Efficient Core Decomposition over Large Heterogeneous Information Networks

Yucan Guo, Chenhao Ma†, Yixiang Fang
The Chinese University of Hong Kong, Shenzhen
{guoyucan, machenhao, fangyixiang}@cuhk.edu.cn

***Abstract*—Core decomposition is a critical metric for evaluating the vertex importance and analyzing graph structure. Given a graph G, a k-core is the largest subgraph of G where each vertex has at least k neighbors. Most existing works mainly focus on homogeneous graphs in which edges are of the same type and cannot be applied to heterogeneous information networks (HINs) directly. However, most real-world networks are HINs which consist of different vertex types and edge types.**

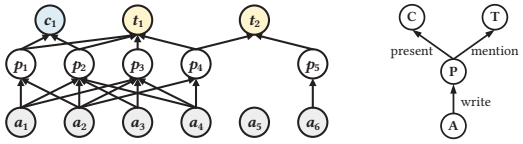
To reveal the cohesive subgraphs with hierarchical relations on HINs, we adopt the well-known $(k, \mathcal{P})$-core model to compute coreness over HINs, where $\mathcal{P}$ is a meta-path, i.e., a sequence of relations defined between different types of vertices. Hence, the $(k, \mathcal{P})$-core is a subgraph where each vertex is connected to at least k other vertices via instances of $\mathcal{P}$. Based on two kinds of sparse matrix products, we propose two kinds of algebraic core decomposition algorithms, which are suitable for general HINs and locally dense HINs, respectively. We have performed extensive empirical evaluations of our algorithms on six large real-world HINs. The results show that the proposed solutions are highly efficient for core decomposition and achieve up to $258.84\times$ speedup than the state-of-the-art parallel algorithm on 20 cores. Moreover, other HIN tasks that involve homogeneous graph construction can also benefit from our algorithms.

***Index Terms*—cohesive subgraphs, dense subgraphs, heterogeneous information networks, matrix computation, parallelization**

I. Introduction

Heterogeneous information networks (HINs) are networks with vertices and edges that are classified into multiple types. Many real systems can often be modeled as HINs to enrich semantics [1], such as bibliographic networks, communication and computer systems, and social media. As shown in Figure 1(a), a toy HIN is constructed for the DBLP network, which consists of vertices of different types (i.e., author, paper, conference, and topic), as well as edges of various types (i.e., write, present, and mention). The directed lines denote their semantic relationships. For example, the authors a_1 and a_2 have *written* a paper p_1 together, which was *presented* in the conference c_1.

Cohesive subgraph search is a fundamental problem in network analysis and has drawn much attention and benefitted many applications among different domains [2, 3, 4]. Among typical cohesive subgraph models (e.g., k-core [5], k-truss [6, 7], and k-clique [8]), k-core is one of the most commonly used ones. A k-core is a graph (or subgraph) where each vertex has at least k neighbors, which plays a critical role in revealing the hierarchical structure of networks, while a larger k value denoting a more cohesive graph (or subgraph). The coreness of a vertex v in a graph G is the largest value of k such that there is a k-core of G containing v. The goal of the k-core decomposition problem is to compute the coreness of each vertex, while the computation of larger coreness naturally depends on the computation of smaller coreness due to the properties of k-core. Extensive studies [9, 10, 11] have been conducted to design efficient k-core decomposition algorithms. However, these studies mainly focus on homogeneous networks, and applying them directly to HINs would be unsuitable since all types of edges would be treated as the same type.



(a) An HIN (b) Schema

Fig. 1: An example HIN with the DBLP network schema.

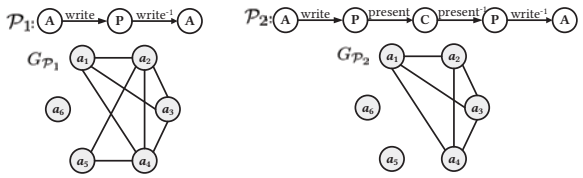
The $(k, \mathcal{P})$-core model. To extend the k-core model to HINs, Fang et al. [12] proposed a core model based on the well-known concept of meta-path [13], namely $(k, \mathcal{P})$-core. A meta-path $\mathcal{P}$ is a sequence of vertex types and edge types between two given vertex types. For a specific edge type R, R^{-1} denotes the inverse edge type of R. Based on a specific meta-path $\mathcal{P}$, the $(k, \mathcal{P})$-core is a set of vertices where each vertex is connected to at least k different vertices via instances of $\mathcal{P}$. For example, in Figure 2(a), a meta-path $\mathcal{P}_1$, defined on authors (A) and papers (P), represents the co-authorship between two authors. In Figure 1(a), the authors $\{a_1, a_2, a_3, a_4\}$ form a cohesive community, in which each pair of authors can be connected by a path instance of $\mathcal{P}_1$. Hence, $\{a_1, a_2, a_3, a_4\}$ is a $(3, \mathcal{P}_1)$-core. By replacing edges with instances of meta-path $\mathcal{P}_1$ on Figure 1(a), we can build a homogeneous graph $G_{\mathcal{P}_1}$, as shown in Figure 2(a), and the 3-core of $G_{\mathcal{P}_1}$ is the same with the $(3,\ \mathcal{P}_1)$-core of Figure 1(a).

The $(k, \mathcal{P})$-core decomposition has a wide range of applications: (1) *Community detection*. Core decomposition can reveal the hierarchical relations of an HIN, which can be used to accelerate the community detection process [14]. Communities in an HIN can be detected by finding $(k, \mathcal{P})$-cores in the network, where k is a variable threshold. The larger the value of k, the more cohesive the community. (2) *Recommendation*. Many E-commerce platforms use HINs to store customer, merchant, and product information. The platform can send advertisement

† Corresponding author.

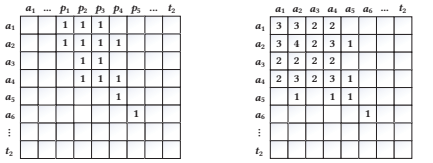
2024 IEEE 40th International Conference on Data Engineering (ICDE) | 979-8-3503-1715-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICDE60146.2024.00189

2375-026X/24/$31.00 ©2024 IEEE
DOI 10.1109/ICDE60146.2024.00189



(a) $\mathcal{P}_1$ and $G_{\mathcal{P}_1}$ (b) $\mathcal{P}_2$ and $G_{\mathcal{P}_2}$

Fig. 2: Illustrating meta-paths and $G_{\mathcal{P}}$s.



(a) M_{write} (b) $M_{write} \times M_{write}^T$

Fig. 3: An example of using matrix product to find co-writers in Figure 1(a).

messages about a specific product to a group of users with similar consumer preferences if one of them has purchased this product before. User groups with similar consumer preferences can be found by $(k, \mathcal{P})$-core decomposition, using meta-paths such as "$customer \xrightarrow{buy} product \xrightarrow{buy^{-1}} customer$" and "$customer \xrightarrow{consume\ in} merchant \xrightarrow{consume\ in^{-1}} customer$". (3) *Ecosystem analysis*. For example, in an HIN of plants and pollinators, core decomposition can point out key species (i.e., species with high coreness in the network) whose stability would prevent cascading extinctions of the ecosystem [15].

Linear algebra for HINs. Although various applications can benefit from the $(k, \mathcal{P})$-core decomposition, existing $(k, \mathcal{P})$-core decomposition algorithms [12] are inefficient on large HINs or long meta-paths due to the tremendous cost of breadth-first search. Hence, the goal of this paper is to design efficient parallel $(k, \mathcal{P})$-core decomposition algorithms. Our solution is based on linear algebra and matrix multiplication, as matrix multiplication is quite parallel-friendly and can enjoy the speedup by boolean matrix multiplication, which will be explained later. Linear algebra plays an important role in graph theory, by which we can investigate adjacency matrices of graphs to help us understand them better [16]. Unlike a homogeneous network which only has one adjacency matrix, an HIN has a series of adjacency matrices, each of which represents a specific type of edge. For example, Figure 3(a) shows the adjacency matrix of the edge type "write" for the DBLP network in Figure 1(a).

Large-scale networks can be processed efficiently by a limited number of matrix multiplications based on their adjacency matrices. When a network is represented by adjacency matrices, matrix-matrix multiplication can be treated as a breadth-first search [17]. For example, co-writers in Figure 1(a) can be searched by matrix multiplication $M_{write} \times M_{write}^T$, as shown in Figure 3(b). We can find that each off-diagonal non-zero entry in $M_{write} \times M_{write}^T$ denotes that the corresponding two vertices are connected in the induced homogeneous graph $G_{\mathcal{P}_1}$ in Figure 2(a).

Challenges and contributions. Based on the above discussions, we adopt the two-stage method to compute the $(k, \mathcal{P})$-core decomposition. We build the induced homogeneous graph $G_{\mathcal{P}}$ first and then compute the coreness of those vertices with target type by using k-core decomposition algorithms on $G_{\mathcal{P}}$. However, $G_{\mathcal{P}}$ is often much denser than the original HIN, as reported in [12], which imposes great challenges for $(k, \mathcal{P})$-core decomposition, especially for the first stage, i.e., the construction of $G_{\mathcal{P}}$. As shown in Table I, the current state-of-the-art $(k, \mathcal{P})$-core decomposition algorithm, HomBCore [12], expends a considerable amount of time in the first stage. In addition, two $\mathcal{P}$-*connected* vertices are always connected by several path instances and result in the redundancy of the search process. To overcome those obstacles that hinder the efficient $(k, \mathcal{P})$-core decomposition, especially the bottleneck of time-consuming construction of the $G_{\mathcal{P}}$, we have made the following contributions:

Datasets	$G_{\mathcal{P}}$ Construction	Coreness Computing
Movielens	99.4%	0.6%
Amazon	96.2%	3.8%
Freebase	96.3%	3.7%
DBLP	93.4%	6.6%
Higgs	97.7%	2.3%
ConceptNet	90.6%	9.4%

TABLE I: The respective average proportions of running time for the first stage and the second stage in HomBCore.

- Considering the sparsity of most real-world HINs, we develop a novel sparse boolean matrix data structure called DP-SpGEM to store the adjacency matrices of HINs. We then develop a sparse boolean matrix multiplication operator $\times_{bool}^{G}$ to accelerate the multiplication process with the DP-SpGEM data structure. An $(k, \mathcal{P})$-core decomposition algorithm named BoolAPCore$^{\mathsf{G}}$ that uses operator $\times_{bool}^{G}$ to compute matrix multiplication is then proposed based on the symmetry of meta-paths and the properties of the transposed matrix. Besides, some other HIN tasks, e.g., $(k, \mathcal{P})$-truss decomposition [18], can also benefit from our fast construction of $G_{\mathcal{P}}$, as they also adopt a two-stage paradigm that will build the induced homogeneous graph at the first stage.
- To reduce the redundant computations for HINs with locally dense regions, e.g., Movielens, we develop a sparse boolean matrix data structure called DP-SpLDM. Based on this data structure, we design a parallel sparse boolean matrix multiplication operator $\times_{bool}^{D}$, which can further accelerate the multiplication process when the HIN has locally dense areas. Based on operator $\times_{bool}^{D}$, an algorithm called BoolAPCore$^{\mathsf{D}}$ is proposed.
- We have experimentally compared our algorithms with the state-of-the-art algorithm on six real-world large HINs. Both theoretical analysis and empirical evaluation validate that our algorithms are up to 258.84× faster than the state-of-the-art algorithm on 20 cores for $(k, \mathcal{P})$-core decomposition.

Outline. The rest of the paper is organized as follows. We review the related work in Section II. In Section III, we

formally present the $(k, \mathcal{P})$-core decomposition problem. We present our boolean algebraic algorithm for general HINs in Section IV, and boolean algebraic algorithm for HINs with locally dense areas in Section V. In Section VI, we show the experimental results. Section VII concludes the paper.

II. Related Work

Cohesive subgraph detection is a fundamental topic in network science that has drawn much attention in academia and industry for decades. To detect the cohesive subgraphs in an effective way, many cohesive subgraph models have been formulated, including k-core [5], k-truss [6, 7], k-clique [8], and k-edge-connected component (k-ECC) [19]. Among these models, k-core, or more generally, k-core decomposition, plays a critical role in revealing the cohesiveness of subgraphs. Existing works on core decomposition can usually be classified into core decomposition over homogeneous graphs (k -core decomposition) and core decomposition over heterogeneous graphs.

***k*-Core Decomposition.** The classical method to perform k-core decomposition in the homogeneous graph is based on a peeling process. Batagelj et al. [20] introduced an algorithm with $O(n+m)$ time cost. In [20], k is iterated from 1 to the maximum degree, and for each k, the vertices with degrees no greater than k are iteratively removed, where k is the coreness of those vertices. Batagelj and Zaveršnik [21] optimized this peeling algorithm with $O(m)$ time complexity based on bin-sort. The peeling algorithms require global information on the entire graph at each step. Thus it is not suitable for parallelization.

In 2016, Linyuan Lü and Stanley [22] introduced a decentralized local method to calculate coreness values in parallel through the connection between H-index, degree, and coreness. Ahmet Erdem Sarıyüce and Pinar [23] generalize Lü et al.'s work for any nucleus decomposition, including k-core and k-truss. However, all the works above focus on homogeneous graphs, and it is unclear how to adapt them for the core decomposition problem over HINs.

Core Decomposition over HINs. Fang et al. [12] were the first to define a notion of $(k, \mathcal{P})$-cores for HINs, generalized from the k-core model on homogeneous graphs. Here, $\mathcal{P}$ denotes a meta-path specified by the users. In [12], they proposed a batch search strategy to build an induced homogeneous graph $G_{\mathcal{P}}$ via a symmetric meta-path $\mathcal{P}$ and extended it for core decomposition, which is named HomBCore. They also proposed a fast algorithm called FastBCore to compute the coreness of a specific query vertex, but it is not specifically designed for core decomposition and performs slower than HomBCore for this purpose. In HomBCore, vertices search for their $\mathcal{P}$-neighbors one by one in a batch fashion with $O(n_1 \cdot d_{1,2} + n_1 \sum_{i=1}^{l} n_i \cdot d_{i,i+1})$ time complexity, in which n_i is the number of vertices whose types match with i-th vertex in meta-path, $d_{i,i+1}$ is the maximum number of vertices with (i+1)-th vertex type that are connected to a vertex with i-th vertex type in meta-path. This approach is well-suited for parallelization at the vertex level, as the batch search process of each vertex can be executed without interference, and without requiring the use of mutex. However, as the graph becomes denser and larger, the efficiency of HomBCore may decrease due to the increased overhead of the search process and the frequently repeated access to the same intermediate vertices. This redundancy can result in numerous unnecessary computations across different vertices, underscoring the need for efficient and innovative solutions that can be parallelized in a scalable manner.

TABLE II: Notations used in this paper.

Notation	Meaning
$\mathcal{H} = (V, E)$	an HIN with vertex set V and edge set E
$\psi(v)$ $(\phi(e))$	the vertex (edge) type of a vertex v (edge e)
$\mathcal{P}$	a symmetric meta-path defined on an HIN schema
l	the length of $\mathcal{P}$
$l(\mathcal{P})$ $(r(\mathcal{P}))$	the left (right) half of a meta-path $\mathcal{P}$
$G_{\mathcal{P}}$	a homogeneous graph induced by a meta-path $\mathcal{P}$ on G
$\mathbf{C}_k$ $(\mathbf{C}_{k,\mathcal{P}})$	a $(k, \mathcal{P})$-core
$d(v)$ $(d(v, S))$	degree number of vertex v, i.e., number of path instances starting from the vertex v and ending at vertex $u \in S$
$c(v)$ $(c(v, S))$	coreness of vertex v
M_R	an adjacency sparse matrix for relation R
M_R^T	the transpose of matrix M_R
$M_R[u, v]$	an element in matrix M_R, the value of which is the number of edges with type R that start from vertex u and end at vertex v

The major bottleneck of HomBCore lies in the significant amount of time required for constructing the homogeneous graph. In 2023, Chatzopoulos et al. [24] proposed a new algorithm for finding $\mathcal{P}$-neighbors based on sparse matrix multiplication and intermediate result caching, namely Atrapos, which can be used to construct $G_{\mathcal{P}}$ in a more efficient way. However, Atrapos opts to directly invoke existing matrix operation libraries for matrix multiplication, limiting its functionality to a serial algorithm. Additionally, generic matrix multiplication may not be perfectly suited for HIN scenarios.

Another core-based model on HINs is the relational community model (r-com) proposed by Jian et al. [25]. The r-com model detects cohesive subgraphs based on a set of relational constraints (e.g., a vertex of type A must have k_1 neighbors of type P). The r-com model aims to find the cohesive subgraph consisting of vertices of different types. However, the relational constraints only restrict the vertices with direct edges on HINs, and it is hard to provide a meaningful topic for the community, which is the focus of the meta-path-based core model [12] adopted in this paper.

III. Problem Definition

A. Preliminaries

We summarize the frequently used notations in Table II.

Definition 3.1 (HIN [13]): An HIN is a directed graph $\mathcal{H} = (V, E)$ with a vertex type mapping function $\psi : V \rightarrow \mathcal{A}$ and an edge type mapping function $\phi : E \rightarrow \mathcal{R}$, where each vertex $v \in V$ belongs to a vertex type $\psi(v) \in \mathcal{A}$, and each edge $e \in E$ belongs to an edge type (also called relation) $\phi(e) \in \mathcal{R}$, and $|\mathcal{A}| + |\mathcal{R}| > 2$.

Definition 3.2 (HIN schema [13]): Given an HIN $\mathcal{H} = (V, E)$ with mappings $\psi : V \rightarrow \mathcal{A}$ and $\phi : E \rightarrow \mathcal{R}$, its schema T_G is a directed graph defined over vertex types $\mathcal{A}$ and edge types (as relations) from $\mathcal{R}$, i.e., $T_G = (\mathcal{A}, \mathcal{R})$

The HIN schema shows all allowable edge types, vertex types, and the relation between each edge type and its vertex types. Figure 1(b) shows the DBLP network schema, in which the vertex types "A", "P", "C", and "T" denote author, paper, conference, and topic, respectively. Note that if there is an edge R from vertex type A to vertex type B, there naturally exists an inverse edge R^{-1} from vertex type B to vertex type A. Here we use lower-case letters (e.g., a_1) to denote vertices and upper-case letters (e.g., A) to denote vertex types.

Definition 3.3 (Meta-path [13]): A meta-path $\mathcal{P}$ is a path defined on an HIN schema $T_G = (\mathcal{A}, \mathcal{R})$, and is denoted in the form $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \cdots \xrightarrow{R_l} A_{l+1}$, where l is the length of $\mathcal{P}$, $A_i \in \mathcal{A}$, and $R_i \in \mathcal{R} (1 \leq i \leq l)$.

We also use vertex type names to denote a meta-path, i.e., $\mathcal{P} = (A_1 A_2 \cdots A_{l+1})$, if there exist no multiple relation types between the same pair of vertex types. The left and right half of a meta-path $\mathcal{P}$ are denoted by $l(\mathcal{P}) = A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \cdots \xrightarrow{R_{\lceil (l-1)/2 \rceil}} A_{\lceil (l+1)/2 \rceil}$ and $r(\mathcal{P}) = A_{l+1} \xrightarrow{R_l^{-1}} A_l \xrightarrow{R_{l-1}^{-1}} \cdots \xrightarrow{R_{\lceil (l+1)/2 \rceil}^{-1}} A_{\lceil (l+1)/2 \rceil}$. The reverse meta-path of a meta-path $\mathcal{P}$ is denoted by $\mathcal{P}^{-1} = A_{l+1} \xrightarrow{R_l^{-1}} A_l \xrightarrow{R_{l-1}^{-1}} \cdots \xrightarrow{R_1^{-1}} A_1$. We say a meta-path $\mathcal{P}$ is symmetric if $l(\mathcal{P})$ is the same as $r(\mathcal{P})$. For example, the meta-path $\mathcal{P}_2$ in Figure 2(b) can be written as $\mathcal{P}_2 = (APCPA)$. The left half, right half, and reverse meta-path of $\mathcal{P}_2$ are $l(\mathcal{P}_2) = A \xrightarrow{write} P \xrightarrow{present} C$, $r(\mathcal{P}_2) = A \xrightarrow{write} P \xrightarrow{present} C$, and $\mathcal{P}_2^{-1} = A \xrightarrow{write} P \xrightarrow{present} C \xrightarrow{present^{-1}} P \xrightarrow{write^{-1}} A$, respectively. Since $l(\mathcal{P}_2) = r(\mathcal{P}_2)$, meta-path $\mathcal{P}_2$ is a symmetric meta-path. While the focus of this paper is on symmetric meta-paths, it is important to note that our algorithms can also be extended to handle asymmetric meta-paths, as we demonstrate in our experimental results presented in Section VI. It is worth noting that unless otherwise stated, all the meta-paths mentioned later in this paper are symmetric.

We call a path $p = a_1 \rightarrow a_2 \cdots \rightarrow a_{l+1}$ a path instance of $\mathcal{P}$ if $\forall i$, satisfy $\psi(a_i) = A_i$ and $\phi(e_i = (a_i, a_{i+1})) = R_i$. Vertex type A_1 is also called the target type. For example, in Figure 1(a), the path $a_1 \rightarrow p_1 \rightarrow c1 \rightarrow p_2 \rightarrow a_2$ is a path instance of $\mathcal{P}_2$, and a_1-a_6 are vertices with the target type.

Definition 3.4 ($\mathcal{P}$-neighbor [18]): Given an HIN $\mathcal{H} = (V, E)$ and a meta-path $\mathcal{P}$, a vertex u is a $\mathcal{P}$-neighbor of a vertex v if there is a path instance of $\mathcal{P}$ between u and v, where $u \neq v$. We also say u and v are $\mathcal{P}$-*connected*, and (u, v) is called a $\mathcal{P}$-pair.

Definition 3.5 ($\frac{\mathcal{P}}{2}$-neighbor [18]): Given an HIN $\mathcal{H} = (V, E)$ and a meta-path $\mathcal{P}$, a vertex u is a $\frac{\mathcal{P}}{2}$-neighbor of a vertex v if there is a path instance of $l(\mathcal{P})$ or $r(\mathcal{P})$ between u and v, and (u, v) is called a $\frac{\mathcal{P}}{2}$-pair.

Example 3.1: Consider the HIN in Figure 1(a) with meta-paths $\mathcal{P}_1$ and $\mathcal{P}_2$ in Figure 2(a) and Figure 2(b). Table III shows the $\mathcal{P}$-neighbors and $\frac{\mathcal{P}}{2}$-neighbors of each vertex with target type A.

TABLE III: Results of $\mathcal{P}$-neighbors and $\frac{\mathcal{P}}{2}$-neighbors on a small DBLP network.

Vertex	$\mathcal{P}_1 = (APA)$		$\mathcal{P}_2 = (APCPA)$	
	$\mathcal{P}$-neighbors	$\frac{\mathcal{P}}{2}$-neighbors	$\mathcal{P}$-neighbors	$\frac{\mathcal{P}}{2}$-neighbors
a_1	$\{a_2, a_3, a_4\}$	$\{p_1, p_2, p_3\}$	$\{a_2, a_3, a_4\}$	$\{c_1\}$
a_2	$\{a_1, a_3, a_4, a_5\}$	$\{p_1, p_2, p_3, p_4\}$	$\{a_1, a_3, a_4\}$	$\{c_1\}$
a_3	$\{a_1, a_2, a_4\}$	$\{p_2, p_3\}$	$\{a_1, a_2, a_4\}$	$\{c_1\}$
a_4	$\{a_1, a_2, a_3, a_5\}$	$\{p_2, p_3, p_4\}$	$\{a_1, a_2, a_3\}$	$\{c_1\}$
a_5	$\{a_2, a_4\}$	$\{p_4\}$	$\emptyset$	$\emptyset$
a_6	$\emptyset$	$\{p_5\}$	$\emptyset$	$\emptyset$

B. Problem Definition

In this paper, we aim to compute the coreness of each vertex with a specific target type over HINs. To connect the vertices with the target type, we adopt a symmetric meta-path $\mathcal{P}$, who is starting and ending with the target type, following the well-known $(k, \mathcal{P})$-core[1] model on HINs [12].

Definition 3.6 ($(k, \mathcal{P})$-core [12]): Given an HIN $\mathcal{H}$, an integer k, and a meta-path $\mathcal{P}$, a $(k, \mathcal{P})$-core of $\mathcal{H}$ is a maximal set $\mathbf{C}_{k,\mathcal{P}}$ of $\mathcal{P}$-connected vertices, s.t. $\forall v \in \mathbf{C}_{k,\mathcal{P}}, d(v, \mathbf{C}_{k,\mathcal{P}}) \geq k$, where vertices of $\mathbf{C}_{k,\mathcal{P}}$ are with the type linked by $\mathcal{P}$.

The $(k, \mathcal{P})$-cores have some interesting properties.

Proposition 1 ([12]): Given an HIN $\mathcal{H}$ and a meta-path $\mathcal{P}$, the $(k+1, \mathcal{P})$-core is contained in the $(k, \mathcal{P})$-core, i.e., $\mathbf{C}_{k+1} \subseteq \mathbf{C}_k$.

Proposition 2 ([12]): Given an HIN $\mathcal{H}$, a meta-path $\mathcal{P}$ and an integer k, for any two $(k, \mathcal{P})$-core $\mathbf{C}_k$ and $\mathbf{C}'_k$, if $\mathbf{C}_k \cap \mathbf{C}'_k \neq \emptyset$, then $\mathbf{C}_k = \mathbf{C}'_k$.

Based on the above properties, the core decomposition over HINs can be defined as follows.

Definition 3.7 ($(k, \mathcal{P})$-core decomposition): Given an HIN $\mathcal{H}$ and a meta-path $\mathcal{P}$, a $(k, \mathcal{P})$-core decomposition is a partition, where vertices with target type are partitioned into layers such that a vertex v is in layer k if it belongs to a $(k, \mathcal{P})$-core but does not belong to a $(k+1, \mathcal{P})$-core. k is called the coreness of v if v is in layer k.

We now formally define the $(k, \mathcal{P})$-core decomposition problem.

Problem 1 ($(k, \mathcal{P})$-core decomposition problem): Given an HIN $\mathcal{H}$ and a meta-path $\mathcal{P}$, return the coreness of each vertex with the target type of $\mathcal{P}$.

Example 3.2: Consider the HIN in Figure 1(a) with meta-path $\mathcal{P}_1$ in Figure 2(a). Vertices with target type A are a_1, a_2, a_3, a_4, a_5, and a_6, the coreness values of them are $c(a_1) = 3$, $c(a_2) = 4$, $c(a_3) = 3$, $c(a_4) = 4$, $c(a_5) = 2$, and $c(a_6) = 0$, respectively.

[1]We use "$(k, \mathcal{P})$-core" to mean "Basic $(k, \mathcal{P})$-core" in this paper.

IV. Boolean Algebraic $(k, \mathcal{P})$-Core Decomposition Algorithms for General HINs

In this section, we present a boolean algebraic algorithm for $(k, \mathcal{P})$-core decomposition that is suitable for general HINs. The main idea of our algebraic algorithms is to construct the induced homogeneous graph $G_{\mathcal{P}}$ via adjacency matrix multiplication and perform the k-core decomposition on $G_{\mathcal{P}}$.

An HIN $\mathcal{H}$ can be represented by a series of adjacency matrices, M_R, $R \in \mathcal{R}$, where the element $M_R[u, v]$ is one when there exists an edge with type R from u to v, and zero when there is no such edge. To reduce space complexity, we only build one matrix M_R for each pair of inverse relations R and R^{-1}, since matrix $M_{R^{-1}}$ is the same as M_R^T. For example, Figure 3(a) shows an adjacency matrix M_{write}, which denotes the edges of type $write$ in Figure 1(a), and the edges of type $write^{-1}$ can be represented by M_{write}^T. Next, we show that the induced homogeneous graph $G_{\mathcal{P}}$ can be computed via adjacency matrix multiplication by the following lemma.

Lemma 4.1 ([26]): Given an HIN $\mathcal{H} = (V, E)$ with three vertex types A, B, and C, suppose that adjacency matrices M_{R_1} and M_{R_2} denote relation $A \xrightarrow{R_1} B$ and $B \xrightarrow{R_2} C$ respectively. For the meta-path $\mathcal{P} = A \xrightarrow{R_1} B \xrightarrow{R_2} C$, the adjacency matrix of the induced homogeneous graph $G_{\mathcal{P}}$ can be obtained by $M_{R_{12}} = M_{R_1} \times M_{R_2}$. $M_{R_{12}}[i, j]$ denotes the number of path instances of $\mathcal{P}$ between vertex i and vertex j.

According to Lemma 4.1, we can build the induced homogeneous graph $G_{\mathcal{P}}$ by matrix chain multiplication, following the order of the relation sequence in meta-path $\mathcal{P}$. To our knowledge, earlier studies like [26, 27, 24] identified $\mathcal{P}$-neighbors using either coordinate (COO) or compressed sparse row (CSR) matrix multiplication. However, multiplying the adjacency matrices in the COO format is both time and space-intensive as it retains each element as a triple. Utilizing the CSR format for multiplication is also time-consuming. Given that HINs tend to be sparse, many empty rows appear in the adjacency matrix, thereby causing unnecessary overhead during row-wise multiplication.

Moreover, when building $G_{\mathcal{P}}$, the non-zero nature of an element is of primary concern, rather than its exact value. Given that matrix multiplication to compute exact values can be notably demanding when vertices with the target type in HINs are linked by numerous path instances of $\mathcal{P}$, opting for sparse boolean matrix multiplication is a more efficient approach for the construction of $G_{\mathcal{P}}$.

A. Sparse Boolean Matrix Multiplication in DP-SpGEM Format

To speed up the adjacency matrix multiplication process and reduce memory usage, we build a new sparse boolean matrix format based on the well-known sparse boolean matrix format [28] and design a parallel sparse boolean matrix multiplication operator to compute the adjacency matrix product for general HINs.

1) Data Structure of Sparse Matrix in DP-SpGEM Format: To adapt the classical CSR format for general HINs (i.e., sparse HINs), we introduce a novel sparse boolean matrix format named Double Pointer Sparse General Matrix (DP-SpGEM). When compared to the CSR format, DP-SpGEM incorporates two primary modifications:

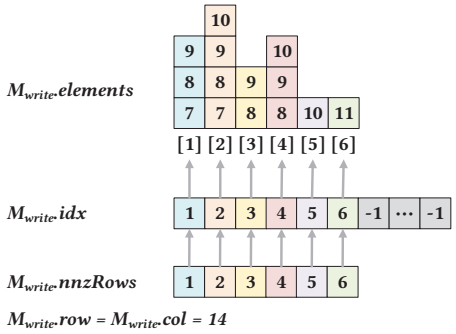


Fig. 4: Illustrating the DP-SpGEM format of M_{write}.

1) DP-SpGEM constructs an additional index, $M.nnzRows[\cdot]$. $M.nnzRows[\cdot]$ retains the original row index of each nonzero row, while $M.idx[\cdot]$ stores the row index of each row in $M.element[\cdot]$ with -1 denoting zero row. Consequently, when the matrix is positioned on the left side of a multiplication, only the nonzero rows are accessed based on the index $M.nnzRows[\cdot]$ and $M.idx[\cdot]$. Meanwhile, when the matrix is on the right side, each row can be accessed directly using the index $M.idx[\cdot]$.
2) Instead of storing it as a monolithic structure, DP-SpGEM accommodates each nonzero row separately within a two-dimensional vector, $M.elements[j][\cdot]$. This design choice ensures parallel computation and insertion for each row.

Table IV shows our data structure of a sparse matrix M in DP-SpGEM format. Each element in $M.elements[j][\cdot]$ with $M.idx[i] = j$ is an integer k, which denotes $M[i, k] = 1$.

TABLE IV: Data structure of a sparse matrix M in DP-SpGEM format.

Member Variable	Meaning
$M.row$ ($M.col$)	the number of rows (columns) in M
$M.nnzRows[\cdot]$	an array storing row indices of all nonzero rows in M
$M.elements[j][\cdot]$	an array storing all nonzero elements of the j-th (candidate) nonzero row in M
$M.idx[i]$	the index in $M.elements$ of the i-th row in M. If the i-th row is a nonzero row, then its nonzero elements are stored in $M.elements[M.idx[i]]$, otherwise $M.idx[i] = -1$

Example 4.1: Figure 4 shows the DP-SpGEM format of M_{write} in Figure 3(a), where $M_{write}.nnzRows$ stores all the nonzero rows of M_{write}, and $M_{write}.idx[i]$ stores the index in $M_{write}.elements$ of the i-th row in M_{write}. $M_{write}.idx[1] = 1$ since the first row of M_{write} is also the first nonzero row of M_{write}, $M_{write}.idx[7] = -1$ since the 7-th row of M_{write} is not a nonzero row.

2) Sparse Boolean Matrix Multiplication Operator $\times_{bool}^{G}$: Based on the DP-SpGEM store of sparse boolean matrices, we design a parallel sparse boolean matrix multiplication operator

$\times_{bool}^{G}$ to compute the matrix product, as shown in Algorithm 1. The main idea of our parallel multiplication operator is to compute different rows in parallel based on Gustavson's classical serial algorithm [28]. Given two sparse boolean matrices M_A and M_B, the operator computes $M_C = M_A \times M_B$. In Gustavson's algorithm, the result matrix has to be calculated row-by-row in serial, regardless of whether the row is empty, since this algorithm does not separately record nonzero rows due to the single pointer structure of the CSR format, and matrix elements are stored in a one-dimensional array $M_C.elements[\cdot]$ in row-major order. To ensure parallelism and boost efficiency, we transform $M_C.elements[\cdot]$ into a two-dimensional vector $M_C.elements[\cdot][\cdot]$ and initialize an array $flag[\cdot]$ to mark nonzero elements for each thread (lines 2-4). After that, each candidate nonzero row (i.e., rows in $M_A.nnzRows[\cdot]$) is computed and nonzero elements are inserted into $M_C.elements[\cdot][\cdot]$ in parallel based on Gustavson's algorithm (lines 5-15). Finally, the array $M_C.nnzRows[\cdot]$ is constructed according to $M_C.elements[\cdot][\cdot]$ and the result matrix M_C is returned (lines 16-19).

Algorithm 1: Sparse Boolean Matrix Multiplication Operator: $\times_{bool}^{G}$

Input: two sparse matrices, M_A and M_B;
Output: the result sparse boolean matrix, $M_C = M_A \times M_B$;

1 $M_C.row \leftarrow M_A.row$, $M_C.col \leftarrow M_B.col$, $M_C.idx[\cdot] \leftarrow -1$;
2 $M_C.elements[\cdot] \leftarrow$ a vector of size($M_A.nnzRows$) empty vectors;
3 **foreach thread** t **do**
4 $flag[\cdot] \leftarrow 0$; `// Initialize per thread`
5 **for** $i \leftarrow 1$ *to size*($M_A.nnzRows[\cdot]$) **in parallel do**
6 $row \leftarrow M_A.nnzRows[i]$, $r \leftarrow M_A.idx[row]$;
7 **foreach** $j \in M_A.elements[r]$ **do**
8 **if** $M_B.idx[j] = -1$ **then continue**;
9 **foreach** $k \in M_B.elements[M_B.idx[j]]$ **do**
10 **if** $flag[k] = 0$ **then**
11 $flag[k] \leftarrow 1$;
12 append k to $M_C.elements[i]$;
13 **if** $M_C.elements[i] \neq \emptyset$ **then**
14 $M_C.idx[row] \leftarrow i$;
15 **foreach** $e \in M_C.elements[i]$ **do** $flag[e] \leftarrow 0$;
16 **foreach** $i \leftarrow 1$ *to size*($M_A.nnzRows[\cdot]$) **do**
17 **if** $M_C.elements[i] \neq \emptyset$ **then**
18 append $M_A.nnzRows[i]$ to $M_C.nnzRows$;
19 **return** M_C;

Figure 5 shows the process of computing the second row of $M_{co-write}$ in Figure 3(b) by sparse boolean matrix multiplication operator $\times_{bool}^{G}$.

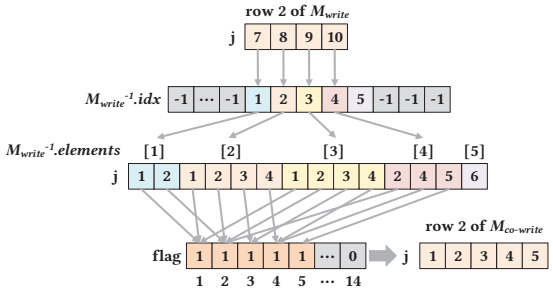


Fig. 5: Illustrating operator $\times_{bool}^{G}$.

B. Boolean Algebraic (k, $\mathcal{P}$)-Core Decomposition Algorithm for General HINs

Based on the parallel sparse boolean matrix multiplication operator $\times_{bool}^{G}$, we present an efficient algorithm in this section, called BoolAPCore$^{\mathsf{G}}$. Given a meta-path $\mathcal{P}$, the algorithm first builds the induced homogeneous graph $G_{\mathcal{P}}$ via sparse matrix multiplication and invokes the homogeneous core decomposition algorithm, AND [23], on $G_{\mathcal{P}}$ to compute coreness of vertices with target type. BoolAPCore$^{\mathsf{G}}$ relies on the following key observation. All the meta-paths are symmetric, which makes $M_{\mathcal{P}} = M_{R_1} \times M_{R_2} \times \cdots \times M_{R_{\frac{l}{2}}} \times M_{R_{\frac{l}{2}}}^T \times \cdots \times M_{R_2}^T \times M_{R_1}^T = M_{R_1} \times M_{R_2} \times \cdots \times M_{R_{\frac{l}{2}}} \times (M_{R_1} \times M_{R_2} \times \cdots \times M_{R_{\frac{l}{2}}})^T$. Hence, we can compute $M_{l(\mathcal{P})}$ first, and then obtain $M_{\mathcal{P}}$ by $M_{l(\mathcal{P})} \times M_{l(\mathcal{P})}^T$, which will significantly reduce the number of multiplications.

However, this solution is not suitable for every meta-path since the overhead of matrix chain multiplications is not always in proportion to the length of the matrix chain. Whether obtaining $M_{\mathcal{P}}$ by $M_{l(\mathcal{P})} \times_{bool}^{G} M_{l(\mathcal{P})}^T$ will reduce the running time or not is mainly related to the densities of matrices we got at each step before we obtain $M_{\mathcal{P}}$ and the density of $M_{l(\mathcal{P})}$. Here, the density of a sparse matrix means the number of nonzero elements over the matrix size. In some cases, the density of $M_{l(\mathcal{P})}$ is much higher than those of other intermediate matrices in the matrix chain, which makes the time cost of computing $M_{l(\mathcal{P})} \times_{bool}^{G} M_{l(\mathcal{P})}^T$ is even higher than the time cost of computing $M_{l(\mathcal{P})} \times_{bool}^{G} M_{R_{\frac{l}{2}}}^T \times_{bool}^{G} \cdots \times_{bool}^{G} M_{R_2}^T \times_{bool}^{G} M_{R_1}^T$. Hence, we will train a multiple linear regression model to decide which multiplication plan to use.

Based on the discussions above, we develop the algorithm BoolAPCore$^{\mathsf{G}}$, shown in Algorithm 2. First, we initialize the average density by 0 and the result matrix by the identity matrix I (lines 1-2). Then, we compute $M_{R_1} \times_{bool}^{G} M_{R_2} \times_{bool}^{G} \cdots \times_{bool}^{G} M_{R_{\frac{l}{2}}}$ step-by-step and record the density of each result matrix (lines 3-5). Next, we calculate the density of $M_{l(\mathcal{P})}$ and the mean density of $(\frac{l}{2} - 1)$ result matrices we got before we obtain $M_{l(\mathcal{P})}$, and then use a function F to evaluate whether to adopt $M_{l(\mathcal{P})} \times_{bool}^{G} M_{l(\mathcal{P})}^T$ to compute $M_{\mathcal{P}}$ or not (lines 6-11). Finally, we construct the induced homogeneous graph $G_{\mathcal{P}}$ and compute the coreness for each vertex in $G_{\mathcal{P}}$ (lines 12-13).

Algorithm 2: BoolAPCore$^{\mathsf{G}}$

Input: $\mathcal{H} = (V, E)$, $\{M_{R_i} | R_i \in \mathcal{R}\}$, $\mathcal{P}$;
Output: $c[\cdot]$: coreness for all vertices with target type;

```
1  ρ_avg ← 0;
2  M ← I ;  // initialize M with identity matrix
3  for i ← 1 to l/2 do
4  |  M ← M ×^G_bool M_{R_i};
5  |  ρ_avg ← ρ_avg + Density(M);
6  ρ_avg ← ρ_avg/(l/2 − 1);
7  ρ_mid ← Density(M);
8  if F(ρ_avg, ρ_mid) > 1 then
9  |  M ← M ×^G_bool M^T;
10 else
11 |  for i ← l/2 + 1 to l do  M ← M ×^G_bool M_{R_i} ;
12 Construct G_P based on M;
13 c[·] ← AND(G_P) ;  // call k-core decomp. algo.
14 return array c[·];
```

Function F is developed to evaluate which method is more suitable for a specific meta-path:

$$F(\rho_{avg}, \rho_{mid}) = c_0 \frac{\rho_{avg}}{\rho_{mid}} + c_1 \rho_{mid} + c_2 \rho_{avg} + c_3 \quad (1)$$

Here, ρ_{avg} and ρ_{mid} are the densities collected in Algorithm 2, c_0, c_2, and c_3 are positive constants, c_1 is a negative constant. The values of c_0, c_1, c_2, and c_3 are trained via multiple regression analysis according to the ratio of running times of the two plans. The result of $F(\rho_{avg}, \rho_{mid})$ is a prediction about $\frac{runtime(\prod_{i=1}^{l} M_{R_i})}{runtime(\prod_{i=1}^{\frac{l}{2}} M_{R_i} \times^G_{bool} (\prod_{i=\frac{l}{2}}^{l} M_{R_i})^T)}$. So when $F(\rho_{avg}, \rho_{mid}) < 1$, we compute $G_{\mathcal{P}}$ in a matrix chain multiplication manner (line 11 in Algorithm 2). Otherwise, we obtain $M_{\mathcal{P}}$ by $M_{l(\mathcal{P})} \times^G_{bool} M^T_{l(\mathcal{P})}$ (line 9 in Algorithm 2).

Lemma 4.2: If BoolAPCore$^{\mathsf{G}}$ use $M_{l(\mathcal{P})} \times^G_{bool} M^T_{l(\mathcal{P})}$ to compute $M_{\mathcal{P}}$, it takes $O((\sum_{i=1}^{\frac{l}{2}-1} nnz(M_i) \cdot nnz(M_{R_{i+1}})/n + nnz(M_{l(\mathcal{P})})^2/n + \frac{l}{2} \cdot np + tm_{\mathcal{P}})/p)$ time, where l is the length of meta-path $\mathcal{P}$, n is the number of vertices in $\mathcal{H}$, $M_i = M_{R_1} \times^G_{bool} M_{R_2} \times^G_{bool} \cdots \times^G_{bool} M_{R_i}$, $nnz(M_i)$ is the number of nonzero elements in M_i, t is the number of iterations when computing coreness on $G_{\mathcal{P}}$, $m_{\mathcal{P}}$ is the number of edges in $G_{\mathcal{P}}$, and p is number of threads.

Proof 4.1: The computation of $M_{l(\mathcal{P})}$ consists of $\frac{l}{2} - 1$ times sparse matrix products, and each product takes $O((nnz(M_i) \cdot nnz(M_{R_{i+1}})/n + np)/p)$ time, where np is the time cost of initializing an array $flag[\cdot]$ for each thread. Thus, building G_p takes $O((\sum_{i=1}^{\frac{l}{2}-1} nnz(M_i) \cdot nnz(M_{R_{i+1}})/n + nnz(M_{l(\mathcal{P})})^2/n + \frac{l}{2} \cdot np)/p)$ time, as F is trained offline with linear time, and the computation of ρ_{avg}, ρ_{mid}, and $F(\rho_{avg}, \rho_{mid})$ can be done in constant time. The time cost of computing coreness estimates for each vertex $v \in G_{\mathcal{P}}$ is $O(size(Neighbor_v))$ per iteration. Therefore, computing coreness can be done in $O(t(\sum_{v \in G_{\mathcal{P}}} size(Neighbor_v))/p) = O(tm_{\mathcal{P}}/p)$. Hence, Lemma 4.2 holds.

V. Boolean Algebraic $(k, \mathcal{P})$-Core Decomposition Algorithms for Locally Dense HINs

In Section IV, we present a boolean algebraic algorithm that is mainly inspired by the sparsity of HINs. However, though real-world HINs are sparse overall, there still exists a number of locally dense regions in some of them, indicating frequently appeared relations.

A. Sparse Boolean Matrix Multiplication in DP-SpLDM Format

To build a sparse matrix format that can support efficient parallel boolean multiplications over sparse matrices with several dense rows/columns, we modify the well-known sparse boolean matrix format [28] by exploiting the locally dense property of the matrices.

1) Data Structure of Sparse Boolean Matrix in DP-SpLDM Format: To adapt the classical CSR format for HINs that are locally dense but globally sparse, we proposed a novel data structure, namely Double Pointer Sparse Locally Dense Matrix (DP-SpLDM). Compared to the CSR format, DP-SpLDM makes two major changes:

1) In DP-SpLDM, a matrix can be stored in both CSR format and CSC format, which reduces unnecessary computations when performing boolean matrix multiplications. Note that it is optional whether to store a matrix in two formats or not. In our algorithm, only those initial adjacency matrices are stored in two formats, as all the intermediate matrices in the matrix multiplication chain are always on the left-hand side and only need to be stored in the CSR format.
2) DP-SpLDM only builds indices for the rows or columns with nonzero elements but not for every row and column, to save space and avoid useless iterations over all-zero rows or columns. To implement, we build an index to record all the nonzero rows and columns, which are named $M.nnzRows[\cdot]$ and $M.nnzCols[\cdot]$, respectively.

TABLE V: Data structure of a sparse boolean matrix M.

Member Variable	Meaning
$M.row$ ($M.col$)	the number of rows (columns) in M
$M.nnzRows[i]$	the row (column) number of the i-th nonzero
($M.nnzCols[i]$)	row (column) in A
$M.rowVal[\cdot]$	an array store the indices of all nonzero
($M.colVal[\cdot]$)	elements of M in CSR (CSC) order
$M.rowIdx[i]$	the index in array $M.rowVal[\cdot]$ of the first element of the i-th nonzero row in M
$M.colIdx[i]$	the index in array $M.colVal[\cdot]$ of the first element of the i-th nonzero column in M

The major difference between DP-SpGEM and DP-SpLDM is that DP-SpLDM does not build an index for those empty rows and can be stored in both CSR format and CSC format simultaneously. Table V shows the data structure of a sparse boolean matrix M in DP-SpLDM format. We abuse the notation M to denote the DP-SpLDM matrix in this section.

Example 5.1: Take the CSR format part in M_{write} as an example. In Figure 6, $M_{write}.nnzRows[\cdot]$ stores the row number of each nonzero row, $M_{write}.rowIdx[\cdot]$ stores the index of the first element of each nonzero row, and $M_{write}.rowVal[\cdot]$ stores the column indices for all nonzero elements in row-major order. The CSC format part in M_{write} is omitted here, as it has a similar structure.

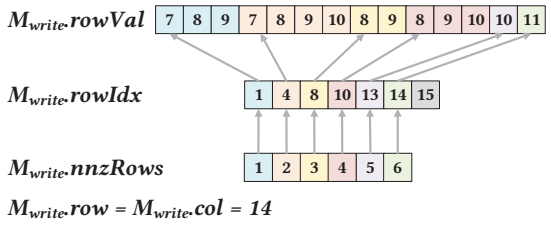


Fig. 6: Illustrating the CSR part of DP-SpLDM M_{write}.

2) Sparse Boolean Matrix Multiplication Operator $\times^D_{bool}$: According to [29], the product of two boolean matrices, $M_C = M_A \times M_B$, is expressed as follows:

$$M_C[i,j] = \bigvee_{k=1}^{m} M_A[i,k] \wedge M_B[k,j], \tag{2}$$

where m is the column number of M_A. From the above equation, we can find that as long as there exists a value of k such that $M_A[i,k] \wedge M_B[k,j]$ is 1 (i.e., True), $M_C[i,j]$ is 1. This observation implies that once we encounter a k value that satisfies the condition, we can cease exploring further k values, preventing redundant computations.

Based on the above discussion, we design an efficient parallel sparse boolean matrix multiplication operator, which is shown in Algorithm 3. We first initialize an array of empty sets to store nonzero elements in each row and initialize an array $flag[\cdot]$ for each thread (lines 2-4). Nonzero elements' computation for each row in M_C can be executed in parallel (lines 5-16). To compute each row in M_C, we set $flag[a.j] = 1$ for every nonzero element a within the row (lines 6-8). Subsequently, the element $M_C[r,c]$ for every nonzero row r in M_A and column c in M_B is determined based on $flag[\cdot]$ as per Equation (2); post-processing the entire row, nonzero elements in $flag[\cdot]$ revert to zero (lines 9-16). Notably, if $M_A[r,i] \wedge M_B[i,c] = 1$ holds true, the nonzero element c is added to $elements[r]$ and the for loop terminates (lines 12-14). Afterward, we use a for loop (lines 17-19) to merge those nonzero elements row-by-row and compute indices for each nonzero row. Finally, the result matrix is returned (line 20).

Example 5.2: Consider the sparse boolean matrix multiplication $M_{co-write} = M_{write} \times^D_{bool} M_{write^{-1}}$. The process of computing the second row of $M_{co-write}$ is shown in Figure 7. Each element $\in flag[1:14]$ is initialized as 0 at first. Then, each element $\in flag[7:10]$ is set as 1 since there are four nonzero elements in the second row: $(2,7)$, $(2,8)$, $(2,9)$, and $(2,10)$. After that, we compute $M_{co-write}[2,1:6]$ since columns 1-6 of $M_{write^{-1}}$ have nonzero elements. As shown in Figure 7, the computation of $M_{co-write}[2,1:5]$ has been stopped at the very beginning. This is because

Algorithm 3: Boolean Matrix Multiplication: $\times^D_{bool}$

Input: two sparse boolean matrices, M_A and M_B;
Output: the result sparse boolean matrix, $M_C = M_A \times M_B$;

1 $M_C.row \leftarrow M_A.row,\ M_C.col \leftarrow M_B.col$;
2 $elements[\cdot] \leftarrow \emptyset$; // $elements[\cdot]$ is an array of sets
3 **foreach thread** t **do**
4 $flag[\cdot] \leftarrow 0$; // Initialize per thread
5 **foreach** *row* $r \in M_A.nnzRows[\cdot]$ **in parallel do**
6 $l_r \leftarrow M_A.rowIdx[r],\ u_r \leftarrow M_A.rowIdx[r+1]-1$;
7 **foreach** $j \in M_A.rowVal[l_r : u_r]$ **do**
8 $flag[j] \leftarrow 1$;
9 **foreach** *column* $c \in B.nnzCols[\cdot]$ **do**
10 $l_c \leftarrow M_B.colIdx[c], u_c \leftarrow M_B.colIdx[c+1]-1$;
11 **foreach** $i \in M_B.colVal[l_c : u_c]$ **do**
12 **if** $flag[i] = 1$ **then**
13 append c to $elements[r]$;
14 **break**;
15 **foreach** $j \in M_A.rowVal[l_r : u_r]$ **do**
16 $flag[j] \leftarrow 0$;
17 **foreach** $elements[i] \neq \emptyset$ **do**
18 $M_C.rowVal[\cdot] \leftarrow M_C.rowVal[\cdot] \cup elements[i]$;
19 update $M_C.nnzRows$ and $M_C.rowIdx$;
20 **return** M_C;

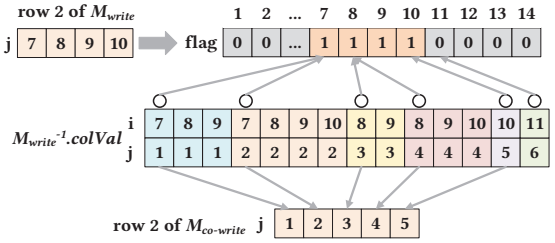


Fig. 7: Illustrating operator $\times^D_{bool}$.

for each column $c \in M_{write^{-1}}.nnzCols[\cdot]$ with the first nonzero element (i,j), we have $flag[i] = 1$. Finally, elements $(2,1)$, $(2,2)$, $(2,3)$, $(2,4)$, and $(2,5)$ are added to $elements[\cdot]$, which forms the second row of $M_{co-write}$.

Reviewing the sparse boolean matrix multiplication in Algorithm 3, we can find the result of $M \times M^T$ is a symmetric matrix. Based on this finding, we can derive the following lemma.

Lemma 5.1: Given an HIN G and a symmetric meta-path $\mathcal{P}$, the adjacency matrix M of the induced homogeneous graph $G_{\mathcal{P}}$ will be a symmetric matrix.

Proof 5.1: The lemma follows from $M = M_{l(\mathcal{P})} \times M^T_{l(\mathcal{P})}$.

Motivated by Lemma 5.1, we propose a novel transpose boolean matrix multiplication operator $\times_{trans}$ to compute $M_{l(\mathcal{P})} \times M^T_{l(\mathcal{P})}$ by only computing the lower triangular part of

the result matrix, shown in Algorithm 4. It consists of three steps: (1) compute the lower triangular part of $M_{l(\mathcal{P})} \times M_{l(\mathcal{P})}^T$ in parallel as Algorithm 3 do (lines 1-2); (2) construct the upper triangular part by transposing the lower triangular part row-by-row (lines 3-5); and (3) merge the nonzero elements and build the row index for the result matrix, then return it (lines 6-7). Note that though M_B in Algorithm 4 is the transpose of M_A, it does not need to be attained by transposing. This is because M_B needs to be stored in column-major order in this algorithm, which is the same as M_A in row-major order. Hence, we can attain the i-th nonzero column of M_B directly by visiting the i-th nonzero row of M_A.

Algorithm 4: Transpose Boolean Matrix Multiplication: $\times_{trans}^{D}$

Input: a sparse boolean matrix, M_A;
Output: the result symmetric sparse boolean matrix $M_C = M_A \times M_A^T$;

1 $M_B \leftarrow M_A^T$;
2 reuse lines 1-14 from Algorithm 3 by restricting $c \leq r$ to only compute the lower triangular part of M_c;
3 **foreach** $elements[r] \neq \emptyset$ **do**
4 **foreach** $c \in elements[r]$, *where* $c \leq r$ **do**
5 append r to $elements[c]$;
6 reuse lines 15-17 in Algorithm 3 to build M_C;
7 **return** M_C;

B. Boolean Algebraic (k, $\mathcal{P}$)-Core Decomposition Algorithm

Based on the parallel sparse boolean matrix multiplication operators, $\times_{bool}^{D}$ and $\times_{trans}^{D}$, we propose a (*k*, $\mathcal{P}$)-core decomposition algorithm, denoted by BoolAPCore$^{\mathsf{D}}$.

Algorithm 5: BoolAPCore$^{\mathsf{D}}$

Input: $\mathcal{H} = (V, E)$, $\{M_{R_i} | R_i \in \mathcal{R}\}$, $\mathcal{P}$;
Output: $c[\cdot]$: coreness for all vertices with target type;

1 $M \leftarrow I$; `// initialize M with identity matrix`
2 **for** $i \leftarrow 2$ *to* $\frac{l}{2}$ **do** `// l is the length of` $\mathcal{P}$
3 $M \leftarrow M \times_{bool}^{D} M_{R_i}$;
4 $M \leftarrow M \times_{trans}^{D} M^T$;
5 construct $G_{\mathcal{P}}$ based on M;
6 $c[\cdot] \leftarrow$ `AND(`$G_{\mathcal{P}}$`)` ; `// call k-core decomp. algo.`
7 **return** *array* $c[\cdot]$;

Algorithm 5 gives the pseudo-code of BoolAPCore. First, it initializes M by the identity matrix (line 1). Second, it computes $M_{l(\mathcal{P})}$ using the sparse boolean matrix multiplication operator $\times_{bool}^{D}$ (Algorithm 3) (line 2-3). Next, it obtains $M_{\mathcal{P}}$ by $M_{l(\mathcal{P})} \times M_{l(\mathcal{P})}^T$ using the transpose multiplication operator $\times_{trans}^{D}$ (Algorithm 4) (line 4). Finally, the algorithm constructs the induced homogeneous graph $G_{\mathcal{P}}$, computes the coreness for each vertex in $G_{\mathcal{P}}$ by AND, and returns the result (lines 5-7).

We have tried to build a similar multiple regression model $F'(\rho_{avg}, \rho_{mid})$ for BoolAPCore$^{\mathsf{D}}$ as we do for BoolAPCore$^{\mathsf{G}}$. However, the model shows that $F'(\rho_{avg}, \rho_{mid}) > 1$ is true for any possible variable with respect to the training data, which implies that using the transpose multiplication operator (Algorithm 4) to get $G_{\mathcal{P}}$ is an optimal solution in most cases.

Lemma 5.2: BoolAPCore$^{\mathsf{D}}$ takes $O(((\sum_{i=1}^{\frac{l}{2}-1} R_{M_i} \cdot (\delta \cdot nnz(M_{R_{i+1}}))) + R_{M_{l(\mathcal{P})}} \cdot (\frac{\delta}{2} nnz(M_{l(\mathcal{P})})) + nnz(M_{l(\mathcal{P})}) + tm_{\mathcal{P}})/p)$ time, where $R_M(C_M)$ denotes the number of nonzero rows (columns) of the matrix M, δ denotes the overall percentage of vertices that have been visited before the early stop during the matrix chain multiplication, n, $nnz(M)$, l, t, $m_{\mathcal{P}}$, and p follow the notation of Lemma 4.2.

Proof 5.2: The construction of $M_{l(\mathcal{P})}$ consists of $\frac{l}{2} - 1$ times sparse boolean matrix products, and each product takes $O(\sum_{r=1}^{R_{M_i}} (nnz(M_i.nnzRows_r) + \delta \sum_{c=1}^{C_{M_{R_{i+1}}}} nnz(M_{R_{i+1}}.nnzCols_c))/p) = O(\sum_{r=1}^{R_{M_i}} (nnz(M_i.nnzRows_r) + \delta \cdot nnz(M_{R_{i+1}}))/p) = O(R_{M_i} \cdot (\delta \cdot nnz(M_{R_{i+1}})) + nnz(M_i)) = O(R_{M_i} \cdot (\delta \cdot nnz(M_{R_{i+1}})))$ time, where $M_i.nnzRows_r(M_i.nnzCols_r)$ denotes the r-th nonzero row (column) of matrix M_i. The time cost of computing $M_{l(\mathcal{P})} \times_{trans}^{D} M_{l(\mathcal{P})}^T$ is $O(\sum_{r=1}^{R_{M_{l(\mathcal{P})}}} (nnz(M_{l(\mathcal{P})}.nnzRows_r) + \frac{\delta}{2} nnz(M_{l(\mathcal{P})}))/p) = O((R_{M_{l(\mathcal{P})}} \cdot (\frac{\delta}{2} nnz(M_{l(\mathcal{P})})) + nnz(M_{l(\mathcal{P})}))/p)$. Hence, Lemma 5.2 holds.

In an HIN with locally dense regions, both R_{M_i} and δ are relatively small, leading to a significant decrease in time complexity. Theoretically, as $\delta_{M_{R_{i+1}}} \propto \frac{1}{nd_{M_{R_{i+1}}}}$, $nnz(M_i) = n^2 d_{M_i}$, when $avg(nnz(M_i) \cdot nnz(M_{R_{i+1}})/n) > avg(R_{M_i} \cdot (\delta_{M_{R_{i+1}}} \cdot nnz(M_{R_{i+1}})))$, i.e., $avg(R_{M_i}) < avg(n^2 d_{M_i} \cdot d_{M_{R_{i+1}}})$, where d_{M_i} is the density of matrix M_i, BoolAPCore$^{\mathsf{D}}$ should be more efficient than BoolAPCore$^{\mathsf{G}}$. When comparing our BoolAPCore$^{\mathsf{D}}$ with HomBCore [12] in terms of time complexity, we observe that HomBCore needs to perform a BFS search for each target node of $\mathcal{P}$ to find the $\mathcal{P}$-neighbors of the current node. This process can result in the repeated traversal of the same intermediate nodes. In contrast, our BoolAPCore$^{\mathsf{D}}$ algorithm can reduce redundant computations among different target nodes through (transposed) matrix multiplications and early stops, leading to significantly improved time efficiency.

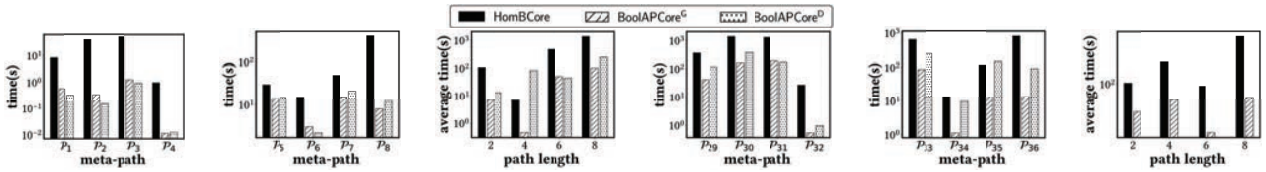
VI. Experiments

We now present the experimental results. We first introduce the experimental setup in Section VI-A, then discuss the results of efficiency evaluation in Section VI-B, and evaluate the effectiveness in Section VI-C.

A. Setup

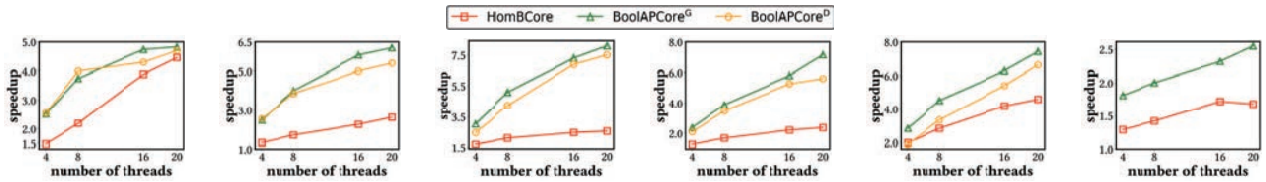
We use six real-world datasets, encompassing domains such as e-commerce, academia, social networks, and knowledge bases: Movielens[2] [30], Amazon[3] [31], Freebase [32, 33], DBLP[4], Higgs [34], and ConceptNet [35]. Freebase and

[2]https://grouplens.org/datasets/movielens/100k/
[3]http://jmcauley.ucsd.edu/data/amazon/
[4]https://dblp.org/xml/



(a) Movielens (b) Amazon (c) Freebase (d) DBLP (e) Higgs (f) Conceptnet

Fig. 8: Single-threaded running times for HomBCore, BoolAPCore^{G} and BoolAPCore^{D} on 6 datasets.



(a) Movielens (b) Amazon (c) Freebase (d) DBLP (e) Higgs (f) Conceptnet

Fig. 9: Average parallel speedup of HomBCore, BoolAPCore^{G} and BoolAPCore^{D}, w.r.t. single-threaded running times.

ConceptNet are knowledge bases with rich schemas, while the other four datasets have relatively simple schemas. Their statistics, such as the number of vertices, the number of edges, the maximum number of edges among the induced homogeneous graphs, the number of edge types, and their domains, are reported in Table VI.

TABLE VI: Datasets used in our experiments.

Dataset	Vertices	Edges	Maximal Edges in $G_{\mathcal{P}}$s	Edge types	Domain
Movielens	2,672	104,777	2,642,090	4	movie
Amazon	13,136	209,746	37,979,370	4	e-commerce
Freebase	75,043	316,232	688,378,491	13	knowledge base
DBLP	135,527	459,444	576,675,971	4	academia
Higgs	456,626	15,367,315	187,911,587	4	social network
ConceptNet	20,967,586	34,030,244	459,754,404	34	knowledge base

We compare the following algorithms in our experiments:

1) HomBCore [12] is the state-of-the-art $(k, \mathcal{P})$-core decomposition algorithm.
2) Atrapos [24] is the state-of-the-art $\mathcal{P}$-neighbors searching algorithm based on sparse matrix multiplication. We mainly compare it with our algorithms in the context of $G_{\mathcal{P}}$ construction.
3) BoolAPCore^{G} is our proposed boolean algebraic $(k, \mathcal{P})$-core decomposition algorithm for general HINs (Section IV-B).
4) BoolAPCore^{D} is our proposed boolean algebraic $(k, \mathcal{P})$-core decomposition algorithm for locally dense HINs (Section V-B).

We evaluate the algorithms w.r.t. in total 58 meta-paths for all datasets. Specifically, for each of Movielens, Amazon, DBLP, and Higgs, we collect 4 frequently-used meta-paths with path lengths varying from 2 to 6, and 20 meta-paths with path lengths varying from 2 to 8 for Freebase and ConceptNet, which contain more vertex and edge types, denoted as $\mathcal{P}_1$-$\mathcal{P}_{56}$. Besides, $\mathcal{P}_{57}$ and $\mathcal{P}_{58}$ denote two asymmetric meta-paths from Movielens and Freebase, respectively. All the algorithms are implemented in C++ with STL used. The source codes for our algorithms are publicly available[5]. We run experiments on a machine having an Intel(R) Xeon(R) Gold 6338 CPU @ 2.00GHz processor and 512GB of memory, with Ubuntu installed.

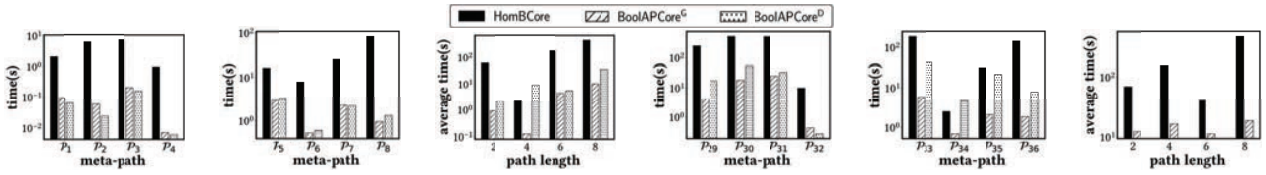
B. Efficiency Evaluation

In this section, we mainly compare our algorithms with the state-of-the-art $(k, \mathcal{P})$-core decomposition algorithm HomBCore and the state-of-the-art $\mathcal{P}$-neighbors searching algorithm Atrapos.

1) Serial Performance: In this experiment, we evaluate the serial efficiency of our algorithms. The single-threaded running times of HomBCore, BoolAPCore^{G} and BoolAPCore^{D} are reported in Figure 8. Note that we do not run BoolAPCore^{D} on ConceptNet since this graph is too sparse and not suitable for it. From Figures 8(a)-8(f), we make the following observations:

- BoolAPCore^{G} outperforms HomBCore in all cases, while BoolAPCore^{D} outperform HomBCore in most meta-paths, except meta-paths with path length 4 in Freebase dataset. The degeneration of BoolAPCore^{D} in this case is caused by the sparsity of the corresponding $G_{\mathcal{P}}s$ (less than $0.1‰$ edges exist among all possible edges in average). Such sparsity results in an inflated δ value, representing many vertices that cannot be pruned by early stop.
- Compared with HomBCore, BoolAPCore^{G} achieves 14.69-114.15×, 2.15-49.84×, 9.66-15.07×, 6.64-49.63×, 7.45-60.97×, and 3.20-11.36× speedups on Movielens, Amazon, Freebase, DBLP, Higgs, and ConceptNet datasets, respectively; while BoolAPCore^{D} achieves 25.38-258.44×, 1.97-32.64×, 0.08-10.85×, 3.21-28.48×, and 0.81-8.84× speedups on the former five datasets, respectively.
- BoolAPCore^{D} outperforms BoolAPCore^{G} and HomBCore significantly on Movielens dataset, this is because Movielens is the densest dataset among all six datasets, which will fully

[5]https://github.com/Yucan-G/parallel-k-P-core-decomposition-code



(a) Movielens (b) Amazon (c) Freebase (d) DBLP (e) Higgs (f) Conceptnet

Fig. 10: Parallel running times for HomBCore, BoolAPCore$^{\mathsf{G}}$ and BoolAPCore$^{\mathsf{D}}$ on 6 datasets with 20 threads.

utilize the early stop mechanism of BoolAPCore$^{\mathsf{D}}$.

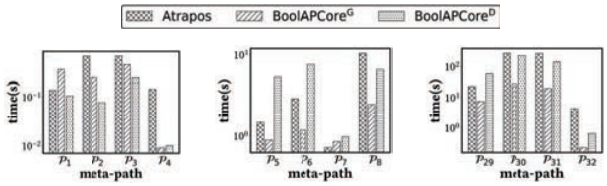
In summary, the choice of algorithm depends on the sparsity of the HIN and the presence of locally dense areas. For HINs with locally dense areas, BoolAPCore$^{\mathsf{D}}$ is the best choice. For sparse HINs without locally dense areas, BoolAPCore$^{\mathsf{G}}$ is the better choice. For most HINs, BoolAPCore$^{\mathsf{G}}$ is a suitable choice, except for those that are extremely dense.

2) Parallel Performance: Figure 9 presents the speedups of HomBCore, BoolAPCore$^{\mathsf{G}}$ and BoolAPCore$^{\mathsf{D}}$ by multi-thread parallel compared to serial computation, respectively. Here, BoolAPCore$^{\mathsf{D}}$ use matrix multiplication operator $\times_{bool}^{D}$ to compute M (line 4 in Algorithm 5), since matrix multiplication operator $\times_{trans}^{D}$ is not suitable for parallelization due to the unbalance of workloads between different threads. The thread numbers tested are 4, 8, 16, and 20. The results reveal that both BoolAPCore$^{\mathsf{G}}$ and BoolAPCore$^{\mathsf{D}}$ achieve more pronounced overall speedups in comparison to HomBCore. As shown in Figures 9(a)-9(e), BoolAPCore$^{\mathsf{G}}$ and BoolAPCore$^{\mathsf{D}}$ obtain significant speedups in all cases with 20 threads and already achieve a relatively high speedup with eight threads. This is reasonable since BoolAPCore$^{\mathsf{G}}$ and BoolAPCore$^{\mathsf{D}}$ only process those rows/columns with nonzero elements, which makes the workloads of each thread more balanced than those in HomBCore. We can also observe that BoolAPCore$^{\mathsf{G}}$ and BoolAPCore$^{\mathsf{D}}$ scale well with the number of threads. However, the speedups of BoolAPCore$^{\mathsf{G}}$ in ConceptNet are relatively low, primarily due to the sparsity of the dataset, which leads to the suboptimal utilization of the array $flag[\cdot]$.

The parallel running times for HomBCore, BoolAPCore$^{\mathsf{G}}$ and BoolAPCore$^{\mathsf{D}}$ with 20 threads are shown in Figure 10. We see that BoolAPCore$^{\mathsf{G}}$ outperforms HomBCore for all datasets, while BoolAPCore$^{\mathsf{D}}$ outperforms HomBCore in 18 out of 20 cases. Compared with HomBCore, BoolAPCore$^{\mathsf{G}}$ and BoolAPCore$^{\mathsf{D}}$ get a maximal of 126.65× and 258.84× speedups, respectively.

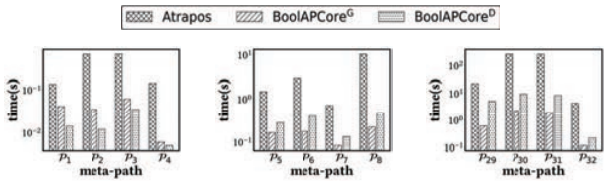
Hence, we conclude that BoolAPCore$^{\mathsf{G}}$ and BoolAPCore$^{\mathsf{D}}$ can be well parallelized, where BoolAPCore$^{\mathsf{G}}$ is a stable algorithm that is suitable for almost all kinds of HINs, BoolAPCore$^{\mathsf{D}}$ is more suitable for those dense HINs, which follows the discussion in Section VI-B1.

3) $G_{\mathcal{P}}$ Construction Performance: In this part, we examine the $G_{\mathcal{P}}$ construction performance of our algorithms against Atrapos. The experiment is conducted on Movielens, Amazon, and DBLP datasets since the implementation of Atrapos only supports HINs where only one type of edge exists between two distinct vertex types. This requirement excludes datasets like Freebase, Higgs, and ConceptNet. Figure 11 shows serial running times for $G_{\mathcal{P}}$ construction of Atrapos, BoolAPCore$^{\mathsf{G}}$ and BoolAPCore$^{\mathsf{D}}$. Our algorithms outperform Atropos in 11 out of 12 meta-paths with up to 19.49× speedup. Figure 12 presents the parallel $G_{\mathcal{P}}$ construction times of our algorithms against the serial running times of Atrapos, as Atrapos is inherently a serial algorithm. BoolAPCore$^{\mathsf{D}}$ outperforms Atrapos at a maximal of 53.58× speedup on Movielens, while BoolAPCore$^{\mathsf{G}}$ outperforms Atrapos at a maximal of 46.01× and 131.89× speedups on Amazon and DBLP, respectively.



(a) Movielens (b) Amazon (c) DBLP

Fig. 11: Single-threaded $G_{\mathcal{P}}$ construction times of algorithms.

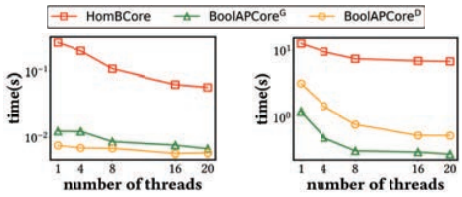


(a) Movielens (20) (b) Amazon (64) (c) DBLP (64)

Fig. 12: Parallel $G_{\mathcal{P}}$ construction times of algorithms (numbers in parentheses represent the number of threads).

4) Performance on Asymmetric Meta-paths: We examine the performance of BoolAPCore$^{\mathsf{G}}$ and BoolAPCore$^{\mathsf{D}}$ with asymmetric meta-paths in this part. Although our algorithms are designed for symmetric meta-paths, they can also be applied to asymmetric meta-paths by using $M_{R_1} \times M_{R_2} \times \cdots \times M_{R_l}$ to obtain $M_{\mathcal{P}}$. We demonstrate the running time of each algorithm with two asymmetric meta-paths in Figure 13. Our results clearly show that our methods outperform HomBCore significantly in both cases, demonstrating the time efficiency of our algorithms for asymmetric meta-paths.

5) Memory Usage: To evaluate the memory usage of our algorithms, we present the single-threaded memory usage of HomBCore, BoolAPCore$^{\mathsf{D}}$, and BoolAPCore$^{\mathsf{G}}$ in Table VII. Since meta-paths on Freebase and ConceptNet datasets were tested in bulk by a program, only the maximum memory usage



(a) Movielens with $\mathcal{P}_{57}$ (b) Freebase with $\mathcal{P}_{58}$

Fig. 13: Performance of algorithms on asymmetric meta-paths.

can be reported. Our results indicate that the average and the maximal memory usage of BoolAPCore$^{\mathsf{D}}$ and BoolAPCore$^{\mathsf{G}}$ are consistently lower than HomBCore in most datasets, confirming that our algorithms are memory-efficient since we use sparse matrices instead of dense matrices.

Furthermore, we examine the memory usage of the initial adjacency matrices in BoolAPCore$^{\mathsf{D}}$, given they are stored in two formats. Despite this redundancy, our results show that their memory overhead is relatively low, which further demonstrates that $G_{\mathcal{P}}$ is much denser than traditional graphs.

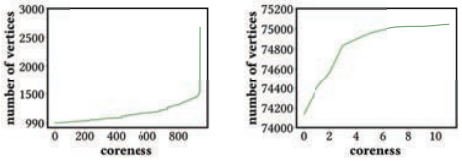
TABLE VII: Memory usage of our algorithms (MB).

Dataset	HomBCore		BoolAPCore$^{\mathsf{D}}$			BoolAPCore$^{\mathsf{G}}$	
	max	mean	max	mean	matrices	max	mean
Movielens	200	105	161	77	0.85	156	58.89
Amazon	2,439	1,633	2,115	1,318	1.75	2126	1,435
Freebase	18,975	-	19,542	-	4.79	24,729	-
DBLP	80,831	28,248	70,346	24,349	6.40	30,302	15,540
Higgs	9,792	4,016	6,258	2,666	126.99	6,339	2,816
ConceptNet	11,027	-	-	-	-	36,025	-

C. Effectiveness Evaluation

1) Core Analysis: In this experiment, we examine the size distribution of $(k, \mathcal{P})$-core, where k ranges from 0 to the maximum coreness. Due to the space limitation, we only show results of two meta-paths, i.e., $\mathcal{P}_1$ and $\mathcal{P}_{12}$ on Movielens and Freebase, respectively, as BoolAPCore$^{\mathsf{D}}$ perform well with $\mathcal{P}_1$ while the performance of BoolAPCore$^{\mathsf{G}}$ is close to HomBCore with $\mathcal{P}_{12}$. We show the distribution of vertices with coreness$\leq k$ in Figure 14. The size distribution of $\mathcal{P}_1$ on Movielens is different from that of $\mathcal{P}_{12}$ on Freebase. Coreness values of vertices in Figure 14(a) range from 0 to 936, and most vertices have a relatively high coreness, which means that there exist some cohesive subgraphs in $G_{\mathcal{P}_1}$. However, the maximum coreness value in Figure 14(b) is 11 with only 12 vertices, and most vertices have a coreness less than 4, which makes $G_{\mathcal{P}_{12}}$ extremely sparse. Thus, the breadth-first search process to build $G_{\mathcal{P}_{12}}$ is more like a chain structure than a tree structure and there exist no local dense subgraphs in the graph, which will cause our BoolAPCore$^{\mathsf{G}}$ algorithm to degenerate.

2) Effectiveness of $F(\rho_{avg}, \rho_{mid})$: In this part, we examine the effectiveness of function $F(\rho_{avg}, \rho_{mid})$ in BoolAPCore$^{\mathsf{G}}$ by a case study. Take a deep look into the meta-path $\mathcal{P}_6 = (UIVIU)$, where "U", "I", "V" denote user, item, and view, respectively. In this case, $F(\rho_{avg}, \rho_{mid}) = 0.72 < 1$, so $M_{\mathcal{P}_6}$ is computed by $M_{l(\mathcal{P}_6)} \times^G_{bool} M^T_{R_2} \times^G_{bool} M^T_{R_1}$. In Figure 15, we



(a) $\mathcal{P}_1$ (b) $\mathcal{P}_{12}$

Fig. 14: Number of vertices with coreness $\leq k$.

show the running times of two methods with different numbers of threads, in which w/o Transpose denotes obtaining $M_{\mathcal{P}_6}$ by $M_{l(\mathcal{P}_6)} \times^G_{bool} M^T_{R_2} \times^G_{bool} M^T_{R_1}$, and Transpose denotes obtaining $M_{\mathcal{P}_6}$ by $M_{l(\mathcal{P}_6)} \times^G_{bool} M^T_{l(\mathcal{P}_6)}$. We can find that in this case, w/o Transpose outperforms Transpose with respect to different numbers of threads, which proves the effectiveness of function $F(\rho_{avg}, \rho_{mid})$ in BoolAPCore$^{\mathsf{G}}$.

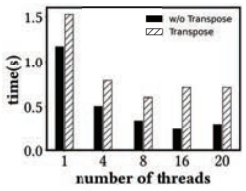


Fig. 15: Running times of 2 methods on $Amazon$ with $\mathcal{P}_6$.

VII. Conclusion

In this paper, we study the core decomposition problem over heterogeneous information networks (HINs). We adopt the well-known $(k, \mathcal{P})$-core model [12] to model cohesive subgraphs. For general HINs and HINs with locally dense properties, we propose efficient boolean algebraic algorithms that can be parallelized in a scalable manner and provide up to two orders of magnitude speedup compared to the existing algorithm on 20 cores. Experimental results on real networks demonstrate that our algorithms are effective and efficient for detecting cohesive subgraphs on HINs. We believe that our algebraic algorithms for building the induced homogeneous graph will be beneficial for many HIN-related problems and make it possible to directly apply those homogeneous graph algorithms over HINs at a small cost.

In future research, we will focus on how to develop parallel algorithms for other cohesiveness models over HINs, e.g., $(k, \mathcal{P})$-truss [18]. It would also be interesting to explore some new cohesive subgraph models for HINs, such as core models with multiple meta-paths.

Acknowledgment

This work was supported in part by NSFC under Grant 62302421 and 62102341, Basic and Applied Basic Research Fund in Guangdong Province under Grant 2023A1515011280, Guangdong Talent Program under Grant 2021QN02X826, and Shenzhen Science and Technology Program under Grants JCYJ20220530143602006 and ZDSYS20211021111415025. This paper was also supported by Shenzhen Science and Technology Program and Guangdong Key Lab of Mathematical Foundations for Artificial Intelligence.

REFERENCES

[1] C. Shi, Y. Li, J. Zhang, Y. Sun, and P. S. Yu, "A survey of heterogeneous information network analysis," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 1, pp. 17–37, 2017.

[2] C. Shi, R. Wang, Y. Li, P. S. Yu, and B. Wu, "Ranking-based clustering on general heterogeneous information networks by network projection," in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014*, J. Li, X. S. Wang, M. N. Garofalakis, I. Soboroff, T. Suel, and M. Wang, Eds. ACM, 2014, pp. 699–708. [Online]. Available: https://doi.org/10.1145/2661829.2662040

[3] Y. Sun, B. Norick, J. Han, X. Yan, P. S. Yu, and X. Yu, "Integrating meta-path selection with user-guided object clustering in heterogeneous information networks," in *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, Beijing, China, August 12-16, 2012*, Q. Yang, D. Agarwal, and J. Pei, Eds. ACM, 2012, pp. 1348–1356. [Online]. Available: https://doi.org/10.1145/2339530.2339738

[4] Y. Zhou and L. Liu, "Social influence based clustering of heterogeneous information networks," in *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, I. S. Dhillon, Y. Koren, R. Ghani, T. E. Senator, P. Bradley, R. Parekh, J. He, R. L. Grossman, and R. Uthurusamy, Eds. ACM, 2013, pp. 338–346. [Online]. Available: https://doi.org/10.1145/2487575.2487640

[5] S. B. Seidman, "Network structure and minimum degree," *Social Networks*, vol. 5, no. 3, pp. 269–287, 1983.

[6] J. Cohen, "Trusses: cohesive subgraphs for social network analysis," National Security Agency, Tech. Rep., 2008.

[7] J. D. Cohen, "Graph twiddling in a mapreduce world," *Comput. Sci. Eng.*, vol. 11, no. 4, pp. 29–41, 2009. [Online]. Available: https://doi.org/10.1109/MCSE.2009.120

[8] J. W. Moon and L. Moser, "On cliques in graphs," *Israel Journal of Mathematics*, vol. 3, no. 1, pp. 23–28, 1965. [Online]. Available: https://doi.org/10.1007/BF02760024

[9] H. Esfandiari, S. Lattanzi, and V. S. Mirrokni, "Parallel and streaming algorithms for k-core decomposition," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, ser. Proceedings of Machine Learning Research, J. G. Dy and A. Krause, Eds., vol. 80. PMLR, 2018, pp. 1396–1405. [Online]. Available: http://proceedings.mlr.press/v80/esfandiari18a.html

[10] W. Khaouid, M. Barsky, S. Venkatesh, and A. Thomo, "K-core decomposition of large networks on a single PC," *Proc. VLDB Endow.*, vol. 9, no. 1, pp. 13–23, 2015. [Online]. Available: http://www.vldb.org/pvldb/vol9/p13-khaouid.pdf

[11] N. S. Dasari, R. Desh, and M. Zubair, "Park: An efficient algorithm for k-core decomposition on multicore processors," in *2014 IEEE International Conference on Big Data (Big Data)*, 2014, pp. 9–16.

[12] Y. Fang, Y. Yang, W. Zhang, X. Lin, and X. Cao, "Effective and efficient community search over large heterogeneous information networks," *Proc. VLDB Endow.*, vol. 13, no. 6, pp. 854–867, 2020.

[13] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu, "Pathsim: Meta path-based top-k similarity search in heterogeneous information networks," *Proc. VLDB Endow.*, vol. 4, no. 11, pp. 992–1003, 2011. [Online]. Available: http://www.vldb.org/pvldb/vol4/p992-sun.pdf

[14] C. Peng, T. G. Kolda, and A. Pinar, "Accelerating community detection by using k-core subgraphs," *CoRR*, vol. abs/1403.2226, 2014. [Online]. Available: http://arxiv.org/abs/1403.2226

[15] J. García-Algarra, J. M. Pastor, J. M. Iriondo, and J. Galeano, "Ranking of critical species to preserve the functionality of mutualistic networks using the k-core decomposition," *PeerJ*, vol. 5, p. e3321, 2017. [Online]. Available: https://europepmc.org/articles/PMC5438587

[16] F. Geerts, "On the expressive power of linear algebra on graphs," *Theory Comput. Syst.*, vol. 65, no. 1, pp. 179–239, 2021. [Online]. Available: https://doi.org/10.1007/s00224-020-09990-9

[17] T. A. Davis, "Algorithm 1000: Suitesparse:graphblas: Graph algorithms in the language of sparse linear algebra," *ACM Trans. Math. Softw.*, vol. 45, no. 4, dec 2019. [Online]. Available: https://doi.org/10.1145/3322125

[18] Y. Yang, Y. Fang, X. Lin, and W. Zhang, "Effective and efficient truss computation over large heterogeneous information networks," in *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*. IEEE, 2020, pp. 901–912. [Online]. Available: https://doi.org/10.1109/ICDE48307.2020.00083

[19] L. Chang, J. X. Yu, L. Qin, X. Lin, C. Liu, and W. Liang, "Efficiently computing k-edge connected components via graph decomposition," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, K. A. Ross, D. Srivastava, and D. Papadias, Eds. ACM, 2013, pp. 205–216. [Online]. Available: https://doi.org/10.1145/2463676.2465323

[20] V. Batagelj, A. Mrvar, and M. Zaveršnik, "Partitioning approach to visualization of large graphs," in *Graph Drawing*, J. Kratochvíyl, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 90–97.

[21] V. Batagelj and M. Zaveršnik, "Fast algorithms for determining (generalized) core groups in social networks," *Advances in Data Analysis and Classification*, vol. 5, no. 2, pp. 129–145, July 2011.

[22] Q.-M. Z. Linyuan Lü, Tao Zhou and H. E. Stanley, "The h-index of a network node and its relation to degree and coreness," *Nature communications*, vol. 7, p. 10168, January 2016.

[23] C. S. Ahmet Erdem Sarıyüce and A. Pinar, "Local

algorithms for hierarchical dense subgraph discovery," *Proceedings of the VLDB Endowment*, vol. 12, no. 1, pp. 43–56, 2018.

[24] S. Chatzopoulos, T. Vergoulis, D. Skoutas, T. Dalamagas, C. Tryfonopoulos, and P. Karras, "Atrapos: Real-time evaluation of metapath query workloads," in *Proceedings of the ACM Web Conference 2023*, ser. WWW '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 2487–2498. [Online]. Available: https://doi.org/10.1145/3543507.3583322

[25] X. Jian, Y. Wang, and L. Chen, "Effective and efficient relational community detection and search in large dynamic heterogeneous information networks," *Proc. VLDB Endow.*, vol. 13, no. 10, pp. 1723–1736, 2020. [Online]. Available: http://www.vldb.org/pvldb/vol13/p1723-jian.pdf

[26] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in *The World Wide Web Conference*, ser. WWW '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 2022–2032. [Online]. Available: https://doi.org/10.1145/3308558.3313562

[27] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim, "Graph transformer networks," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019.

[28] F. G. Gustavson, "Two fast algorithms for sparse matrices: Multiplication and permuted transposition," *ACM Trans. Math. Softw.*, vol. 4, no. 3, pp. 250–269, 1978. [Online]. Available: https://doi.org/10.1145/355791.355796

[29] R. D. Luce, "A note on boolean matrix theory," *Proceedings of the American Mathematical Society*, vol. 3, no. 3, pp. 382–388, 1952.

[30] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, pp. 19:1–19:19, 2016. [Online]. Available: https://doi.org/10.1145/2827872

[31] J. J. McAuley, C. Targett, Q. Shi, and A. van den Hengel, "Image-based recommendations on styles and substitutes," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, Santiago, Chile, August 9-13, 2015*, R. Baeza-Yates, M. Lalmas, A. Moffat, and B. A. Ribeiro-Neto, Eds. ACM, 2015, pp. 43–52. [Online]. Available: https://doi.org/10.1145/2766462.2767755

[32] K. D. Bollacker, C. Evans, P. K. Paritosh, T. Sturge, and J. Taylor, "Freebase: a collaboratively created graph database for structuring human knowledge," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, J. T. Wang, Ed. ACM, 2008, pp. 1247–1250. [Online]. Available: https://doi.org/10.1145/1376616.1376746

[33] R. Socher, D. Chen, C. D. Manning, and A. Y. Ng, "Reasoning with neural tensor networks for knowledge base completion," in *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, Eds., 2013, pp. 926–934.

[34] M. De Domenico, A. Lima, P. Mougel, and M. Musolesi, "The anatomy of a scientific rumor," *Scientific Reports*, vol. 3, no. 2980, 2013. [Online]. Available: https://doi.org/10.1038/srep02980

[35] R. Speer, J. Chin, and C. Havasi, "Conceptnet 5.5: An open multilingual graph of general knowledge," pp. 4444–4451, 2017.