Community Detection in Heterogeneous Information Networks Without Materialization

JIAXIN JIANG, National University of Singapore, Singapore
SIYUAN YAO, National University of Singapore, Singapore
YUHANG CHEN, National University of Singapore, Singapore
BINGSHENG HE, National University of Singapore, Singapore
YUDONG NIU, Singapore Management University, Singapore
YUCHEN LI, Singapore Management University, Singapore
SHIXUAN SUN, Shanghai Jiao Tong University, China
YONGCHAO LIU, Ant Group, China

Community detection in heterogeneous information networks (HINs) poses significant challenges due to the diversity of entity types and the complexity of their interrelations. While traditional algorithms may perform adequately in some scenarios, many struggle with the high memory usage and computational demands of large-scale HINs. To address these challenges, we introduce a novel framework, SCAR, which efficiently uncovers community structures in HINs without requiring network materialization. SCAR leverages insights from meta-paths to interpret multi-relational data through compact vertex-based sketches, significantly reducing computational overhead and materialization overhead. We propose a sketch-based technique for estimating changes in modularity, improving both the precision and speed in community detection. Our extensive evaluations on diverse real-world datasets provide detailed comparative metrics, demonstrating that SCAR outperforms several state-of-the-art methods, including Gdy, Louvain, Leiden, Infomap, Walktrap, and Networkit, in execution time and memory consumption while maintaining competitive accuracy. Overall, SCAR offers a robust and scalable solution for revealing community structures in large HINs, with applications across various domains, including social networks, academic collaboration networks, and e-commerce platforms.

CCS Concepts: • **Theory of computation → Graph algorithms analysis**; **Approximation algorithms analysis**; • **Mathematics of computing → Probabilistic algorithms**.

Additional Key Words and Phrases: Graph Algorithms, Community Detection

ACM Reference Format:
Jiaxin Jiang, Siyuan Yao, Yuhang Chen, Bingsheng He, Yudong Niu, Yuchen Li, Shixuan Sun, and Yongchao Liu. 2025. Community Detection in Heterogeneous Information Networks Without Materialization. *Proc. ACM Manag. Data* 3, 3 (SIGMOD), Article 139 (June 2025), 27 pages. https://doi.org/10.1145/3725276

1 Introduction

Graphs are fundamental structures in numerous applications, including social networks [17, 48, 59], communication networks [2, 4, 27, 37], and academic citation networks [20, 23]. One fundamental

Authors' Contact Information: Jiaxin Jiang, National University of Singapore, Singapore, jxjiang@nus.edu.sg; Siyuan Yao, National University of Singapore, Singapore, siyuan.y@u.nus.edu; Yuhang Chen, National University of Singapore, Singapore, yuhangc@comp.nus.edu.sg; Bingsheng He, National University of Singapore, Singapore, hebs@comp.nus.edu.sg; Yudong Niu, Singapore Management University, Singapore, ydniu@smu.edu.sg; Yuchen Li, Singapore Management University, Singapore, yuchenli@smu.edu.sg; Shixuan Sun, Shanghai Jiao Tong University, China, sunshixuan@sjtu.edu.cn; Yongchao Liu, Ant Group, China, yongchao.ly@antgroup.com.

This work is licensed under a Creative Commons Attribution International 4.0 License.

© 2025 Copyright held by the owner/author(s).
ACM 2836-6573/2025/6-ART139
https://doi.org/10.1145/3725276

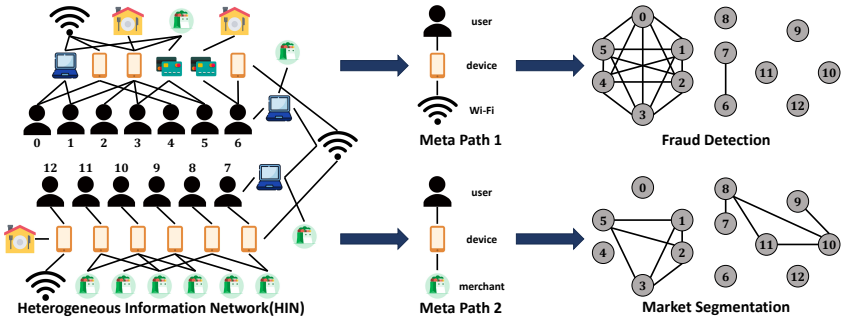

Fig. 1. Ant Group's data pipelines for community detection.

challenge is *community detection*—the task of uncovering groups of nodes that exhibit significantly denser internal connections compared to their links with the rest of the network [44, 45, 64]. Detecting such communities reveals underlying structural insights and enables practical applications, including uncovering collaboration patterns [34, 62], tracking the spread of information or diseases [31], enhancing targeted marketing strategies [8, 22, 24], identifying filter bubbles [31], and detecting fraudulent activities [17, 24].

In many real-world settings, networks exhibit heterogeneous characteristics [33, 46, 51, 53], featuring multiple node types and complex, multi-relational link structures. This inherent complexity makes it challenging to directly apply traditional community-detection techniques. Consequently, many existing approaches rely on meta-path-based graph materialization or advanced modeling methods to capture the rich semantics present in HINs [33, 51, 53].

Motivating Applications. HINs play a critical role in various domains. For example, on e-commerce platforms, nodes can represent vendors, products, or customers, while edges encode relationships such as product recommendations, customer reviews, and purchase histories—ultimately forming a complex network that lends itself to diverse real-world analytical scenarios:

(1) **Financial Fraud Detection.** In financial transaction networks, nodes often represent individuals or accounts linked by transactions or shared account usage. As shown in Figure 1, the meta-path "User → Device → Wi-Fi" can be used to construct a user-to-user relationship network, capturing cases where different users access the same Wi-Fi network. Community detection then helps identify potential fraudulent groups; for instance, a large cluster of suspicious users connecting through the same Wi-Fi forms a community, indicating possible fraud.

(2) **Market Segmentation.** The meta-path "User → Device → Merchant" can represent users who purchase goods from the same merchant. Such meta-paths facilitate community detection, enabling user segmentation for recommendations (e.g., users in the same community likely share similar interests). Moreover, community detection can reveal vendor communities that collaborate or engage in unethical activities—such as price-fixing or misleading advertisements—thereby aiding in fair trading enforcement and maintaining marketplace integrity.

Case Study: Community Detection at Ant Group. As a leading technology enterprise offering services like digital payments, Ant Group utilizes community detection to analyze its vast network of entities, including users, merchants, drivers, devices, and transactions. Figure 1 illustrates its data pipeline for community detection, involving: (1) Converting data into an HIN, where different node types correspond to various entities. (2) Defining *meta-paths* based on business needs to capture specific relational patterns. (3) Using these meta-paths to construct homogeneous graphs. (4) Performing community detection on these homogeneous graphs to identify community structures.

Challenges in Large-Scale HINs. Applying community detection to large-scale HINs is challenging due to diverse entity types and complex relational structures. Traditional algorithms [7, 16, 35, 54, 61] often struggle with the computational and memory demands of materializing homogeneous graphs from such diverse data sources. Specifically, Ant Group faces the following challenges:

(1) **Graph Materialization Overhead.** Up to **99.0%** of the processing time is spent materializing homogeneous graphs via meta-paths, creating a significant bottleneck. For instance, on FreeBase—a comprehensive general-purpose HIN—it takes over 1,000 seconds on average.
(2) **High Memory Usage.** Materialization requires substantial memory, with space complexity reaching $O(|V|^2)$, which hampers scalability for large graphs. In our industrial dataset, AG, which comprises 135 million nodes, 1.81 billion edges, and 37 distinct node types, the materializations result in out-of-memory errors.
(3) **Diverse Business Requirements.** The variability in business scenarios necessitates constructing different homogeneous graphs for different meta-paths, which is impractical due to the associated computational and storage costs.

Our Approach. To address these challenges, we propose SCAR (**S**ketch-based **C**ommunity detection **A**lgo**R**ithms for Heterogeneous Information Networks), a novel framework for efficient community detection in HINs without explicit graph materialization. Our main contributions are as follows:

(1) **Unmaterialized Framework.** We introduce a novel framework that computes community structures from HINs and meta-paths, avoiding the materialization of homogeneous graphs.
(2) **Sketch-Based Algorithms with Accuracy Guarantees.** We design sketch-based techniques that approximate modularity changes using neighbor cardinality estimation, as well as union and intersection estimation, with provable accuracy guarantees. This enables reliable community detection while significantly reducing computational overhead.
(3) **Extensive Empirical Validation.** Experiments on real-world datasets demonstrate that SCAR achieves up to **99.0%** faster execution and **93.6%** lower memory usage compared to state-of-the-art methods, while maintaining high modularity that reflects robust community structures.

Paper Organization. Section 2 provides background on community detection and HINs. Section 3 details the sketch-based community detection algorithms. Section 4 describes techniques to enhance efficiency and effectiveness. Section 5 reports the experimental evaluation of SCAR. Section 6 reviews related work and Section 7 concludes the paper and suggests directions for future research.

2 Background and Problem Statement

In this section, we provide the essential background, introduce key notations (see Table 1), and define the problem we address.

2.1 Preliminary

A *heterogeneous information network (HIN)* is modeled as a directed graph $G = (V, E, L)$, where the labeling function L assigns each node and edge a type. In other words, $L : V \to \mathcal{A}$ maps each node to a node type in the set $\mathcal{A}$, and $L : E \to \mathcal{R}$ maps each edge to an edge type in the set $\mathcal{R}$. This heterogeneous structure is the basis for encoding rich semantics in many real-world datasets.

One widely adopted technique to extract useful semantic information from HINs is the concept of *meta-paths*. Introduced in [51], a meta-path P is a sequence of node and edge types, such as

$$P = A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \dots \xrightarrow{R_n} A_{n+1},$$

which specifies a schema-level relationship. Here n represents the number of edge types in the meta-path, and each $R_i \in \mathcal{R}$ denotes a specific edge type. Concrete paths following P in G are called *matching instances*. This approach is widely utilized in various studies [14, 30, 47, 52, 58, 63].

Table 1. Frequently used notations.

Notation	Meaning
$G = (V, E, L)$	Heterogeneous information network
A_i/R_i	Node type/Edge type
P	Meta-path
M	Matching instance of a meta-path
$G_P = (V_P, E_P)$	Induced homogeneous graph from meta-path P
$N_{G_P}(u)$	the neighbors of u in G_P
k_{ij}	Number of the edges between nodes u_i and u_j
k_i	Total number of edges incident to node u_i
C_i	Community to which node u_i belongs
Q	Modularity of the network

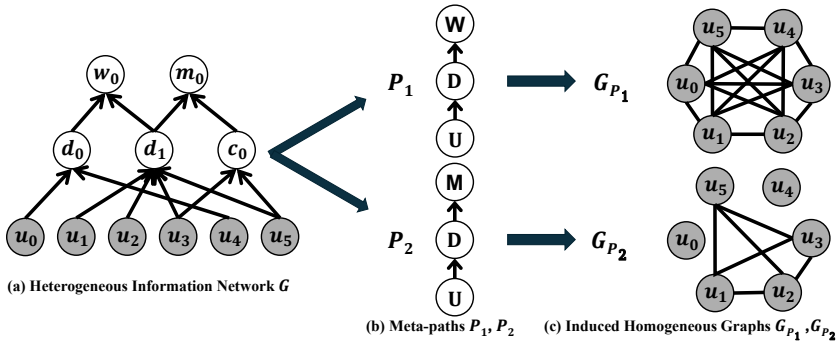

Fig. 2. A HIN, two meta-paths, and their corresponding induced homogeneous graphs.

Definition 2.1 (Matching Instance). *Given a HIN G and a meta-path P, a* matching instance *M of P, denoted $M \triangleright P$, is a sequence of nodes and edges $M = u_1 \xrightarrow{e_1} u_2 \xrightarrow{e_2} \dots \xrightarrow{e_n} u_{n+1}$ such that:*

- *$L(u_i) = A_i$ for $1 \le i \le n+1$;*
- *Each edge e_i connects u_i to u_{i+1} with $L(e_i) = R_i$.*

Traditional community detection algorithms are designed for homogeneous graphs, so a common strategy is to transform the HIN into an *induced homogeneous graph*. The idea is to focus on one node type and connect nodes that share a meta-path-defined relationship.

Definition 2.2. *For a given HIN G and a meta-path P, the* induced homogeneous graph *$G_P = (V_P, E_P)$ is constructed as follows:*

- *V_P contains all nodes in V of type A_1 that participate in at least one matching instance $M \triangleright P$;*
- *An edge (u, u') exists in E_P iff there exists at least one node w such that there are matching instances M and M' of P starting at u and u', respectively, and both ending at w (i.e., $M(u_{n+1}) = M'(u_{n+1}) = w$).*

This transformation allows us to apply community detection algorithms designed for homogeneous graphs to HINs by leveraging the semantic relationships defined by meta-paths. Additionally, $N_{G_P}(u)$ denotes the set of neighboring vertices connected to u in G_P.

Example 2.1. *Consider the HIN G illustrated in Figure 2(a), which includes users (U), devices (D), Wi-Fi (W), and merchants (M). We examine two meta-paths in Figure 2(b): $P_1 = (U, D, W)$; and $P_2 = (U, D, M)$. For meta-path P_1, the induced homogeneous graph G_{P_1} connects users who have accessed the same Wi-Fi via devices. For instance, u_0 and u_5 are connected in G_{P_1} because they both have matching instances ending at w_0: $M_{u_0} = u_0 \to d_0 \to w_0$ and $M_{u_5} = u_5 \to d_1 \to w_0$. Similar analysis applies to other matching instances to form G_{P_1} shown in Figure 2(c). Likewise, for meta-path P_2, we derive the induced homogeneous graph G_{P_2} depicted in Figure 2(c).*

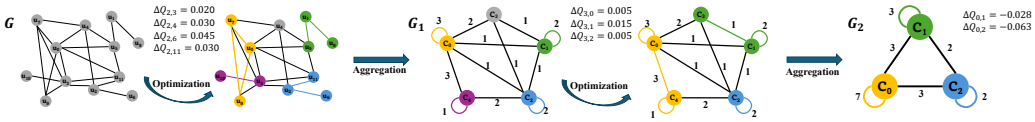

Fig. 3. Running Example of Louvain Algorithm.

Algorithm 1: Louvain Algorithm

Input: Graph $G = (V, E)$
Output: Community partition

1 Initialize each node u_i in its own community C_i
2 Calculate initial modularity Q
3 **repeat**
4 **foreach** $u_i \in V$ **do**
5 **foreach** community C adjacent to u_i **do**
6 Compute ΔQ for moving u_i to C using Equation (2).
7 Move u_i to the community C that maximizes ΔQ (if $\Delta Q > 0$)
8 **Community Aggregation:** Build a new graph G' where each node represents a community from G
9 **until** *modularity Q does not increase*
10 **return** *Final community assignments*

2.2 Revisiting the Louvain Algorithm

The Louvain algorithm [4] is a popular approach for detecting communities in large networks by maximizing *modularity* [40]. It iterates between local optimization and aggregation, reducing the problem size in subsequent iterations.

Community. Given a graph $G = (V, E)$, a *community* is a subset of nodes $C \subseteq V$. Each node $u_i \in V$ belongs to a community C_i. The goal is to partition the network into communities such that the connections within communities are denser than between them.

Modularity. *Modularity* measures how well a network is partitioned into communities, comparing observed intra-community edges against those expected under a random baseline. Formally,

$$Q = \frac{1}{2m} \sum_{u_i, u_j \in V} \left[k_{ij} - \frac{k_i k_j}{2m} \right] \delta(C_i, C_j), \tag{1}$$

- k_{ij} is the number of edges between nodes u_i and u_j;
- $k_i = \sum_j k_{ij}$ is the number of edges incident to node u_i;
- $m = \frac{1}{2} \sum_i k_i$ is the number of edges in the network;
- $\delta(C_i, C_j)$ is the Kronecker delta function: $\delta(C_i, C_j) = 1$ if $C_i = C_j$, and 0 otherwise.

When relocating a node u_i to a neighboring community C, the change in modularity ΔQ is:

$$\Delta Q = \frac{1}{2m} \left(\sum_{u_j \in C} k_{ij} - \frac{k_i \sum_{u_j \in C} k_j}{2m} \right). \tag{2}$$

$\sum_{u_j \in C} k_{ij}$ is the number of edges from u_i to C, and $\sum_{u_j \in C} k_j$ is the sum of degrees of nodes in C.

Algorithmic Steps. Louvain proceeds in two phases, as summarized in Algorithm 1 and Figure 3:

(1) *Local Optimization:* Each node u_i is initially in its own community (Line 1). We repeatedly check if moving u_i to an adjacent community yields the greatest *positive* ΔQ (Equation (2)). If so, u_i is reassigned to that community, thus *maximizing* its local contribution to Q (Lines 4 to 7).
(2) *Community Aggregation:* Once no further improvement is possible, all nodes in each community are merged into a single node, forming a smaller graph G'. The algorithm restarts the local optimization on G' until no further modularity gains are achieved.

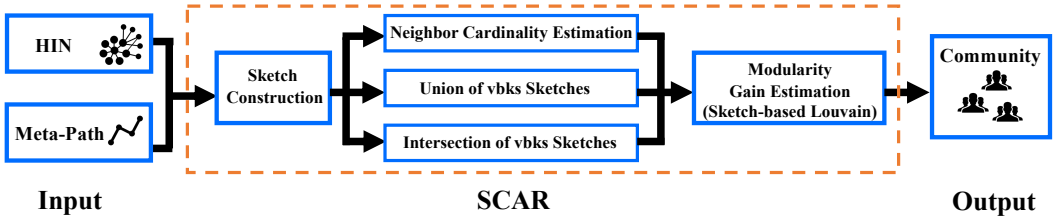

Fig. 4. Overview of SCAR.

Example 2.2 (Running Example of Louvain Algorithm (Figure 3)). *Consider the node u_2 in a graph undergoing community detection using the Louvain algorithm.*

(1) Initially, u_2 belongs to community C_2. The algorithm evaluates whether moving u_2 to the community of one of its neighboring nodes (such as u_3, u_4, u_6, or u_{11}) would maximize the modularity by calculating the change in modularity, $\Delta Q_{2,j}$ for each possible assignment.

(2) Moving u_2 to the community of u_6 results in the highest $\Delta Q_{2,6}$, with a value of 0.045. Since this value is the largest and positive, u_2 is moved from C_2 to the community of u_6. If other neighboring nodes resulted in lower ΔQ values, the algorithm would avoid those moves.

(3) When no further movements can increase modularity, we proceed to graph aggregation, where nodes within the same community are merged; for example, u_0, u_7, u_9 would be merged into community c_0, resulting in a new graph G_1. Recursively, local optimization and aggregation continue on G_1, eventually leading to the final graph G_2.

While the Louvain algorithm is efficient for homogeneous graphs, applying it directly to HINs is challenging due to the need to materialize induced homogeneous graphs for each meta-path, which is computationally expensive. Therefore, our goal is to develop a framework that enables efficient community detection on HINs without explicit graph materialization.

Problem Statement. Given a HIN $G = (V, E, L)$ and a meta-path P, our objective is to efficiently compute a community partition of the induced homogeneous graph $G_P = (V_P, E_P)$ that maximizes the modularity metric Q [1].

3 SCAR: Sketch-based Community Detection Algorithms on HIN

Overview (Figure 4). In this section, we introduce SCAR—a **S**ketch-based **C**ommunity Detection **A**lgo**R**ithm in HINs—to meet the dual challenges of efficiency and scalability in heterogeneous network mining. The core idea is to replace expensive graph materialization and exhaustive neighbor enumeration with succinct *Bottom-K Sketches* that capture how nodes connect through specific meta-paths. SCAR then plugs these compact sketches into a Louvain-style modularity optimization process: at each iteration, the algorithm estimates both node degrees and inter-community links via sketch merges and unions, reassigns nodes to improve modularity, and finally aggregates resulting communities before repeating. By leveraging these data structures, SCAR maintains high-quality community partitions without exploding either memory or time complexity. The subsections detail each building block of SCAR:

(1) *Sketch Construction:* Section 3.1 describes how vbks sketches are constructed by propagating random values along meta-paths. These sketches capture the topological information of constraint-based neighborhoods, enabling efficient neighbor estimation.

(2) *Neighbor Cardinality Estimation:* Section 3.2 explains how vbks sketches can be used to estimate the degree of each vertex in the induced homogeneous graph. By applying both forward and

[1]The current scope of this paper focuses exclusively on the topological structure, estimating communities based solely on connectivity patterns without incorporating node attributes or edge weights.

reverse meta-paths, we ensure that the sketches capture the size of neighborhoods without explicitly materializing the graph.

(3) *Union of* vbks *and Intersection of* vbks*:* Section 3.3 and 3.4 detail efficient estimation of inter-community connections. Instead of counting edges, SCAR approximates the size of neighborhood unions and intersections by merging sketches and using the inclusion-exclusion principle.

(4) *Sketch-based Louvain:* Section 3.5 adapts the Louvain algorithm for modularity optimization using vbks sketches. In each iteration, nodes are reassigned to communities based on estimated modularity gains, computed through the degree approximations.

3.1 Sketch Construction

Traditional approaches often involve materializing induced homogeneous graphs or exhaustively enumerating all possible paths, which are computationally expensive. To overcome these challenges, we introduce the Vertex Bottom-K Sketch (vbks), a compact probabilistic data structure that enables scalable neighborhood estimation under specific meta-path constraints. At the core of vbks is the Bottom-K Sketch, which succinctly captures the smallest K random values from a set, allowing us to estimate the cardinality of large neighborhoods without exhaustive storage.

Definition 3.1 (Bottom-K Sketch [10, 11]). *A* Bottom-K sketch *for a set R retains the K smallest random values $\{r_1, r_2, \ldots, r_K\}$ from R. Each element in R is assigned a random value $r \in (0, 1)$, drawn uniformly at random. The estimated cardinality $\hat{n}$ of R is*

$$\hat{n} \;=\; \frac{K-1}{r_K},$$

where r_K is the largest among the K smallest values stored in the sketch (i.e., the K-th order statistic). We denote r_K of a set R by $r_K(R)$.

Intuitively, the position of the K-th smallest value in $(0, 1)$ reveals the approximate size of R. This design avoids storing all elements of R, keeping only K values. Leveraging the Bottom-K Sketch, we extend it to the Vertex Bottom-K Sketch (vbks) to estimate the number of shared matching instances for each target vertex in an induced homogeneous graph.

Definition 3.2 (Vertex Bottom-K Sketch (vbks)). *Let $G = (V, E, L)$ be a HIN and P a meta-path. The* Vertex Bottom-K Sketch *of a vertex $u \in V$, $\mathsf{vbks}_P(u)$, is a Bottom-K sketch of vertices that reach u through paths conforming to P. Formally, $\mathbb{N}_P(u)$ is the set of vertices that can reach u via P, then*

$$\mathsf{vbks}_P(u) = \mathsf{BotK}(\{r(x) \mid x \in \mathbb{N}_P(u)\}),$$

$r(\cdot)$ is a hash function mapping vertices to $(0, 1)$ and $\mathbb{N}_P(u)$ denotes all vertices that can reach u via P.

The approximate neighborhood size for u under P is given by

$$|\mathbb{N}_P(u)| \;\approx\; \frac{K-1}{r_K(\mathsf{vbks}_P(u))},$$

where $r_K(\mathsf{vbks}_P(u))$ is the K-th smallest (i.e., the largest among the retained) hash value in $\mathsf{vbks}_P(u)$.

Meta-path Supervised Vertex Bottom-K Sketch Construction (Algorithm 2). The Meta-path Supervised Vertex Bottom-K Sketch (vbks) Construction Algorithm initializes and populates compact sketches for each vertex in a graph $G = (V, E, L)$, guided by a meta-path P. Each vertex $u \in V$ of type A_1 (the starting node type of the meta-path) is assigned a random number $r(u)$ within the range $(0, 1)$, which is then added to an initially empty sketch $\mathsf{vbks}(u)$. The algorithm performs a Breadth-First Search (BFS), traversing the graph along paths that conform to the meta-path P. At each step, it aggregates the K smallest hash values from the current vertex's sketch and those of its neighboring vertices that match the next node type in P connected via edges of the specified types. This process progressively captures connectivity information along paths defined by P, enriching

Algorithm 2: Meta-path Supervised vbks Construction

Input: $G = (V, E, L)$, $P = A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \dots \xrightarrow{R_n} A_{n+1}$, Sketch size K
Output: The vbks sketches for vertices of type A_{n+1}

```
foreach vertex u ∈ V do
    Initialize vbks(u) as empty
    if L(u) = A_1 then
        Assign random value r(u) ∈ [0, 1]
        Add r(u) to vbks_P(u)
        Enqueue (u, 1) into queue Q
while Q is not empty do
    Dequeue (u, d) from Q
    if d > n then
        Continue
    foreach v ∈ N(u) do
        if L(v) = A_{d+1} and L((u, v)) = R_d then
            Update vbks_P(v) = BotK(vbks_P(v) ∪ vbks_P(u))
            Enqueue (v, d + 1) into Q
return vbks(v) for all v ∈ V where L(v) = A_{n+1}
```

each vertex's sketch with data that encapsulates reachability and structural relationships specified by the meta-path. The final output is a set of sketches for vertices of the target type A_{n+1}, which succinctly summarize the topological structures defined by the meta-path constraints.

Time Complexity. The algorithm involves initializing sketches for all vertices, which takes $O(|V|)$ time. The BFS traversal explores edges that match the meta-path constraints. For each such edge, it performs a sketch merge operation, which takes $O(K)$ time, where K is the sketch size. Therefore, the time complexity is $O(K(|V|+|E|))$.

Theoretical analysis. The primary objective of vbks is to accurately estimate the size of the constraint-based neighborhood $\mathbb{N}_P(u)$ for each vertex u. We present the following theorem that provides probabilistic bounds on the estimation accuracy.

Theorem 3.1. *Given a vertex $u \in V$ and a meta-path P, the estimated size of the neighborhood $|\hat{\mathbb{N}_P}(u)|$ using $\text{vbks}_P(u)$ satisfies:*

$$\Pr\left(\left||\hat{\mathbb{N}_P}(u)| - |\mathbb{N}_P(u)|\right| \geq \epsilon \cdot |\mathbb{N}_P(u)|\right) \leq \delta,$$

provided that the sketch size K satisfies:

$$K \geq \frac{1}{\epsilon^2 \delta} + 2.$$

Proof. The Bottom-K sketch provides an unbiased estimator for $|\mathbb{N}_P(u)|$, i.e., $\mathbb{E}\left[|\hat{\mathbb{N}_P}(u)|\right] = |\mathbb{N}_P(u)|$. The estimation variance is:

$$\text{Var}\left[|\hat{\mathbb{N}_P}(u)|\right] = |\mathbb{N}_P(u)|^2 \cdot \frac{1}{K-2}.$$

This variance formula is derived from the properties of order statistics of uniform random variables [13]. Applying Chebyshev's inequality [38], we have:

$$\Pr\left(\left||\hat{\mathbb{N}_P}(u)| - |\mathbb{N}_P(u)|\right| \geq \epsilon \cdot |\mathbb{N}_P(u)|\right) \leq \frac{\text{Var}\left[|\hat{\mathbb{N}_P}(u)|\right]}{(\epsilon \cdot |\mathbb{N}_P(u)|)^2} = \frac{1}{(K-2)\epsilon^2}.$$

Setting the right-hand side equal to δ:

$$\frac{1}{(K-2)\epsilon^2} = \delta \implies K \geq \frac{1}{\epsilon^2 \delta} + 2.$$

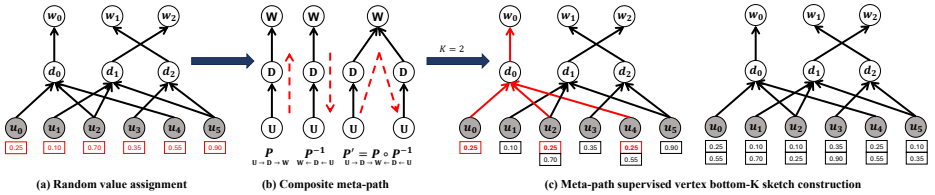

Fig. 5. Running Example of Sketch Construction.

This ensures that the probability that the relative estimation error exceeds ϵ is at most δ, i.e.,

$$\Pr\left(\left|\left|\mathbb{N}_P\hat{(u)}\right| - \left|\mathbb{N}_P(u)\right|\right| < \epsilon \cdot \left|\mathbb{N}_P(u)\right|\right) \geq 1 - \delta.$$

□

In summary, vbks provides a scalable way to estimate the cardinalities of meta-path–constrained neighborhoods, which is crucial for downstream tasks such as modularity-based community detection, all without enumerating paths or materializing large induced graphs.

3.2 Homogeneous Neighbor Cardinality Estimation

To facilitate degree estimation in the induced homogeneous graph G_P, we introduce *inverse meta-paths* and their concatenations with the original meta-paths.

Definition 3.3 (Inverse Meta-path). *Given a meta-path* $P = A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \dots \xrightarrow{R_n} A_{n+1}$, *the* inverse meta-path P^{-1} *is defined by reversing the sequence of node types and edge types, and traversing edges in the reverse direction:*

$$P^{-1} = A_{n+1} \xleftarrow{R_n} A_n \xleftarrow{R_{n-1}} \dots \xleftarrow{R_1} A_1,$$

where each $\xleftarrow{R_i}$ *indicates traversal over edge* R_i *from node type* A_{i+1} *to node type* A_i, *opposite to the edge's direction in the graph.*

By concatenating the original meta-path P with its inverse P^{-1}, we form the *composite meta-path* P'. This composite meta-path captures paths that start and end at the same node type, effectively allowing us to model connections within the induced homogeneous graph.

Definition 3.4 (Composite Meta-path P'). *P' is obtained by concatenating P and P^{-1}:*

$$P' = P \circ P^{-1} = A_1 \xrightarrow{R_1} \dots \xrightarrow{R_n} A_{n+1} \xleftarrow{R_n} A_n \xleftarrow{R_{n-1}} \dots \xleftarrow{R_1} A_1.$$

This composite meta-path P' captures bidirectional relationships to model connections within G_P without explicitly constructing it.

Lemma 3.2. *In the induced homogeneous graph $G_P = (V_P, E_P)$ derived from G and meta-path P, there exists an edge $e = (u, v) \in E_P$ between vertices $u, v \in V_P$ iff there exists a matching instance M of the composite meta-path P' such that $M(u_1) = u$ and $M(u_{2n+1}) = v$.*

Proof. **(⇒)** Suppose there exists an edge $e = (u, v) \in E_P$ in the induced homogeneous graph G_P. By the definition of G_P, this implies that there exists at least one common terminal vertex $w \in V$ such that there are matching instances $M_1 \triangleright P$ and $M_2 \triangleright P$ ending at w.

Construct the composite meta-path $P' = P \circ P^{-1}$ as follows:

$$P' = A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \dots \xrightarrow{R_n} A_{n+1} \xleftarrow{R_n} A_n \xleftarrow{R_{n-1}} \dots \xleftarrow{R_1} A_1.$$

A matching instance $M \triangleright P'$ can be constructed by concatenating M_1 and M_2's reverse. Specifically:

$$M = M_1 \circ M_2^{-1},$$

where M_2^{-1} is the inverse path of M_2. This matching instance starts at u and ends at v, thus establishing the existence of $e = (u, v)$ in G_P.

(⇐) Conversely, suppose there exists a matching instance $M \triangleright P'$ such that $M(u_1) = u$ and $M(u_{2n+1}) = v$. By the definition of P', M consists of a forward traversal along P from u to some vertex w, followed by a reverse traversal along P^{-1} from w to v.

This implies that there are matching instances $M_1 \triangleright P$ from u to w and $M_2 \triangleright P$ from v to w. According to the construction of the induced homogeneous graph G_P, the existence of these two matching instances sharing the terminal vertex w implies the existence of an edge $e = (u, v)$ in E_P.

Conclusion: Therefore, $e = (u, v) \in E_P$ if and only if there exists a matching instance $M \triangleright P'$ connecting u and v via the composite meta-path $P' = P \circ P^{-1}$. □

Definition 3.5 (HOM-Neighborhood). *Given a vertex $u \in V_P$, the* neighborhood *of u along the composite meta-path P' is defined as:*

$$N_{G_P}(u) = \mathbb{N}_{P'}(u) = \{v \in V_P \mid \exists M \triangleright P' \text{ such that } M = v \xrightarrow{P'} u\}.$$

where $M \triangleright P'$ denotes a matching instance of P' from v to u.

Example 3.1. *The process begins by assigning a random value to each vertex of the source type U in the graph (see Figure 5(a)). In the first step of* SCAR*, with a sketch size of $K = 2$, we apply the meta-path $P = \{U \to D \to W\}$ and its inverse $P^{-1} = \{W \leftarrow D \leftarrow U\}$ to construct the sketches, resulting in the composite meta-path $P' = P \circ P^{-1}$ (see Figure 5(b)).*
For instance, vertex u_0 is assigned a random value $r(u_0) = 0.25$, which propagates along matching instances of P', such as the path $u_0 \to d_0 \to w_0 \leftarrow d_0 \leftarrow u_2$, so that the sketch of u_2 becomes $\mathsf{vbks}(u_2) = \{0.25, 0.70\}$ *(illustrated on the left-hand side of Figure 5(c)). After all random values have been propagated, the sketches for all vertices are fully constructed (as shown on the right-hand side of Figure 5(c)); for example, since $r(u_1) = 0.1$ propagates via the path $u_1 \to d_1 \to w_2 \leftarrow d_1 \leftarrow u_2$, the final sketch of u_2 becomes* $\mathsf{vbks}(u_2) = \{0.10, 0.25\}$.

Using the vbks sketches constructed along the composite meta-path $P' = P \circ P^{-1}$, the degree k_i of a vertex u_i in the induced homogeneous graph G_P can be estimated by:

$$k_i = |\mathbb{N}_{P'}(u_i)| \approx \frac{K-1}{r_K(\mathsf{vbks}_{P'}(u_i))}.$$

$\mathsf{vbks}_{P'}(u_i)$ is the Vertex Bottom-K Sketch of u_i constructed along the composite meta-path P'.

Example 3.2. *Consider vertex u_2 in the induced homogeneous graph G_P derived from a heterogeneous network G using the meta-path P (see Figure 6(a)). Although* SCAR *does not require explicit graph materialization, this example illustrates its neighborhood estimation process. Suppose that u_2 receives four random values $\{0.25, 0.10, 0.55, 0.90\}$ from vertices $\{u_0, u_1, u_4, u_5\}$ via matching instances of the composite meta-path $P' = P \circ P^{-1}$. With a sketch size of $K = 2$, the sketch for u_2 is determined as* $\mathsf{vbks}(u_2) = \{0.10, 0.25\}$*, retaining the two smallest values. The maximum value in the sketch is 0.25, so the estimated neighborhood size of u_2 is calculated as* $|N_{G_P}(u_2)| \approx \frac{K-1}{r_K(\mathsf{vbks}(u_2))} = \frac{2-1}{0.25} = 4$.

This example demonstrates how the sketch efficiently captures the connectivity of u_2 in G_P, enabling accurate neighborhood estimation without explicitly enumerating all its neighbors. Similarly, we can estimate the neighborhood sizes of other vertices in G_P using their corresponding sketches.

Complexity. The vbks sketch allows SCAR to efficiently estimate neighborhood sizes. The time complexity is dominated by the initialization and Breadth-First Search (BFS) traversal phases.

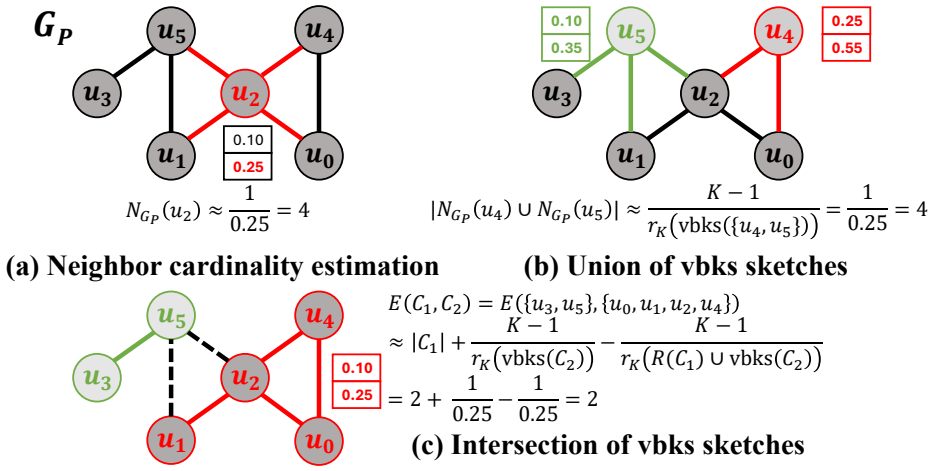

Fig. 6. Sketch Union and Sketch Intersection.

Initializing sketches for all vertices requires $O(|V|)$ time. During BFS traversal, each edge that matches the meta-path constraints is processed, and merging sketches incurs $O(K)$ time per edge. Consequently, the overall time complexity is $O(K(|V|+|E|))$. Regarding space complexity, each vertex of type A_{n+1} maintains a sketch of size K, leading to an overall space complexity of $O(K|V|)$.

3.3 Union of vbks Sketches

To compute the ΔQ in community detection, we must estimate inter-node and inter-community edges without enumerating all connections. This subsection shows how unions (and later intersections) of vbks sketches yield estimates of neighborhood unions and inter-community connectivity.

Definition 3.6 (Union of vbks Sketches). *Let $u_i, u_j \in V$ have Vertex Bottom-K sketches* $\text{vbks}(u_i)$ *and* $\text{vbks}(u_j)$. *The* union sketch *is*

$$\text{vbks}(\{u_i,u_j\}) \;=\; \text{BotK}\Big(\text{vbks}(u_i) \;\cup\; \text{vbks}(u_j)\Big),$$

where $\text{BotK}(S)$ *retains the K smallest random values from set S.*

By performing the union of the sketches, we obtain a new sketch that estimates the size of the combined neighborhood:

$$|\mathbb{N}_{P'}(u_i) \cup \mathbb{N}_{P'}(u_j)| \;\approx\; \frac{K-1}{r_K(\text{vbks}(\{u_i,u_j\}))},$$

To simplify notation and improve readability, we define:

$$\hat{n}_i = \text{estimated size of } \mathbb{N}_{P'}(u_i) = \frac{K-1}{r_K(\text{vbks}(u_i))},$$

$$\hat{n}_j = \text{estimated size of } \mathbb{N}_{P'}(u_j) = \frac{K-1}{r_K(\text{vbks}(u_j))},$$

$$\hat{n}_{i\cup j} = \text{estimated size of } \mathbb{N}_{P'}(u_i) \cup \mathbb{N}_{P'}(u_j) = \frac{K-1}{r_K(\text{vbks}(\{u_i,u_j\}))}.$$

Similarly, the true sizes are denoted by $n_i = |\mathbb{N}_{P'}(u_i)|$, $n_j = |\mathbb{N}_{P'}(u_j)|$, and $n_{i\cup j} = |\mathbb{N}_{P'}(u_i) \cup \mathbb{N}_{P'}(u_j)|$.

Theorem 3.3 (Accuracy of Union Estimation). *For any $\epsilon > 0$ and $\delta \in (0,1)$, if K satisfies:*

$$K \geq \frac{1}{\epsilon^2 \delta} + 2,$$

then the estimate $\hat{n}_{i\cup j}$ satisfies:

$$\Pr\left(\left|\hat{n}_{i\cup j} - n_{i\cup j}\right| \geq \epsilon n_{i\cup j}\right) \leq \delta.$$

Proof. The Bottom-K sketch of a set provides an unbiased estimator of the cardinality of that set. Since vbks(u_i) and vbks(u_j) are the Bottom-K sketches of $\mathbb{N}_{P'}(u_i)$ and $\mathbb{N}_{P'}(u_j)$ respectively, their union vbks($\{u_i, u_j\}$) retains the K smallest random values from the combined set $\mathbb{N}_{P'}(u_i) \cup \mathbb{N}_{P'}(u_j)$.

Because the random values assigned to elements are independent and uniformly distributed over $(0, 1)$, the Bottom-K sketch of the union set $\mathbb{N}_{P'}(u_i) \cup \mathbb{N}_{P'}(u_j)$ is correctly maintained by vbks($\{u_i, u_j\}$). Therefore, using the Bottom-K estimator:

$$\hat{n}_{i\cup j} = \frac{K-1}{r_K(\text{vbks}(\{u_i, u_j\}))}$$

is an unbiased estimator of $n_{i\cup j}$. This holds true regardless of whether $\mathbb{N}_{P'}(u_i)$ and $\mathbb{N}_{P'}(u_j)$ are disjoint, as the union sketch captures the K smallest random values from the combined neighborhood. □

3.4 Intersection of vbks Sketches

Estimating the number of edges between nodes and communities is essential for computing ΔQ, yet enumerating all connections in the induced homogeneous graph G_P can be prohibitively expensive. This subsection shows how *union* and *intersection* of vbks sketches, in conjunction with the inclusion-exclusion principle, can efficiently approximate inter-community connectivity.

Estimating Intersection Sizes Using Inclusion-Exclusion. While directly estimating the size of the intersection $\mathbb{N}_{P'}(u_i) \cap \mathbb{N}_{P'}(u_j)$ from the sketches is challenging, we can approximate it using the inclusion-exclusion principle:

$$\hat{n}_{i\cap j} = \hat{n}_i + \hat{n}_j - \hat{n}_{i\cup j}, \tag{3}$$

where $\hat{n}_i$, $\hat{n}_j$, and $\hat{n}_{i\cup j}$ are the estimated sizes of $\mathbb{N}_{P'}(u_i)$, $\mathbb{N}_{P'}(u_j)$, and their union, respectively.

Example 3.3 (Union and Intersection of vbks Sketches). *Figures 6(b) and 6(c) illustrate how to perform union and intersection operations using* vbks *sketches. For example, consider vertices u_4 and u_5 with sketches* vbks$(u_4) = \{0.25, 0.55\}$ *and* vbks$(u_5) = \{0.10, 0.35\}$. *Their union sketch, obtained by taking the two smallest values from the combined set, is* vbks$(u_4, u_5) = \{0.10, 0.25\}$. *Consequently, the estimated neighborhood size for $\{u_4, u_5\}$ is given by $|N(\{u_4, u_5\})| \approx \frac{K-1}{r_K(\text{vbks}(u_4,u_5))} = \frac{2-1}{0.25} = 4$. Similarly, for vertex sets $C_1 = \{u_3, u_5\}$ and $C_2 = \{u_0, u_1, u_2, u_4\}$, the union sketch for C_2 is* vbks$(C_2) = \{0.10, 0.25\}$. *Using the inclusion-exclusion formula (Equation* (4)*), the inter-community connectivity is estimated as $E(C_1, C_2) \approx 2 + \frac{1}{0.25} - \frac{1}{0.25} = 2$.*

Bottom-K Sketch for Communities. For a community $C \subseteq V$, we define its sketch as:

$$\text{vbks}(C) = \text{BotK}\left(\bigcup_{u\in C} \text{vbks}(u)\right).$$

The estimated size of the union neighborhood of C is:

$$\hat{n}_C = \frac{K-1}{r_K(\text{vbks}(C))},$$

where $r_K(\text{vbks}(C))$ is the K-th smallest hash value in vbks(C).

Definition 3.7 (Inter-Community Connectivity). *Given two disjoint communities $C_1, C_2 \subseteq V$, the* inter-community connectivity *between C_1 and C_2 is defined as the total number of edges where one endpoint is in C_1 and the other is in C_2:*

$$E(C_1, C_2) = \left|\{(u, v) \in E_P \mid u \in C_1, v \in C_2\}\right|.$$

Alternatively, the inter-community connectivity can be expressed via neighborhood intersection:

$$E(C_1, C_2) \;=\; |\, C_1 \,\cap\, \mathbb{N}_{P'}(C_2)|.$$

Estimating Inter-Community Connectivity. To avoid enumerating all cross edges, we apply inclusion-exclusion on the sets C_1 and $\mathbb{N}_{P'}(C_2)$. Let $R(C_1) = \{r(u) \mid u \in C_1\}$ be the hash values from vertices in C_1. Then we can approximate $E(C_1, C_2)$ as:

$$\hat{E}(C_1, C_2) \;\approx\; |C_1| + \hat{n}_{C_2} \;-\; |C_1 \,\cup\, \mathbb{N}_{P'}(C_2)|. \tag{4}$$

Here, the estimated size of union neighborhood of C_2 is $\hat{n}_{C_2} \approx (K-1)/r_K(\mathsf{vbks}(C_2))$, and $|C_1 \cup \mathbb{N}_{P'}(C_2)| \approx (K-1)\,/\,r_K(R(C_1) \cup \mathsf{vbks}(C_2))$.

Complexity Analysis. The union and intersection operations on vbks sketches involve merging sorted lists of size K and selecting the top K smallest elements, resulting in time complexity of $O(K)$ per operation. Maintaining sketches for all communities requires $O(K|C|)$ space, where $|C|$ is the number of communities.

3.5 Sketch-based Louvain

We now adapt Louvain to leverage vbks sketches for efficient community detection in large HINs.

Algorithm Overview. The algorithm begins by assigning each node $u_i \in V$ to its own community C_i and computing its Vertex Bottom-K Sketch $\mathsf{vbks}(u_i)$ (see Algorithm 3, Lines 2–4). The degree k_i of each node u_i is estimated by:

$$k_i \approx \frac{K-1}{r_K(\mathsf{vbks}(u_i))}. \tag{5}$$

The number of edges m of G_P is then estimated (Algorithm 3, Line 5) as:

$$m \approx \frac{1}{2} \sum_{u_i \in V_P} k_i. \tag{6}$$

In the local optimization phase (Lines 6–24), each node u evaluates the modularity gain ΔQ for moving to adjacent communities, estimated using the inclusion-exclusion principle and sketches (Line 12). If $\Delta Q > 0$, u moves to the community that maximizes ΔQ. When no further improvement is possible, communities are aggregated to form a new graph G', and community sketches are updated (Lines 21–22). The process repeats until the modularity stabilizes.

Modularity Gain Estimation. The ΔQ when moving node u_i to community C is given by:

$$\Delta Q = \frac{1}{2m}\left(\sum_{u_j \in C} k_{ij} - \frac{k_i \cdot \sum_{u_j \in C} k_j}{m}\right) \tag{7}$$

$$\approx \frac{1}{2m}\left(E(u_i, C) - \frac{k_i \cdot k_C}{m}\right), \tag{8}$$

- k_{ij} is the number of edges between u_i and u_j in G_P.
- k_i is the degree of node u_i in G_P.
- $k_C = \sum_{u_j \in C} k_j$ is the total degree of community C.
- $E(u_i, C)$ is the estimated number of edges between node u_i and community C.

Since G_P is not materialized, $E(u_i, C)$ is estimated using sketches:

$$E(u_i, C) \approx \hat{E}(u_i, C) = \hat{k}_i + |C| - |C \cup N_{G_P}(u_i)|, \tag{9}$$

- $\hat{k}_i = \dfrac{K-1}{r_K(\mathsf{vbks}(u_i))}$ is the estimated degree of u_i.
- $|C \cup N_{G_P}(u_i)| \approx \dfrac{K-1}{r_K(R(C) \cup \mathsf{vbks}(u_i))}$ is the estimated size of the union of $N_{G_P}(u_i)$ and C.

Edge Count Estimation Error. To apply our sketch-based approach reliably, we must bound the error in approximating $E(u_i, C)$. The following theorem shows that $\hat{E}(u_i, C)$ remains close to the true edge count $E(u_i, C)$ with high probability:

Theorem 3.4. *Let $\epsilon > 0$ and $\delta \in (0, 1)$. If*

$$K \;\geq\; \frac{1}{\epsilon^2\,\delta} \;+\; 2,$$

then with probability at least $1 - 2\delta$,

$$|\hat{E}(u_i, C) - E(u_i, C)| \;\leq\; 2\,\epsilon\,S,$$

where $S = \max\{\, k_i, k_C, k_{i\cup C}\}$ and $k_{i\cup C} = |\, C \cup N_{G_P}(u_i)|$.

Proof. Define $\hat{k}_i$ and $\hat{k}_{i\cup C}$ similarly as unbiased estimates of k_i and $k_{i\cup C}$. Their variance decreases with sketch size K. By Chebyshev's inequality, the probability that $\hat{k}_i$ (or $\hat{k}_{i\cup C}$) deviates from its true value by at least ϵS is at most δ. The union bound implies both estimates simultaneously deviate with probability $\leq 2\delta$. Consequently,

$$|\hat{E}(u_i, C) - E(u_i, C)| \;\leq\; |\hat{k}_i - k_i| \;+\; |\hat{k}_{i\cup C} - k_{i\cup C}| \;\leq\; 2\,\epsilon\,S.$$

□

Theorem 3.5 (Estimation Error Impact). *Let $\Delta Q_{est}(u_i, C)$ be the estimated gain for u_i joining C, and $\Delta Q_{true}(u_i, C)$ the actual gain. Suppose $|\Delta Q_{est}(u_i, C) - \Delta Q_{true}(u_i, C)| \leq \epsilon$. Let C^* be the community that maximizes ΔQ_{true}, and let C' be any other community. If*

$$\Delta_{\min} = \;\Delta Q_{true}(u_i, C^*) - \;\Delta Q_{true}(u_i, C') > \;2\,\epsilon,$$

then u_i joins C^* with probability at least $1 - \delta$, δ is the probability that the error bound is exceeded.

Proof. By estimation-error assumption, for any community C:

$$\Delta Q_{\text{true}}(u_i, C) - \epsilon \;\;\leq\;\; \Delta Q_{\text{est}}(u_i, C) \;\;\leq\;\; \Delta Q_{\text{true}}(u_i, C) + \epsilon.$$

If u_i moves to $C' \neq C^*$, we must have

$$\Delta Q_{\text{est}}(u_i, C') \;\;\geq\;\; \Delta Q_{\text{est}}(u_i, C^*).$$

Combining these bounds gives

$$\begin{aligned} \Delta Q_{\text{true}}(u_i, C') + \epsilon \;&\geq\; \Delta Q_{\text{est}}(u_i, C') \\ &\geq\; \Delta Q_{\text{est}}(u_i, C^*) \\ &\geq\; \Delta Q_{\text{true}}(u_i, C^*) - \epsilon. \end{aligned} \tag{10}$$

$$\Rightarrow \Delta Q_{\text{true}}(u_i, C^*) \;-\; \Delta Q_{\text{true}}(u_i, C') \;\;\leq\;\; 2\,\epsilon.$$

This contradicts that $\Delta_{\min} > 2\epsilon$. Therefore, s long as $\epsilon < \frac{\Delta_{\min}}{2}$, u_i will correctly move to C^* with probability at least $1 - \delta$. □

Implications. The theorem shows that if ϵ is small compared to the gap between the best and second-best communities, nodes will determine their optimal communities with high probability.

Ensuring Correct Community Assignment. To guarantee u_i joins the optimal community C^*, select K so $\epsilon \leq \frac{\Delta_{\min}}{2}$. This ensures sketch-based estimation errors do not force a suboptimal choice.

4 Eliminating Graph Aggregation

In the Louvain algorithm, after each phase of community detection, an aggregated graph G' is constructed where each node represents a community from the previous phase. This graph aggregation step can be computationally expensive and memory-intensive for large HINs. To

Algorithm 3: Sketch-Based Louvain Algorithm

Input : Graph $G = (V, E, L)$; Meta-path P; Sketch size K
Output : Community assignments

foreach $u \in V$ **do**
  Assign u to its own community C_u
  Compute vbks(u)
  Estimate degree $k_u \leftarrow \dfrac{K-1}{\max\{\text{vbks}(u)\}}$
Estimate the number of edges $m \leftarrow \dfrac{1}{2}\sum_u k_u$
repeat
  foreach $u \in V$ **do**
    $C_u \leftarrow$ community of u
    $\Delta Q_{\max} \leftarrow 0$
    $C_{\text{best}} \leftarrow C_u$
    foreach *community C adjacent to u* **do**
      Estimate ΔQ
      if $\Delta Q > \Delta Q_{max}$ **then**
        $\Delta Q_{\max} \leftarrow \Delta Q$
        $C_{\text{best}} \leftarrow C$
    if $C_{best} \neq C_u$ ***and*** $\Delta Q_{max} > 0$ **then**
      Move u to community C_{best}
      Mark improvement
  if no improvement **then**
    break
  Aggregate communities to form G'
  Update sketches vbks(C)
  $G \leftarrow G'$
until no improvement
return *Final community assignments*

address this, we introduce an optimization that eliminates the need for explicit graph aggregation by leveraging recursive computation of vbks sketches and efficient estimation of modularity gains.

Recursive Computation of vbks **Sketches.** Each node u_i in the conceptual aggregated graph G' represents a community C_i comprising multiple nodes from the original graph G. Instead of physically constructing G', we compute the vbks sketch for each community C_i by merging the sketches of all nodes within the community:

$$\text{vbks}(C_i) = \text{BotK}\left(\bigcup_{u \in C_i} \text{vbks}(u)\right). \tag{11}$$

Estimating Community Degrees. The degree k_C of a community C is estimated using its sketch:

$$\hat{k}_C = \frac{K-1}{r_K(\text{vbks}(C))}, \tag{12}$$

where $\hat{k}_C$ is the estimated degree, and k_C is the true degree.

From Theorem 3.3, the estimation error for the degree of a community C is bounded. Specifically, for any $\epsilon > 0$ and $\delta \in (0, 1)$, if the sketch size K satisfies $K \geq \frac{1}{\epsilon^2 \delta} + 2$, then:

$$\Pr\left(\left|\hat{k}_C - k_C\right| \geq \epsilon k_C\right) \leq \delta \tag{13}$$

Efficient Modularity Gain Computation. The modularity gain ΔQ when considering merging community C_i into another community C_j is given by:

$$\Delta Q = \frac{1}{2m}\left(E(C_i, C_j) - \frac{\hat{k}_{C_i} \cdot \hat{k}_{C_j}}{2m}\right), \tag{14}$$

where $E(C_i, C_j)$ is the number of edges between C_i and C_j.

Estimating Inter-Community Connections. We approximate the number of edges between communities C_i and C_j using an inclusion-exclusion–based formula:

$$\hat{E}(C_i, C_j) = \hat{k}_{C_i} + |C_j| - \hat{k}_{C_i \cup C_j}, \tag{15}$$

where $\hat{k}_{C_i \cup C_j}$ estimates the size of $(C_i \cup \bigcup_{u \in C_j} N_{G_P}(u))$ as:

$$\hat{k}_{C_i \cup C_j} = \frac{K-1}{r_K(R(C_i) \cup \mathsf{vbks}(C_j))}.$$

Here, $R(C_i)$ is the set of hash values from all nodes in C_i. By avoiding explicit enumeration, $\hat{E}(C_i, C_j)$ approximates inter-community edges, enabling fast ΔQ calculations when merging communities.

Time Complexity. Without optimization, SCAR incurs a time complexity of $O(|E|+K|V'|)$ per iteration, where $|E|$ is the number of edges and $|V'|$ is the number of nodes in the aggregated graph. In contrast, with optimization, the complexity per iteration is reduced to $O(K|V|)$, where $|V|$ is the number of original nodes. This is achieved by avoiding the reconstruction of the entire graph and instead performing localized sketch updates and modularity gain computations.

Space Complexity. Regarding space complexity, without optimization, the algorithm requires extra $O(|V'|+|E'|+K|V'|)$ space to store the aggregated graph and sketches. With optimization, the space requirement is reduced to $O(K|V|+K|C|)$, where $|C|$ is the number of communities. By eliminating the need to store the aggregated graph G', our approach substantially lowers memory usage, enabling the analysis of larger-scale networks.

Advantages. *Firstly*, eliminating explicit graph aggregation saves time and memory, especially in large networks where aggregation can be costly. *Secondly,* community sketches can be updated incrementally, making the algorithm more efficient during iterative community detection phases.

5 Experimental evaluation

5.1 Experimental Setup

Our experiments were conducted on a machine equipped with an Intel X5650 CPU and 512 GB of RAM[2]. The system is entirely memory-resident and was developed in C++. All code was compiled using GCC version 9.3.0 with the `-O3` optimization flag.

Datasets. Our experiments are conducted on five real-world heterogeneous information networks (HINs) and an additional dataset from our industry collaborator, Ant Group. Dataset statistics are summarized in Table 2. `IMDB` is a HIN that captures relationships between actors, directors, and movies. `ACM` and `DBLP` are academic HINs that represent co-authorship networks, where researchers are linked through papers and conferences. `PubMed` is a biomedical HIN, illustrating the interactions between genes, chemicals, and diseases across various species. `FreeBase` is a comprehensive general-purpose HIN, originally developed by Google, encompassing a wide range of entities and relationships. In the industrial dataset AG, the graph consists of 37 distinct node types, including Devices, Merchants, Wi-Fi access points, IP addresses, and transaction terminals. The edges represent relationships such as consumer interactions between users and merchants, shared network connections, transactions initiated from specific devices, and the co-location of

[2]The code is available at https://drive.google.com/drive/folders/1YQWet_XYArEBVcBwxKAjNpCnQOgBpG7a.

Table 2. Statistics of Real-World Datasets.

Datasets	$\|V\|$	$\|E\|$	$\|L\|$	$\|\mathcal{M}\|$	Avg. Degree	avg. $\|V_P\|$	avg. $\|E_P\|$	Type
IMDB	21.4K	86.6K	10	679	8.09	6.5K	2.2M	Movie HIN
ACM	10.9K	548K	12	40	100.73	2.6K	1.6M	Academic HIN
DBLP	26.1K	239.6K	10	48	18.37	14K	61.5M	Academic HIN
PubMed	63.1K	236.5K	14	172	7.49	9.1K	39.2M	Biomedical HIN
FreeBase	180K	1.06M	44	151	11.78	15.4K	116.1M	General HIN
AG[3]	135M	1.81B	37	48	13.30	-	-	Transaction HIN

devices. This dataset captures the complex interactions within an industrial environment, enabling the analysis of operational dynamics and connectivity patterns in large-scale systems.

Default Settings. To generate candidate meta-paths, we utilize AnyBURL [39], resulting in a set denoted by $\mathcal{M}$. We configure AnyBURL with a confidence threshold of 0.1 and capture snapshots at intervals of ten seconds. It is important to highlight that our methodologies operate independently of the meta-path generation process. As illustrated in Table 2, the column $|\mathcal{M}|$ indicates the number of meta-paths extracted for each respective dataset. Additionally, we initialize the sketch size with a default value of $K = 512$.

Comparisons. We conduct the evaluation by comparing SCAR against 7 SOTA algorithms:

- Gdy [9]: Optimizes modularity by iteratively merging.
- Louvain [3, 4, 21]: Hierarchical method for efficient modularity-based community detection.
- Leiden [55]: Enhances Louvain by ensuring well-connected and higher-quality communities.
- Infomap [43]: Compresses random walk descriptions using Map Equation to detect communities.
- Walktrap [42]: Uses short random walks to measure node similarity for hierarchical clustering.
- Networkit [50]: A toolkit offering optimized community detection algorithms for large networks.

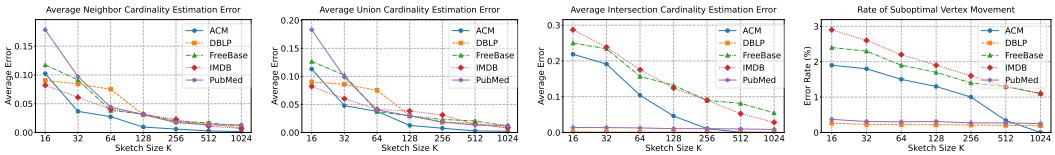

Fig. 7. Relative Estimation Errors for (a) Neighbor Cardinality, (b) Union Cardinality, and (c) Intersection Cardinality; (d) Rate of vertices failing to transition to the optimal community.

5.2 Characteristics of SCAR

We first evaluate the characteristics of SCAR, including the impact of K and the estimation error. Figure 7 illustrates the relative estimation error for three key tasks: neighbor cardinality, union cardinality, and inter-community connectivity estimation, across different values of the sketch size K. The results demonstrate that the estimation error consistently decreases as K increases from 32 to 1024. Finally, we assess the effect of different K values on modularity.

Firstly, for both neighbor cardinality and union cardinality estimations, the error trends are similar. Across all tested datasets, when $K \geq 256$, the estimation error falls below 2.0%. Further increasing K to 1024 nearly eliminates the error, bringing it close to 0%.

Secondly, In the case of inter-community connectivity estimation, the error is slightly higher than the other two tasks, due to the inclusion-exclusion principle, which accumulates errors from two vbks estimations. Nonetheless, when K is set to 1024, the error in inter-community connectivity estimation reduces to approximately 2.0% for most datasets. An exception is observed in the FreeBase, where the relative error remains around 5.0%. This higher error in FreeBase is primarily

[3]The graph is too large to be explicitly materialized in memory.

Table 3. Effect of K Parameter on Modularity and Runtime.

Dataset	Metric	16	32	64	128	256	512	1024
IMDB	Modularity	0.15	0.16	0.16	0.18	0.19	0.20	0.21
	Time (s)	0.20	0.27	0.32	0.35	0.42	0.67	0.96
ACM	Modularity	0.19	0.20	0.22	0.22	0.29	0.31	0.35
	Time (s)	0.07	0.09	0.11	0.14	0.18	0.21	0.30
DBLP	Modularity	0.05	0.10	0.10	0.18	0.21	0.22	0.24
	Time (s)	0.46	0.62	0.90	1.00	1.03	1.46	1.68
PubMed	Modularity ($\times 10^{-3}$)	0.55	0.95	1.00	1.10	1.30	1.40	3.10
	Time (s)	0.06	0.30	0.56	0.95	1.93	3.47	3.72
FreeBase	Modularity ($\times 10^{-2}$)	0.00	1.60	1.60	2.00	2.60	2.60	4.60
	Time (s)	0.61	1.28	2.15	4.26	8.23	15.40	25.70
AG[4]	Modularity	0.45	0.47	0.47	0.49	0.52	0.56	0.58
	Time (s)	89.90	109.00	168.00	284.00	553.00	1088.00	1660.00

because the intersection sizes are relatively small, resulting in larger relative estimation errors. Despite this, increasing K to 1024 still significantly mitigates the error.

Impact of Sketch Size K on Modularity and Runtime. We randomly selected 10 meta-paths from each dataset for evaluation. As shown in Table 3, the modularity consistently increases with larger K values, indicating that a higher K leads to better-detected communities. For example, on DBLP, modularity increases from 0.05 at $K = 16$ to 0.24 at $K = 1024$; similarly, on ACM, it rises from 0.19 to 0.35, and on AG, from 0.45 to 0.58. On the other hand, the runtime exhibits an almost linear growth with K (e.g., AG runs in 89.90 seconds at $K = 16$ and 1660.00 seconds at $K = 1024$). Based on our experiments, setting $K = 512$ provides a favorable trade-off between achieving high modularity and maintaining an acceptable runtime.

Suboptimal Vertex Movement Rate and Sketch Size. Figure 7(d) illustrates the suboptimal vertex movement rate—as the percentage of vertices that are not moved into the community with the highest modularity gain—plotted against the sketch size K. The y-axis represents the error rate, quantifying the frequency at which a vertex is left in a suboptimal community when the true modularity gains between the best and second-best communities are nearly identical. Observations reveal that, across all datasets, the error rate consistently decreases as K increases. Specifically, ACM and DBLP show nearly zero error for $K \geq 256$, Freebase starts with a higher error rate and stabilizes more slowly, IMDB exhibits intermediate behavior, and PubMed presents the highest initial error rate, requiring $K \geq 512$ to achieve similar stability. These results indicate that while a smaller K leads to a higher rate of suboptimal vertex movement, increasing K improves the accuracy of vertex assignments, thereby enhancing the overall community detection quality.

Graph Sparsity Analysis. Table 3 highlights distinct trends between dense and sparse graphs as the sketch size K varies. Dense graphs, represented by datasets such as ACM and DBLP, achieve near-optimal modularity even at small K (e.g., $K = 256$), indicating that limited sketching is sufficient to capture their community structure. In contrast, sparse graphs require larger K (e.g., $K = 1024$) to improve modularity estimation; among these, PubMed is the sparsest, followed by IMDB and then FreeBase. This suggests that while denser graphs can be effectively processed with a small sketch size, sparser graphs demand higher K values to yield reliable community detection.

[4]For AG, the graph cannot be materialized; hence, the error rate is not available.

5.3 Efficiency

Table 4. Running Time (in seconds) of Various Community Detection Algorithms on Different Datasets. **Abbreviations**: **Sk.** = Sketching Time; **CD.** = Community Detection Time; **GM.** = Graph Materialization Time; **Total** = **Sk.** + **CD.** for SCAR, **GM.** + **CD.** for others; **TLE** = Time Limit Exceeded; **OOM** = Out of Memory.

Dataset	SCAR (ours)				GM.	Gdy		Leiden		Infomap		Walktrap		Networkit		Louvain	
	Sk.	CD.	Total	Speedup vs. 2nd		CD.	Total	CD.	Total	CD.	Total	CD.	Total	CD.	Total	CD.	Total
IMDB	0.36	2.45	**2.81**	4.70x ⇑	12.56	36.86	49.42	1.79	14.35	8.54	21.10	165.97	178.53	3.14	15.70	0.64	13.20
ACM	0.42	0.29	**0.71**	93.91x ⇑	66.58	26.52	93.10	1.00	67.58	4.27	70.85	54.25	120.83	1.75	68.33	0.10	66.68
DBLP	0.95	2.25	**3.20**	158.45x ⇑	516.69	5284.32	5801.01	51.60	568.29	171.98	688.67	9440.85	9957.54	195.23	711.92	4.60	521.29
PubMed	1.03	9.61	**10.64**	27.91x ⇑	293.39	1481.27	1774.66	32.66	326.05	148.14	441.53	TLE	TLE	109.72	347.64	3.53	296.92
FreeBase	2.21	20.06	**22.27**	47.80x ⇑	1053.95	TLE	TLE	45.44	1099.39	432.24	1486.19	TLE	TLE	612.29	1666.24	10.58	1064.53
AG	3.14	291.86	**295.00**	NA	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM

Table 5. Peak Memory Usage (in GB) on Different Datasets.

Dataset	SCAR	Gdy	Leiden	Infomap	Walktrap	Networkit	Louvain
IMDB	**0.41**	3.14	2.19	21.33	5.05	2.12	1.18
ACM	0.33	0.88	0.69	1.06	1.42	**0.27**	0.32
DBLP	**0.51**	17.96	14.49	20.24	18.52	2.04	6.50
PubMed	**1.22**	16.10	14.38	29.72	TLE	2.24	6.16
FreeBase	**6.33**	TLE	85.93	99.01	TLE	15.83	48.93
AG	**59.22**	OOM	OOM	OOM	OOM	OOM	OOM

Sketching Construction Time vs. Graph Materialization Time. We compare the sketching construction time of SCAR with the graph materialization time (**GM.**) required by the baseline methods. As shown in Table 4, SCAR significantly reduces the preprocessing time across all datasets. For instance, on the IMDB dataset, SCAR completes sketching in 0.36 seconds, whereas the baselines require 12.56 seconds for graph materialization. This represents a **97.1%** reduction in preprocessing time. Similarly, on the larger FreeBase dataset, SCAR's sketching time is 2.21 seconds, compared to the baselines' 1053.95 seconds, achieving a substantial **99.8%** reduction. These improvements are primarily due to SCAR's use of compact vertex-based sketches, which efficiently capture essential graph properties without storing the entire graph in memory. Unlike baseline methods that require full graph materialization, SCAR minimizes the amount of data that needs to be stored and processed, leading to faster preprocessing times and lower memory consumption.

The reduction in preprocessing time is attributed to SCAR's efficient sketching. By constructing compact vertex-based sketches, SCAR captures essential graph properties without needing to materialize the entire graph. This eliminates the overhead associated with graph construction and storage, especially in large-scale datasets.

Efficiency Improvement. We assess the overall efficiency by comparing the total running time of SCAR (sketching time + community detection time) with that of the baseline methods (graph materialization time + community detection time). Table 4 highlights that SCAR consistently outperforms the baselines in total execution time.

On the IMDB dataset, SCAR completes in 2.81 seconds, while the fastest baseline, Louvain, takes 13.20 seconds. This results in a **78.6%** reduction in total running time. For the ACM dataset, SCAR achieves a total time of 0.71 seconds, compared to 66.68 seconds for Louvain, marking a **89.4%** reduction. The performance gap widens with larger datasets. On DBLP, SCAR completes in 3.20 seconds, whereas Louvain requires 521.29 seconds, leading to a **99.4%** reduction in total time. For the FreeBase dataset, SCAR finishes in 22.27 seconds, significantly faster than Louvain's 1064.53 seconds, which is a **97.8%** reduction.

Reasons for Improvement. The substantial efficiency gains of SCAR are primarily attributed to three key factors. First, SCAR eliminates the need for graph materialization by operating directly on sketches, thereby avoiding the costly preprocessing step required by baseline methods. This reduction in preprocessing time significantly lowers the overall execution time, especially for large-scale datasets. Second, the community detection algorithms employed by SCAR are optimized for sketch-based representations, which reduces computational complexity and enhances processing speed. Lastly, using compact sketches decreases the amount of data that needs to be processed, leading to faster execution times. These combined factors not only contribute to the remarkable speedups observed in our experiments but also highlight SCAR's strengths in scalability and resource efficiency, making it highly suitable for real-world applications involving large and complex graphs.

Memory Usage. We evaluate the peak memory consumption during the execution of each algorithm across different datasets. Table 5 presents the memory usage (in GB) for SCAR and the baseline methods. SCAR consistently demonstrates lower memory footprints compared to the baselines. For instance, on the IMDB dataset, SCAR uses 0.41 GB of RAM, whereas Gdy and Infomap consume 3.14 GB and 21.33 GB, respectively. Similarly, on the ACM dataset, SCAR requires only 0.33 GB, compared to 0.88 GB for Gdy and 1.06 GB for Infomap. The trend continues with the DBLP dataset, where SCAR utilizes 0.51 GB, significantly lower than 17.96 GB by Gdy and 20.24 GB by Infomap. On the PubMed dataset, SCAR consumes 1.22 GB, whereas Gdy and Infomap require 16.10 GB and 29.72 GB, respectively. For the large-scale FreeBase dataset, SCAR maintains a memory usage of 6.33 GB, in stark contrast to 99.01 GB by Infomap and 48.93 GB by Louvain. Notably, on the largest dataset, AG, SCAR is the only method able to process the data without running out of memory, using 59.22 GB, while all other methods exceeded the available memory (marked as "OOM" in Table 5). These results indicate that SCAR achieves substantial memory savings, enabling it to scale to larger datasets without necessitating additional hardware resources.

This significant reduction in memory consumption is primarily due to SCAR's use of compact vertex-based sketches, which capture essential graph properties without the need to store the entire graph in memory. Unlike baseline methods that require full graph materialization, SCAR minimizes the amount of data stored and processed by leveraging compact sketch representations. Additionally, the algorithms within SCAR utilize memory-efficient data structures optimized for sketch-based computations, further reducing memory overhead. These advantages contribute to SCAR's strengths in scalability, resource efficiency, and performance stability, making it highly suitable for handling large-scale datasets in real-world applications.

5.4 Effectiveness

We evaluate the effectiveness of SCAR by comparing its modularity scores against established baseline community detection algorithms across multiple datasets, as presented in Table 6.

Across various datasets, SCAR achieves modularity scores up to 88.6% of the highest scores attained by leading algorithms. On the IMDB dataset, SCAR's modularity is about 71.4% of Louvain's 0.28. In the ACM dataset, SCAR maintains a modularity score that is roughly 88.6% of Louvain's 0.35. For the DBLP dataset, SCAR matches the performance of Gdy and achieves nearly 88.0% of Louvain's 0.25.

However, on larger datasets such as PubMed and FreeBase, SCAR's modularity scores are more significantly lower, representing approximately 3.0% and 37.7% of Louvain's scores, respectively. Despite these reductions, SCAR offers a balanced approach by maintaining reasonable community detection quality while providing substantial advantages in execution time and memory consumption. This balance makes SCAR a practical tool for large-scale and resource-constrained environments where computational efficiency is paramount. Furthermore, on extremely large graphs like AG, most baseline algorithms fail to complete due to computational constraints such as

Table 6. Modularity Scores of Various Community Detection Algorithms on Different Datasets.

Dataset	SCAR	Gdy	Leiden	Infomap	Walktrap	Networkit	Louvain
IMDB	0.20	0.23	0.20	0.18	0.19	0.26	0.28
ACM	0.31	0.32	0.29	0.33	0.33	0.35	0.35
DBLP	0.22	0.22	0.21	0.04	0.24	0.24	0.25
PubMed ($\times 10^{-3}$)	1.00	29.00	0.26	0.04	TLE	33.00	33.00
FreeBase ($\times 10^{-2}$)	2.60	TLE	1.20	1.80	TLE	6.90	6.90
AG	0.56	OOM	OOM	OOM	OOM	OOM	OOM

Table 7. Modularity Scores when Directly Running Community Detection on Heterogeneous Graphs.

Dataset	SCAR	Gdy	Leiden	Infomap	Walktrap	Networkit	Louvain
IMDB	**0.35**	0.14	0.04	0.01	0.08	0.08	0.05
ACM	**0.40**	0.28	0.01	0.00	0.01	0.01	0.30
DBLP	**0.54**	0.22	0.01	0.04	0.15	0.15	0.17
PubMed ($\times 10^{-2}$)	**1.29**	0.78	0.01	0.72	0.50	0.50	0.40
FreeBase ($\times 10^{-2}$)	**4.66**	1.05	0.01	1.52	1.46	1.45	1.24

exceeding memory limits or timeouts. In contrast, SCAR successfully processes these large-scale networks, delivering modularity estimates of 0.56. This capability underscores SCAR's scalability and robustness, demonstrating its unique advantage in handling complex and expansive graph structures that are otherwise infeasible for traditional community detection methods.

While SCAR does not always achieve the highest modularity scores compared to all baselines, it consistently provides competitive results across various datasets. These outcomes highlight SCAR's balance between community detection effectiveness and its significant advantages in execution time and memory consumption, making it a practical tool for large-scale and complex networks where computational resources are a critical constraint.

Table 8. Relative Difference in Modularity Gain (Δ/best) at Different Thresholds. Δ is the difference in modularity gain between assigning a node to its best vs. second-best community, and best is the modularity gain obtained by moving the node to its best community.

Dataset	$\Delta_{\min}$	**2%**	**2%-4%**	**4%-6%**	**6%-8%**	**8%-10%**
ACM	3.54×10^{-11}	1.10%	0.12%	0.09%	0.05%	0.08%
DBLP	1.83×10^{-13}	0.37%	0.08%	0.07%	0.07%	0.07%
FreeBase	7.58×10^{-12}	0.76%	0.24%	0.17%	0.15%	0.15%
IMDB	1.26×10^{-12}	1.00%	0.21%	0.18%	0.17%	0.15%
PubMed	2.01×10^{-16}	0.37%	0.16%	0.13%	0.12%	0.11%

Relative Difference in Modularity Gain. Table 8 presents the relative difference in modularity gain (Δ/best) across various thresholds ranging from 2% to 10%. Although the minimum $\Delta_{\min}$ values are extremely small, indicating that only a few nodes have negligible differences in modularity gain between their best and second-best community assignments, the proportion of such nodes is minimal. For instance, only 1.10% of nodes in the ACM dataset and 0.37% in DBLP exhibit a relative difference of 2% or lower. Similarly, IMDB and PubMed show low percentages (1.00% and 0.37%, respectively) within the same threshold. This suggests that the majority of nodes experience higher relative differences in modularity gain, meaning that the gap between their best and second-best community assignments is substantial. Hence, while a few nodes achieve near-optimal community

Table 9. Impact of Noise on DBLP: The numbers represent NMI after removing a certain percentage of edges.

Noise Ratio	SCAR	Gdy	Leiden	Infomap	Walktrap	Networkit	Louvain
2%	0.87	0.87	0.95	0.93	0.95	0.85	0.91
4%	0.88	0.87	0.97	0.91	0.95	0.85	0.93
6%	0.87	0.84	0.96	0.94	0.95	0.85	0.93
8%	0.82	0.82	0.94	0.88	0.93	0.84	0.91
10%	0.82	0.82	0.94	0.86	0.90	0.82	0.90

assignments with minimal Δ, the overall estimation process maintains controlled error margins without significant misassignments, as demonstrated in Tables 3 and Figure 7(d).

Comparison with Homogenization-based HIN Community Detection. Existing approaches [61] for community detection on HINs typically treat the entire network as a homogeneous graph and then partition nodes of the same type into communities. As a consequence, if two meta-paths share the same source type, these methods yield identical community detection results. To ensure a fair evaluation of our approach, we randomly select one meta-path from each collection of meta-paths that share the same source type for testing. Table 7 shows the modularity scores obtained by SCAR and six baseline methods when applied directly to the heterogeneous graphs. As demonstrated, SCAR consistently achieves higher modularity scores across various datasets—e.g., achieving 0.54 on DBLP compared to 0.22 by Gdy and 0.17 by Louvain—thereby more effectively capturing the intrinsic community structure in these networks.

Impact of Noise. Real-world graphs often contain noise, which can degrade the quality of community detection. To assess the robustness of various methods under noisy conditions, we employ Normalized Mutual Information (NMI) [12] as a similarity metric between clustering results. NMI measures the amount of shared information between two clusterings and is defined as:$\text{NMI}(A, B) = \frac{2\, I(A;B)}{H(A)+H(B)}$, where $I(A; B)$ is the mutual information between clusterings A and B, and $H(\cdot)$ denotes the entropy. Higher NMI values indicate that the detected communities more closely match the original structure.

Our experimental evaluation demonstrates that SCAR exhibits strong robustness to noise, effectively retaining community structure even as the network is subjected to significant edge removals. As shown in Table 9, when the noise ratio increases from 2% to 10%, SCAR consistently maintains high NMI values—indicating minimal degradation in community quality. In contrast, existing algorithms such as Gdy and Networkit show comparatively lower NMI scores under identical noise conditions. These results confirm that SCAR not only provides computational and memory efficiency but also delivers enhanced robustness against noise.

5.5 Case study

In this case study, we evaluate the effectiveness of SCAR on the DBLP dataset using the Author-Paper-Author (APA) meta-path. The DBLP dataset is a comprehensive bibliographic database covering major computer science journals and conference proceedings, providing detailed information about authors, publications, and their relationships. By focusing on the APA meta-path, we construct a homogeneous information network that encapsulates meaningful connections among authors based on their collaborative publications.

We applied all seven community detection algorithms—SCAR and six baselines (Gdy, Infomap, Walktrap, NetworKit, Louvain, and Leiden)—to the DBLP-derived graph. We used the NMI score to compare the resulting community partitions (Figure 8), where a higher NMI indicates a high degree of similarity between the partitions. Our results show that, except for Infomap, SCAR and the other methods achieve NMI scores above 0.8. Notably, SCAR and Louvain obtain an NMI of 0.83,

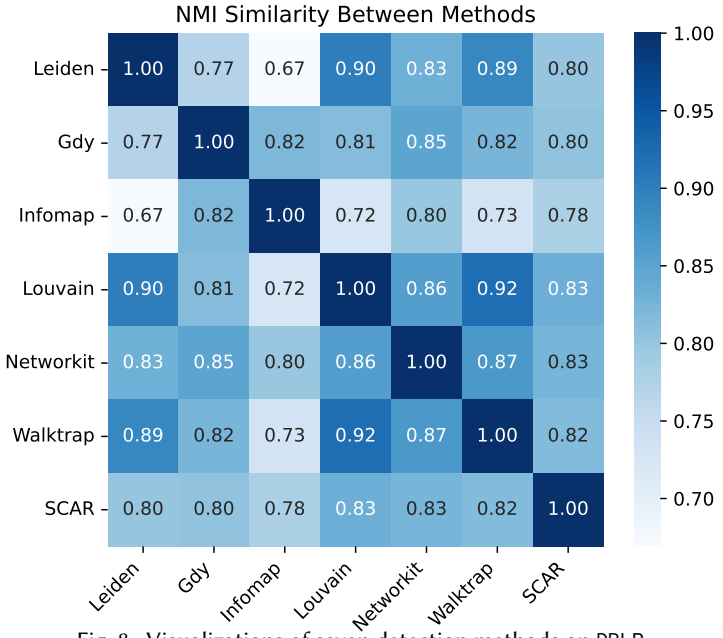

Fig. 8. Visualizations of seven detection methods on DBLP.

demonstrating that the community structure detected by SCAR closely matches that of Louvain. Overall, these findings confirm that SCAR produces community detection results of comparable quality to established algorithms, while simultaneously enabling a significant speedup (by tens of times), making it a robust and practical choice for large-scale applications.

6 Related work

Community Detection. Community detection is fundamental for uncovering the underlying structure of networks, providing insights into their functional and organizational properties. Traditional methods, primarily developed for homogeneous graphs, include modularity optimization [4, 40, 55], dense subgraph discovery [5, 17, 36], label propagation [29, 49, 60], k-truss [18, 19, 25], k-core [26, 28], and k-clique [32, 56]. Additionally, algorithms such as [3, 4, 9, 21, 42, 43, 50, 55] have been widely adopted for efficient modularity-based community detection in large-scale networks. While these algorithms demonstrate effectiveness in homogeneous settings, they struggle with the heterogeneous aspects of real-world networks, such as multiple node and edge types.

Community Detection and Graph Materialization in Heterogeneous Networks. The complexity of HINs, characterized by diverse node and edge types, necessitates specialized community detection approaches. Existing methods often rely on meta-path-based graph transformations or advanced modeling techniques to capture the rich semantics inherent in HINs [33, 51, 53]. Tang et al. [54] require pre-constructed multi-dimensional graphs, restricting flexibility for diverse scenarios and differing from SCAR's model. Zhang and Chen [7, 61] introduced a modularity definition for heterogeneous communities, resulting in mixed node types within communities, whereas SCAR focuses on homogeneous communities for clearer applicability. Additionally, Chen et al. [7] and Liu et al. [35] rely on materializing V^2 similarity matrices, leading to excessive storage demands and out-of-memory (OOM) errors on large datasets, which severely limits their scalability. Recent studies have focused on improving the efficiency of relational graph materialization. For instance, Guo et al. [16] developed an algorithm employing boolean matrix multiplication to materialize relational graphs, optimizing for areas with local density. Similarly, Chatzopoulos et al. [6] introduced

a technique for enumerating instances across various meta-paths by sharing the computational workload. Additionally, the utilization of worst-case optimal join algorithms—such as those documented in Ngo et al. [41], Freitag et al. [15], Arroyuelo et al. [1], and Veldhuizen [57]—facilitates the materialization of relational graphs by efficiently traversing meta-path instances while avoiding redundant node visits. Despite these advancements, materialization methods still suffer from significant resource overheads, making them less practical for industrial-scale applications. These methods may require substantial memory resources, and the time taken for materialization often becomes a computational bottleneck. In our experiments, graph materialization accounted for over 99.0% of the total runtime, highlighting the impracticality of constructing the graph prior to community detection in large-scale settings.

Comparative Analysis. Traditional community detection algorithms such as Louvain [3, 4, 21], and Leiden [55] excel in modularity optimization for homogeneous networks but falter in heterogeneous settings due to their inability to capture multi-typed relationships efficiently. Algorithms like Infomap [43] and Walktrap [42] utilize random walks and hierarchical clustering to identify communities but do not inherently support HINs without significant modifications.

In contrast, SCAR outperforms these existing methods by eliminating the need for explicit graph aggregation and leveraging recursive sketch computations. This results in substantial reductions in both runtime and memory usage. While existing methods effectively capture intricate community structures, they do not address the computational challenges HINs pose. Our approach maintains high accuracy in community detection while efficiently handling the complexities of heterogeneous networks, as evidenced by our experimental results, where graph materialization in traditional methods accounted for over 99.0% of the total runtime. Additionally, our framework achieves up to a 93.6% reduction in memory consumption, making it both scalable and resource-efficient.

7 Conclusion and Future Work

In this paper, we introduced SCAR, a novel community detection algorithm that leverages compact vertex-based sketches to efficiently identify community structures within large-scale graphs. Through extensive experiments on diverse real-world datasets, SCAR consistently outperformed established baseline methods in both execution time and memory consumption. Specifically, SCAR achieved up to a 158.45x speedup in total running time and reduced memory usage by up to **98%** compared to the fastest baseline, Louvain. These significant improvements are primarily due to SCAR's ability to bypass the costly graph materialization step and operate directly on compact sketches, thereby minimizing computational overhead and optimizing resource utilization. The scalability and resource efficiency demonstrated by SCAR make it a highly effective tool for community detection in large and complex graphs, enabling timely and resource-conscious analysis in various application domains such as social networks, bioinformatics, and recommendation systems.

Building on the promising results of SCAR, future research can explore several avenues to enhance its capabilities and applicability further. A key avenue is to explicitly incorporate node attributes and edge weights into the community detection process. This extension aims to avoid over- or underestimating communities by leveraging richer graph information. In addition, one potential direction is extending SCAR to handle dynamic and evolving graphs, enabling real-time community detection as the graph structure changes.

Acknowledgement. This research is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG2-TC-2021-002), the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 (Award No: MOE2019-T2-2-065), the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant, 23-SIS-SMU-063, and the National Key Research and Development Plan of China (2023YFB4502305). Yuchen Li is partially supported by the Lee Kong Chian fellowship.

References

[1] Diego Arroyuelo, Aidan Hogan, Gonzalo Navarro, Juan L. Reutter, Javiel Rojas-Ledesma, and Adrián Soto. 2021. Worst-Case Optimal Graph Joins in Almost No Space. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 102–114.

[2] Jeff Baumes, Mark Goldberg, Malik Magdon-Ismail, and William Al Wallace. 2004. Discovering hidden groups in communication networks. In *International Conference on Intelligence and Security Informatics*. Springer, 378–389.

[3] Vincent Blondel, Jean-Loup Guillaume, and Renaud Lambiotte. 2023. Fast unfolding of communities in large networks: 15 years later. *arXiv preprint arXiv:2311.06047* (2023).

[4] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment* 2008, 10 (2008), P10008.

[5] Moses Charikar. 2000. Greedy approximation algorithms for finding dense components in a graph. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*. Springer, 84–95.

[6] Serafeim Chatzopoulos, Thanasis Vergoulis, Dimitrios Skoutas, Theodore Dalamagas, Christos Tryfonopoulos, and Panagiotis Karras. 2023. Atrapos: Real-time Evaluation of Metapath Query Workloads. In *Proceedings of the ACM Web Conference 2023 (WWW '23)*. Association for Computing Machinery, New York, NY, USA, 2487–2498.

[7] Limin Chen, Yan Zhang, and Liu Yang. 2019. Semi-supervised meta-path-based algorithm for community detection in heterogeneous information networks. In *Web Information Systems and Applications: 16th International Conference, WISA 2019, Qingdao, China, September 20-22, 2019, Proceedings 16*. Springer, 506–511.

[8] Yuhang Chen, Jiaxin Jiang, Shixuan Sun, Bingsheng He, and Min Chen. 2024. RUSH: Real-time Burst Subgraph Detection in Dynamic Graphs. *Proceedings of the VLDB Endowment* 17, 11 (2024), 3657–3665. doi:10.14778/3681954.3682028

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

[10] Edith Cohen. 1997. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. System Sci.* 55, 3 (1997), 441–453.

[11] Edith Cohen and Haim Kaplan. 2007. Summarizing data using bottom-k sketches. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*. 225–234.

[12] Leon Danon, Albert Diaz-Guilera, Jordi Duch, and Alex Arenas. 2005. Comparing community structure identification. *Journal of statistical mechanics: Theory and experiment* 2005, 09 (2005), P09008.

[13] Herbert A David and Haikady N Nagaraja. 2004. *Order statistics*. John Wiley & Sons.

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

[15] Michael J. Freitag, Maximilian Bandle, Tobias Schmidt, Alfons Kemper, and Thomas Neumann. 2020. Adopting Worst-Case Optimal Joins in Relational Database Systems. *Proc. VLDB Endow.* 13, 11 (2020), 1891–1904.

[16] Yucan Guo, Chenhao Ma, and Yixiang Fang. 2024. Efficient Core Decomposition over Large Heterogeneous Information Networks. In *40th IEEE International Conference on Data Engineering (ICDE)*. IEEE, to appear.

[17] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. 2016. Fraudar: Bounding graph fraud in the face of camouflage. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 895–904.

[18] Jinbin Huang, Xin Huang, and Jianliang Xu. 2020. Truss-based structural diversity search in large graphs. *IEEE Transactions on Knowledge and Data Engineering* 34, 8 (2020), 4037–4051.

[19] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying k-truss community in large and dynamic graphs. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (Snowbird, Utah, USA) *(SIGMOD '14)*. Association for Computing Machinery, New York, NY, USA, 1311–1322. doi:10.1145/2588555.2610495

[20] Xin Huang, Jiaxin Jiang, Byron Choi, Jianliang Xu, Zhiwei Zhang, and Yunya Song. 2018. PP-DBLP: Modeling and Generating Attributed Public-Private Networks with DBLP. In *ICDM*. IEEE.

[21] Lucas G. S. Jeub, Marya Bazzi, Inderjit S. Jutla, and Peter J. Mucha. 2017. A generalized Louvain method for community detection implemented in MATLAB. https://github.com/GenLouvain/GenLouvain. Accessed: insert-date-here.

[22] Jiaxin Jiang, Yuhang Chen, Bingsheng He, Min Chen, and Jia Chen. 2024. Spade+: A Generic Real-Time Fraud Detection Framework on Dynamic Graphs. *IEEE Transactions on Knowledge and Data Engineering* (2024).

[23] Jiaxin Jiang, Xin Huang, Byron Choi, Jianliang Xu, Sourav S. Bhowmick, and Lyu Xu. 2020. PPKWS: An Efficient Framework for Keyword Search on Public-Private Networks.

[24] Jiaxin Jiang, Yuan Li, Bingsheng He, Bryan Hooi, Jia Chen, and Johan Kok Zhi Kang. 2022. Spade: a real-time fraud detection framework on evolving graphs. *Proceedings of the VLDB Endowment* 16, 3 (2022), 461–469.

[25] Yuli Jiang, Xin Huang, and Hong Cheng. 2021. I/O efficient k-truss community search in massive graphs. *The VLDB Journal* 30, 5 (2021), 713–738.

[26] Wissam Khaouid, Marina Barsky, Venkatesh Srinivasan, and Alex Thomo. 2015. K-core decomposition of large networks on a single PC. *Proceedings of the VLDB Endowment* 9, 1 (2015), 13–23.

[27] B Klimat. 2004. The enron corpus: A new datasct tor email classification rcscarch. In *Proceedings of the European Conference on Machinc Learning (ECML). 2004.*

[28] Yi-Xiu Kong, Gui-Yuan Shi, Rui-Jie Wu, and Yi-Cheng Zhang. 2019. k-core: Theories and applications. *Physics Reports* 832 (2019), 1–32.

[29] Yusuke Kozawa, Toshiyuki Amagasa, and Hiroyuki Kitagawa. 2017. Gpu-accelerated graph clustering via parallel label propagation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 567–576.

[30] Jonathan Kuck, Honglei Zhuang, Xifeng Yan, Hasan Cam, and Jiawei Han. 2015. Query-based outlier detection in heterogeneous information networks. In *Advances in database technology: proceedings. International Conference on Extending Database Technology*, Vol. 2015. NIH Public Access, 325.

[31] Laks VS Lakshmanan. 2022. On a quest for combating filter bubbles and misinformation. In *Proceedings of the 2022 International Conference on Management of Data*. 2–2.

[32] Ronghua Li, Sen Gao, Lu Qin, Guoren Wang, Weihua Yang, and Jeffrey Xu Yu. 2020. Ordering Heuristics for k-clique Listing. *Proc. VLDB Endow.* (2020).

[33] Xiang Li, Danhao Ding, Ben Kao, Yizhou Sun, and Nikos Mamoulis. 2021. Leveraging meta-path contexts for classification in heterogeneous information networks. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 912–923.

[34] Xuankun Liao, Qing Liu, Jiaxin Jiang, Xin Huang, Jianliang Xu, and Byron Choi. 2022. Distributed D-core decomposition over large directed graphs. *Proc. VLDB Endow.* 15, 8 (apr 2022), 1546–1558. doi:10.14778/3529337.3529340

[35] Dongjiang Liu, Leixiao Li, and Zhiqiang Ma. 2020. A community detection algorithm for heterogeneous information networks. *IEEE Access* 8 (2020), 195655–195663.

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

[37] Malik Magdon-Ismail, Mark Goldberg, William Wallace, and David Siebecker. 2003. Locating hidden groups in communication networks using hidden markov models. In *Intelligence and Security Informatics: First NSF/NIJ Symposium, ISI 2003, Tucson, AZ, USA, June 2–3, 2003 Proceedings 1*. Springer, 126–137.

[38] Albert W Marshall and Ingram Olkin. 1960. Multivariate chebyshev inequalities. *The Annals of Mathematical Statistics* (1960), 1001–1014.

[39] Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. 2019. Anytime Bottom-Up Rule Learning for Knowledge Graph Completion. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 3137–3143.

[40] Mark EJ Newman. 2006. Modularity and community structure in networks. *Proceedings of the national academy of sciences* 103, 23 (2006), 8577–8582.

[41] Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. 2018. Worst-case Optimal Join Algorithms. *J. ACM* 65, 3, Article 16 (2018), 40 pages.

[42] Pascal Pons and Matthieu Latapy. 2005. Computing communities in large networks using random walks. In *Computer and Information Sciences-ISCIS 2005: 20th International Symposium, Istanbul, Turkey, October 26-28, 2005. Proceedings 20*. Springer, 284–293.

[43] Martin Rosvall and Carl T Bergstrom. 2008. Maps of random walks on complex networks reveal community structure. *Proceedings of the national academy of sciences* 105, 4 (2008), 1118–1123.

[44] Mahsa Seifikar, Saeed Farzi, and Masoud Barati. 2020. C-blondel: an efficient louvain-based dynamic community detection algorithm. *IEEE Transactions on Computational Social Systems* 7, 2 (2020), 308–318.

[45] Jiaxing Shang, Lianchen Liu, Feng Xie, Zhen Chen, Jiajia Miao, Xuelin Fang, and Cheng Wu. 2014. A real-time detecting algorithm for tracking community structure of dynamic networks. *arXiv preprint arXiv:1407.2683* (2014).

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

[47] Chuan Shi, Zhiqiang Zhang, Ping Luo, Philip S Yu, Yading Yue, and Bin Wu. 2015. Semantic path based personalized recommendation on weighted heterogeneous information networks. In *Proceedings of the 24th ACM international on conference on information and knowledge management*. 453–462.

[48] Kijung Shin, Bryan Hooi, Jisu Kim, and Christos Faloutsos. 2017. Densealert: Incremental dense-subtensor detection in tensor streams. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1057–1066.

[49] Jyothish Soman and Ankur Narang. 2011. Fast community detection algorithm with gpus and multicore architectures. In *2011 IEEE International Parallel & Distributed Processing Symposium*. IEEE, 568–579.

[50] Christian L Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. 2016. NetworKit: A tool suite for large-scale complex network analysis. *Network Science* 4, 4 (2016), 508–530.
[51] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. 2011. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment* 4, 11 (2011), 992–1003.
[52] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. 2022. Heterogeneous information networks: the past, the present, and the future. *Proceedings of the VLDB Endowment* 15, 12 (2022).
[53] Yizhou Sun, Brandon Norick, Jiawei Han, Xifeng Yan, Philip S Yu, and Xiao Yu. 2013. Pathselclus: Integrating meta-path selection with user-guided object clustering in heterogeneous information networks. *ACM transactions on knowledge discovery from data (TKDD)* 7, 3 (2013), 1–23.
[54] Lei Tang, Xufei Wang, and Huan Liu. 2012. Community detection via heterogeneous interaction analysis. *Data mining and knowledge discovery* 25 (2012), 1–33.
[55] Vincent A Traag, Ludo Waltman, and Nees Jan Van Eck. 2019. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific reports* 9, 1 (2019), 5233.
[56] Charalampos Tsourakakis. 2015. The k-clique densest subgraph problem. In *Proceedings of the 24th international conference on world wide web*. 1122–1132.
[57] Todd L. Veldhuizen. 2014. Triejoin: A Simple, Worst-Case Optimal Join Algorithm. In *Proceedings of the 17th International Conference on Database Theory (ICDT)*. OpenProceedings.org, 96–106.
[58] Yixing Yang, Yixiang Fang, Xuemin Lin, and Wenjie Zhang. 2020. Effective and Efficient Truss Computation over Large Heterogeneous Information Networks. In *36th IEEE International Conference on Data Engineering (ICDE)*. IEEE, 901–912.
[59] Siyuan Yao, Yuchen Li, Shixuan Sun, Jiaxin Jiang, and Bingsheng He. 2024. uBlade: Efficient Batch Processing for Uncertainty Graph Queries. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–24.
[60] Chang Ye, Yuchen Li, Bingsheng He, Zhao Li, and Jianling Sun. 2021. GPU-Accelerated Graph Label Propagation for Real-Time Fraud Detection. In *Proceedings of the 2021 International Conference on Management of Data*. 2348–2356.
[61] Jingfei Zhang and Yuguo Chen. 2020. Modularity based community detection in heterogeneous networks. *Statistica Sinica* 30, 2 (2020), 601–629.
[62] Liang Zhou, Dan Wu, Zhenjiang Dong, and Xuelong Li. 2017. When collaboration hugs intelligence: Content delivery over ultra-dense networks. *IEEE Communications Magazine* 55, 12 (2017), 91–95.
[63] Yingli Zhou, Yixiang Fang, Wensheng Luo, and Yunming Ye. 2023. Influential Community Search over Large Heterogeneous Information Networks. *Proceedings of the VLDB Endowment* 16, 8 (2023), 2047–2060.
[64] Di Zhuang, J Morris Chang, and Mingchen Li. 2019. DynaMo: Dynamic community detection by incrementally maximizing modularity. *IEEE Transactions on Knowledge and Data Engineering* 33, 5 (2019), 1934–1945.

Received October 2024; revised January 2025; accepted February 2025