen, "A hypergraph model for the yeast protein complex network," in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.* IEEE, 2004, p. 189.

[7] Q. Luo, D. Yu, Z. Cai, X. Lin, G. Wang, and X. Cheng, "Toward maintenance of hypercores in large-scale dynamic hypergraphs," *The VLDB Journal*, vol. 32, no. 3, pp. 647–664, 2023.

[8] N. A. Arafat, A. Khan, A. K. Rai, and B. Ghosh, "Neighborhood-based hypergraph core decomposition," *Proc. VLDB Endow.*, vol. 16, no. 9, pp. 2061–2074, 2023.

[9] Z. Yang, W. Zhang, X. Lin, Y. Zhang, and S. Li, "Hgmatch: A match-by-hyperedge approach for subgraph matching on hypergraphs," in *2023 IEEE 39th International Conference on Data Engineering (ICDE).* IEEE, 2023, pp. 2063–2076.

[10] Q. Luo, D. Yu, Y. Liu, Y. Zheng, X. Cheng, and X. Lin, "Finer-grained engagement in hypergraphs," in *2023 IEEE 39th International Conference on Data Engineering (ICDE).* IEEE, 2023, pp. 423–435.

[11] Q. Luo, W. Zhang, Z. Yang, D. Wen, X. Wang, D. Yu, and X. Lin, "Hierarchical structure construction on hypergraphs," in *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management, CIKM 2024, Boise, ID, USA, October 21-25, 2024*, E. Serra and F. Spezzano, Eds. ACM, 2024, pp. 1597–1606.

[12] B. Yang, D. Wen, L. Qin, Y. Zhang, L. Chang, and R. Li, "Index-based optimal algorithm for computing k-cores in large uncertain graphs," in *35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019.* IEEE, 2019, pp. 64–75.

[13] D. Wen, B. Yang, L. Qin, Y. Zhang, L. Chang, and R. Li, "Computing k-cores in large uncertain graphs: An index-based optimal approach," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 7, pp. 3126–3138, 2022.

[14] T. Zhang, X. Cai, L. Chen, Z. Yang, Y. Gao, B. Cao, and J. Fan, "Towards efficient simulation-based constrained temporal graph pattern matching," *World Wide Web (WWW)*, vol. 27, no. 3, p. 22, 2024.

[15] T. Zhang, Y. Gao, J. Zhao, L. Chen, L. Jin, Z. Yang, B. Cao, and J. Fan, "Efficient exact and approximate betweenness centrality computation for temporal graphs," in *Proceedings of the ACM on Web Conference 2024, WWW 2024, Singapore, May 13-17, 2024.* ACM, 2024, pp. 2395–2406.

[16] X. Xu, N. Yuruk, Z. Feng, and T. A. Schweiger, "Scan: a structural clustering algorithm for networks," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2007, pp. 824–833.

[17] G. Salton, "Automatic text processing: The transformation, analysis, and retrieval of," *Reading: Addison-Wesley*, vol. 169, 1989.

[18] P. Jaccard, "Distribution de la flore alpine dans le bassin des dranses et dans quelques régions voisines," *Bull Soc Vaudoise Sci Nat*, vol. 37, pp. 241–272, 1901.

[19] 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, pp. 1–12, 2012.

[20] 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," *BMC Bioinform.*, vol. 12, no. S-10, p. S7, 2011.

[21] Z. Liu, Q. Shi, D. Ding, R. Kelly, H. Fang, and W. Tong, "Translating clinical findings into knowledge in drug safety evaluation-drug induced liver injury prediction system (dilips)," *PLoS computational biology*, vol. 7, no. 12, p. e1002310, 2011.

[22] M. Mete, F. Tang, X. Xu, and N. Yuruk, "A structural approach for finding functional modules from large biological networks," in *Bmc Bioinformatics*, vol. 9. Springer, 2008, pp. 1–14.

[23] 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 the 5th ACM on International Conference on Multimedia Retrieval*, 2015, pp. 203–210.

[24] 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 the 23rd ACM international conference on Multimedia*, 2015, pp. 189–192.

[25] S. Papadopoulos, Y. Kompatsiaris, A. Vakali, and P. Spyridonos, "Community detection in social media: Performance and application considerations," *Data mining and knowledge discovery*, vol. 24, pp. 515–554, 2012.

[26] S. Lim, S. Ryu, S. Kwon, K. Jung, and J.-G. Lee, "Linkscan*: Overlapping community detection using the link-space transformation," in *2014 IEEE 30th international conference on data engineering.* IEEE, 2014, pp. 292–303.

[27] C. X. Lin, Y. Yu, J. Han, and B. Liu, "Hierarchical web-page clustering via in-page and cross-page link structures," in *Advances in Knowledge Discovery and Data Mining: 14th Pacific-Asia Conference, PAKDD 2010, Hyderabad, India, June 21-24, 2010. Proceedings. Part II 14.* Springer, 2010, pp. 222–229.

[28] Y. Qiu, D. Wen, R. Li, L. Qin, M. Yu, and X. Lin, "Computing significant cliques in large labeled networks," *IEEE Trans. Big Data*, vol. 9, no. 3, pp. 904–917, 2023.

[29] S. Papadopoulos, Y. Kompatsiaris, and A. Vakali, "A graph-based clustering scheme for identifying related tags in folksonomies," in *Data Warehousing and Knowledge Discovery: 12th International Conference, DAWAK 2010, Bilbao, Spain, August/September 2010. Proceedings 12.* Springer, 2010, pp. 65–76.

[30] S. Papadopoulos, C. Zigkolis, Y. Kompatsiaris, and A. Vakali, "Cluster-based landmark and event detection for tagged photo collections," *IEEE Multimedia Magazine*, vol. 18, no. 1, pp. 52–63, 2011.

[31] S. Papadopoulos, C. Zigkolis, G. Tolias, Y. Kalantidis, P. Mylonas, Y. Kompatsiaris, and A. Vakali, "Image clustering through community detection on hybrid image similarity graphs," in *2010 IEEE International Conference on Image Processing.* IEEE, 2010, pp. 2353–2356.

[32] S. S. Chawathe, "Clustering blockchain data," *Clustering Methods for Big Data Analytics: Techniques, Toolboxes and Applications*, pp. 43–72, 2019.

[33] P. Domingos and M. Richardson, "Mining the network value of customers," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, 2001, pp. 57–66.

[34] Y. Wang, D. Chakrabarti, C. Wang, and C. Faloutsos, "Epidemic spreading in real networks: An eigenvalue viewpoint," in *22nd International Symposium on Reliable Distributed Systems, 2003. Proceedings.* IEEE, 2003, pp. 25–34.

[35] U. Kang and C. Faloutsos, "Beyond'caveman communities': Hubs and spokes for graph compression and mining," in *2011 IEEE 11th international conference on data mining.* IEEE, 2011, pp. 300–309.

[36] B. Perozzi, L. Akoglu, P. I. Sánchez, and E. Müller, "Focused clustering and outlier detection in large attributed graphs," in *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014.* ACM, 2014, pp. 1346–1355.

[37] S. Song, F. Gao, R. Huang, and Y. Wang, "On saving outliers for better clustering over noisy data," in *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021.* ACM, 2021, pp. 1692–1704.

[38] C. Böhm, C. Faloutsos, and C. Plant, "Outlier-robust clustering using independent components," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008.* ACM, 2008, pp. 185–198.

[39] L. Biabani, A. Hennes, M. Monemizadeh, and M. Schmidt, "Faster query times for fully dynamic k-center clustering with outliers," in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[40] L. Meng, L. Yuan, Z. Chen, X. Lin, and S. Yang, "Index-based structural clustering on directed graphs," in *2022 IEEE 38th International Conference on Data Engineering (ICDE).* IEEE, 2022, pp. 2831–2844.

[41] D. Yu, D. Wang, Q. Luo, Y. yi, G. Wang, and Z. Cai, "Stable structural clustering in uncertain graphs," *Inf. Sci.*, vol. 586, pp. 596–610, 2022.

[42] J. Howie, V. Srinivasan, and A. Thomo, "Scaling up structural clustering to large probabilistic graphs using lyapunov central limit theorem," *Proc. VLDB Endow.*, vol. 16, no. 11, pp. 3165–3177, 2023.

[43] S. Laenen and H. Sun, "Higher-order spectral clustering of directed graphs," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020.

[44] Y. Chen, R. Chen, Q. Li, X. Fang, J. Li, and W. K. Wong, "Denoising high-order graph clustering," in *40th IEEE International Conference on Data Engineering, ICDE 2024, Utrecht, The Netherlands, May 13-16, 2024*. IEEE, 2024, pp. 3111–3124. [Online]. Available: https://doi.org/10.1109/ICDE60146.2024.00241

[45] Q. Li, H. Liu, R. Jiang, and T. Wang, "Incorporating higher-order structural information for graph clustering," in *Database Systems for Advanced Applications - 29th International Conference, DASFAA 2024, Gifu, Japan, July 2-5, 2024, Proceedings, Part IV*, ser. Lecture Notes in Computer Science, vol. 14853. Springer, 2024, pp. 507–517.

[46] Z. Lin, Z. Kang, L. Zhang, and L. Tian, "Multi-view attributed graph clustering," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 2, pp. 1872–1880, 2023.

[47] A. R. Benson, R. Abebe, M. T. Schaub, A. Jadbabaie, and J. Kleinberg, "Simplicial closure and higher-order link prediction," *Proceedings of the National Academy of Sciences*, vol. 115, no. 48, pp. E11 221–E11 230, 2018.

[48] J. Ah-Pine and G. Jacquet, "Clique-based clustering for improving named entity recognition systems," in *EACL 2009, 12th Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference, Athens, Greece, March 30 - April 3, 2009*, A. Lascarides, C. Gardent, and J. Nivre, Eds. The Association for Computer Linguistics, 2009, pp. 51–59.

[49] H. Zhang, M. Fiszman, D. Shin, B. Wilkowski, and T. C. Rindflesch, "Clustering cliques for graph-based summarization of the biomedical research literature," *BMC Bioinform.*, vol. 14, p. 182, 2013.

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

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

[52] T. R. Stovall, S. Kockara, and R. Avci, "Gpuscan: Gpu-based parallel structural clustering algorithm for networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 12, pp. 3381–3393, 2014.

[53] J. H. Seo and M. H. Kim, "pm-scan: an i/o efficient structural clustering algorithm for large-scale graphs," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017, pp. 2295–2298.

[54] Y. Che, S. Sun, and Q. Luo, "Parallelizing pruning-based graph structural clustering," in *Proceedings of the 47th International Conference on Parallel Processing*, 2018, pp. 1–10.

[55] K. Hao, L. Yuan, Z. Yang, W. Zhang, and X. Lin, "Efficient and scalable distributed graph structural clustering at billion scale," in *Database Systems for Advanced Applications - 28th International Conference, DASFAA 2023, Tianjin, China, April 17-20, 2023, Proceedings, Part III*, vol. 13945. Springer, 2023, pp. 234–251.

[56] D. Wen, L. Qin, Y. Zhang, L. Chang, and X. Lin, "Efficient structural graph clustering: an index-based approach," *The VLDB Journal*, vol. 28, no. 3, pp. 377–399, 2019.

[57] T. Tseng, L. Dhulipala, and J. Shun, "Parallel index-based structural graph clustering and its approximation," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 1851–1864.

[58] B. Ruan, J. Gan, H. Wu, and A. Wirth, "Dynamic structural clustering on graphs," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 1491–1503.

[59] L. Meng, L. Yuan, Z. Chen, X. Lin, and S. Yang, "Index-based structural clustering on directed graphs," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2022, pp. 2831–2844.

[60] D. Yu, D. Wang, Q. Luo, Y. Zheng, G. Wang, and Z. Cai, "Stable structural clustering in uncertain graphs," *Information Sciences*, vol. 586, pp. 596–610, 2022.

[61] W. Li, M. Gao, D. Wen, H. Zhou, C. Ke, and L. Qin, "Manipulating structural graph clustering," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2022, pp. 2749–2761.

[62] F. Zhang and S. Wang, "Effective indexing for dynamic structural graph clustering," *Proceedings of the VLDB Endowment*, vol. 15, no. 11, pp. 2908–2920, 2022.

[63] K. Liu, S. Wang, Y. Zhang, and C. Xing, "An efficient algorithm for distance-based structural graph clustering," *Proceedings of the ACM on Management of Data*, vol. 1, no. 1, pp. 1–25, 2023.

[64] J. J. Whang, R. Du, S. Jung, G. Lee, B. L. Drake, Q. Liu, S. Kang, and H. Park, "MEGA: multi-view semi-supervised clustering of hypergraphs," *Proc. VLDB Endow.*, vol. 13, no. 5, pp. 698–711, 2020.

[65] Y. Takai, A. Miyauchi, M. Ikeda, and Y. Yoshida, "Hypergraph clustering based on pagerank," in *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*. ACM, 2020, pp. 1970–1978.

[66] N. Veldt, A. Wirth, and D. F. Gleich, "Parameterized correlation clustering in hypergraphs and bipartite graphs," in *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*. ACM, 2020, pp. 1868–1876.

[67] I. Amburg, N. Veldt, and A. R. Benson, "Clustering in graphs and hypergraphs with categorical edge labels," in *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*. ACM / IW3C2, 2020, pp. 706–717.

[68] C. J. Zhu, Q. Liu, and J. Bi, "Communication efficient distributed hypergraph clustering," in *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*. ACM, 2021, pp. 2131–2135.

[69] H. Zhong, Y. Zhang, C. Yan, Z. Xuan, T. Yu, J. Zhang, S. Ying, and Y. Gao, "Penalized flow hypergraph local clustering," *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 5, pp. 2110–2125, 2024.

[70] P. S. Chodrow, N. Veldt, and A. R. Benson, "Generative hypergraph clustering: From blockmodels to modularity," *Science Advances*, vol. 7, no. 28, p. eabh1303, 2021.

[71] S. Agarwal, J. Lim, L. Zelnik-Manor, P. Perona, D. J. Kriegman, and S. J. Belongie, "Beyond pairwise clustering," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA*. IEEE Computer Society, 2005, pp. 838–845.

[72] Z. Feng, M. Qiao, and H. Cheng, "Modularity-based hypergraph clustering: Random hypergraph model, hyperedge-cluster relation, and computation," *Proc. ACM Manag. Data*, vol. 1, no. 3, pp. 215:1–215:25, 2023. [Online]. Available: https://doi.org/10.1145/3617335

[73] "Source code and datasets," 2024. [Online]. Available: https://github.com/pardon-hnu/Hyper-SCAN

[74] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.

[75] C. Avin, Z. Lotker, Y. Nahum, and D. Peleg, "Random preferential attachment hypergraph," in *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, 2019, pp. 398–405.

[76] J. Huang, H. Sun, Q. Song, H. Deng, and J. Han, "Revealing density-based clustering structure from the core-connected tree of a network," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 8, pp. 1876–1889, 2013.

[77] C. Wu, Y. Gu, and G. Yu, "Dpscan: Structural graph clustering based on density peaks," in *International Conference on Database Systems for Advanced Applications*. Springer, 2019, pp. 626–641.

[78] Y. Qiu, R. Li, J. Li, S. Qiao, G. Wang, J. X. Yu, and R. Mao, "Efficient structural clustering on probabilistic graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 10, pp. 1954–1968, 2018.

[79] D. Yu, D. Wang, Q. Luo, Y. Zheng, G. Wang, and Z. Cai, "Stable structural clustering in uncertain graphs," *Inf. Sci.*, vol. 586, no. C, p. 596–610, Mar. 2022.

[80] L. Hubert and P. Arabie, "Comparing partitions," *Journal of classification*, vol. 2, pp. 193–218, 1985.

[81] T. Bonald, N. de Lara, Q. Lutz, and B. Charpentier, "Scikit-network: Graph analysis in python," *J. Mach. Learn. Res.*, vol. 21, pp. 185:1–185:6, 2020.

[82] "Microsoft academic graph," 2024. [Online]. Available: https://www.microsoft.com/en-us/research/project/microsoft-academic-graph/