gCore: Exploring Cross-layer Cohesiveness in Multi-layer Graphs

Dandan Liu
Harbin Institute of Technology
Harbin, Heilongjiang, China
ddliu@hit.edu.cn

Zhaonian Zou
Harbin Institute of Technology
Harbin, Heilongjiang, China
znzou@hit.edu.cn

ABSTRACT

As multi-layer graphs can give a more accurate and reliable picture of the complex relationships between entities, cohesive subgraph mining, a fundamental task in graph analysis, has been studied on multi-layer graphs in the literature. However, existing cohesive subgraph models are designated for special multi-layer graphs such as multiplex networks and heterogeneous information networks. In this paper, we propose generalized core (gCore), a new notion of cohesive subgraph on general multi-layer graphs without any predefined constraints on the interconnections between vertices. The gCore model considers both the intra-layer and cross-layer cohesiveness of vertices. Three related problems are studied in this paper including gCore search (GCS), gCore decomposition (GCD), and gCore indexing (GCI). A polynomial-time algorithm based on the peeling paradigm is proposed to solve the GCS problem. By considering the containment among gCores, a "tree of trees" data structure called KP-tree is designed for efficiently solving the GCD problem and serving as a compact storage and index of all gCores. Several advanced lossless compaction techniques including node/subtree elimination, subtree transplant, and subtree merge are proposed to help reduce the storage overhead of the KP-tree and speed up the process of solving GCD and GCI. Besides, a KP-tree-based GCS algorithm is designed, which can retrieve any gCore in linear time in the size of the gCore and the height of the KP-tree. The experiments on 10 real-world graphs verify the effectiveness of the gCore model and the efficiency of the proposed algorithms.

PVLDB Reference Format:
Dandan Liu and Zhaonian Zou. gCore: Exploring Cross-layer Cohesiveness in Multi-layer Graphs. PVLDB, 16(11): 3201 - 3213, 2023.
doi:10.14778/3611479.3611519

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/SSSuperDan/gCore.

1 INTRODUCTION

Motivation. Cohesive subgraph mining (CSM), as a fundamental task in graph analysis, aims at finding densely connected vertices. It has witnessed considerable applications, such as community detection/search [6], graph visualization [1], product promotion [9], and biological module discovery [37]. While CSM has been extensively studied on single-layer graphs [20], the limitations of single-layer graphs in capturing multiple types of relationships, such as diverse social relationships [8], have led to growing research interest in CSM on *multi-layer graphs* (referred to as ML-CSM) [3, 12, 18, 22, 37].

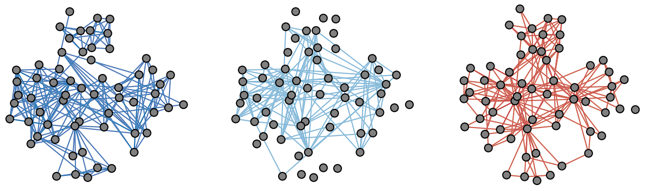


Figure 1: An example of a pillar multi-layer graph extracted from the AUCS dataset [8] with three layers representing different relationships between employees in a university.

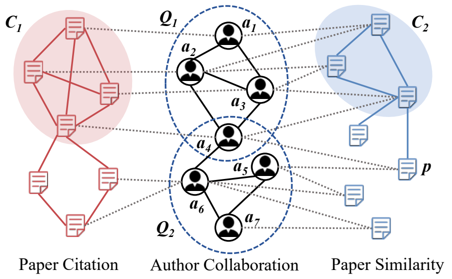


Figure 2: An example of a general multi-layer graph representing a 3-layer academic network. The layers from left to right represent the paper citation relationships, author collaborations, and paper similarities. Each author is connected to his/hers published papers through cross-layer links.

A multi-layer graph is represented as a collection of interconnected layered graphs (layers for short), with each layer corresponding to a specific type of relationship [4, 19]. There are two main kinds of multi-layer graphs [18]. Figure 1 gives an example of a *pillar multi-layer graph* that has the same set of vertices (entities) on all layers, and every vertex in a layer has exactly one cross-layer link to its copy (mirror) on every other layer. In contrast, a *general multi-layer graph*, as depicted in Figure 2, allows different sets of vertices in different layers, and any vertex in a layer has zero to many cross-layer links to the vertices on every other layer.

Based on the multi-faceted relationships captured by multi-layer graphs, ML-CSM enables the discovery of more reliable cohesive subgraphs [28, 37]. For instance, in the general multi-layer graph shown in Figure 2, cohesive author groups are of our interest. Obviously, the vertices in Q_1 form a cohesive group in the author layer. Moreover, their cross-layer neighbors, representing the papers they

Proceedings of the VLDB Endowment, Vol. 16, No. 11 ISSN 2150-8097.
doi:10.14778/3611479.3611519

have published, also show dense connections in terms of both similarity and citation relationships. This information complements the existing author group and enhances its cohesiveness.

Existing work investigates ML-CSM on a special *pillar* multi-layer graph model named **m**ulti**p**lex **n**etwork (MPN) [11, 12, 37] and a typical *general* multi-layer graph model termed **h**eterogeneous **i**nformation **n**etwork (HIN) [10, 17, 34]. MPNs are commonly employed when diverse relationships among entities of the same type are considered, while HINs are used to study ML-CSM in the context of complex relationships between heterogeneous entities.

Unfortunately, there is currently a lack of cohesive subgraph models and algorithms that can effectively handle a generic scenario involving relationships between both homogeneous entities and heterogeneous entities, i.e., on a **g**eneral **m**ulti-layer **g**raph (GMG), as illustrated in Figure 2.

Prior Work. In the literature, two major types of cohesive subgraph models have been proposed for solving ML-CSM on MPNs, namely *cross-layer quasi-clique* [5, 28, 35] and *multi-layer core* [11, 22, 37]. They can be seen as simple extensions from the classical γ-quasi-clique [25] model and the k-core [29] model in single-layer graphs. Specifically, a cross-layer quasi-clique (resp. multi-layer core) is defined as a subset of vertices that forms a γ-quasi-clique (resp. k-core) in every layer. Certainly, such concepts are not suitable for GMGs where different layers may have different vertex sets.

A well-known branch of ML-CSM models proposed for HINs such as $(k, \mathcal{P})$-core [10] and (k, Ψ)-NMC [17] are based on *meta-paths*, which are sequences of vertex types and edge types between two given vertex types. Meta-paths define new adjacency relationships between vertices. For example, in Figure 2, authors a_4 and a_5 are considered adjacent under a commonly used meta-path $\mathcal{P}^* =$ author-paper-author as they both connect to the paper p. Given a meta-path $\mathcal{P}$ and a set Ψ of meta-paths, the $(k, \mathcal{P})$-core [10] is exactly the k-core [29] of the graph describing the new adjacency relationship defined by $\mathcal{P}$, and the (k, Ψ)-NMC [17] refers to the multi-layer core [37] of the MPN defined by meta-paths in Ψ. However, these models suffer from the following limitations: (1) They overlook the most direct and informative relationships between homogeneous entities, such as the co-authoring among authors and the similarity between papers, as shown in Figure 2. (2) Meta-path instances only reflect the connections between entities under a specific pattern, lacking the ability to capture the cohesiveness among intermediate entities. For example, when considering the author collaboration and paper similarity layers in Figure 2, the vertices in Q_1 and Q_2 yield the same adjacency pattern under meta-path $\mathcal{P}^*$. The information on the cohesiveness of their connected papers is completely hidden from higher-level models and algorithms.

Another type of ML-CSM model designed for HINs, namely *relational community* [15], finds a vertex set that satisfies a collection of user-specified constraints describing "each vertex of type A has $\geq k$ neighbors of type B". However, it's challenging for users unfamiliar with the HIN schema to provide meaningful constraints [17]. Additionally, as each constraint is imposed on a pair of vertex types, it fails to distinguish multiple relationships between homogeneous types of entities. For example, the constraints cannot express the requirement that each paper has $\geq k_1$ neighbors in the similarity layer and meanwhile $\geq k_2$ neighbors in the citation layer.

Solution. In this paper, we propose a new cohesive subgraph model called *generalized core (gCore)* to solve the ML-CSM problem on GMGs. It overcomes the limitations of the existing models discussed before. To obtain reliable and robust cohesive subgraphs, we expect the vertices to show cohesiveness in each layer [16, 37]. A thorny problem is to define the cohesiveness of vertices on different layers based on the many-to-many cross-layer mappings between vertices. Our idea is to use a *fraction* [36] of cross-layer neighbors of a vertex to represent its engagement in different layers. Then, by extending the extension scheme used in cross-layer quasi-cliques [16, 28] and multi-layer cores [11] with the k-core model [29], which requires each vertex to connect to at least k other vertices in the k-core, we have the gCore model described below. Suppose there are l layers in a GMG and the l-th layer is of users' interest. Let $k_1, k_2, \ldots, k_l \in \mathbb{N}$ and $p_1, p_2, \ldots, p_{l-1} \in [0, 1]$. A subset Q of vertices on the l-th layer forms a gCore if 1) Q is a k_l-core on the l-th layer; 2) in each other i-th layer, there exists a k_i-core $Q' \subseteq N(Q)$ such that every vertex in Q has at least a fraction p_i of neighbors on the i-th layer participating Q', where $N(Q)$ is the set of all vertices in the i-th layer that link to vertices in Q; and 3) Q is maximal.

Let us see an example. In the GMG shown in Figure 2, we number the layers from left to right as 1, 3, and 2. Given $(k_1, k_2, k_3) = (3, 2, 2)$ and $(p_1, p_2) = (1, 0.5)$, Q_1 is a gCore. Specifically, Q_1 itself forms a 2-core in the author layer. C_1 and C_2 are a 3-core and 2-core on the paper citation and similarity layers, respectively, covering all and at least a half of cross-layer neighbors of each vertex in Q_1. Obviously, the authors in the gCore exhibit cohesiveness in both their direct connections and each possible relationship between their published papers. Q_2 is not a gCore due to the low similarity between the papers published by the authors in this group.

Application. Here are typical applications of the gCore model: (1) Collaboration analysis. On an academic multi-layer network like Figure 2, gCore facilitates the identification of cohesive groups of authors who have collaborated on a series of similar papers, possibly on the same topic. These groups are well-suited to be invited to present tutorials on the topic and share their related papers. (2) Recommendation/Product promotion. E-commerce platforms often maintain HINs of products (items) and users [10], while social media platforms like Last.fm (https://www.last.fm/) have bipartite data describing the "like" relationships between users and shared items. By enriching the existing HINs with additional homogeneous relationships like friendships or similarities between users, as well as similarities between items, gCore can be utilized to discover groups of users who are not only friends but also share similar preferences in specific items. This information can be leveraged to boost sales or user engagement by recommending the items purchased or liked by users in the group to other users within the same group. (3) Biological analysis. Diseases can exhibit similarities in various ways, such as sharing a significant number of therapies [7] or having commonly associated pathways [21]. Besides, communities of genes can also unveil similarities between diseases [31]. By representing these relationships in a GMG, gCore can be employed to identify meaningful disease clusters, providing new insights into disease etiology, classification, and shared biological mechanisms [21].

Technical Contributions. To address the gap in solving the ML-CSM problem on GMGs, we propose the gCore model and fully investigate its properties. One notable property is that given vectors

$\mathbf{k} = (k_1, k_2, \ldots, k_l)$ and $\mathbf{p} = (p_1, p_2, \ldots, p_{l-1})$, there exists a unique gCore, which we therefore refer to as the $(\mathbf{k}, \mathbf{p})$-core.

Based on the gCore model, we study three related problems in this paper, namely *gCore search (GCS)*, *gCore decomposition (GCD)*, and *gCore indexing (GCI)*. GCS finds the $(\mathbf{k}, \mathbf{p})$-core in an l-layer GMG for given vectors $\mathbf{k}$ and $\mathbf{p}$. This is crucial for retrieving cohesive subgraphs with particular features. GCD finds all nonempty gCores in a GMG, which is essential for exploring the structure of a GMG. GCI constructs an index to speed up GCS.

To solve the GCS problem, we propose a polynomial-time algorithm based on the *vertex peeling* paradigm.

To address the GCD problem, we propose a "tree of trees" data structure called *KP-tree* to organize all gCores in a systematic way by leveraging their containment relationships. The KP-tree determines a DFS-order generation of all gCores, which reduces redundant computations and enables fast identification of empty gCores.

Moreover, the KP-tree derives an index structure to support fast gCore retrieval. Based on this index, a more efficient algorithm is proposed to solve the GCS problem, which runs in linear time in the size of the queried gCore and the height of the KP-tree. Due to the redundancy of information in the KP-tree index, we design a series of lossless compaction schemes, including node/subtree elimination, subtree transplant, and subtree merge, to further save storage and improve the index construction efficiency.

Extensive experiments have been conducted to evaluate the gCore model and the proposed algorithms. The results show that: (1) Vertices in the gCore attain more closeness with each other compared with those in the k-core and other variations of the existing ML-CSM models adapted to suit GMGs. (2) The index helps improve the efficiency of gCore search by up to 1–4 orders of magnitude. (3) The index compaction techniques significantly reduce both the index construction time, which includes the time to solve the GCD problem, and the storage overhead of the index.

2 RELATED WORK

CSM on MPNs. To characterize cohesive subgraphs on an MPN, existing work always uses a unified classic cohesive subgraph model to restrict the cohesiveness of vertices in some/all layers. Wang et al. [33] and Zeng et al. [35] studied to mine maximal frequent cliques and γ-quasi-cliques in a series of graphs. Pei et al. [28] and Jiang et al. [16] extended the notion of quasi-clique and introduced the cross-layer quasi-clique model and the frequent cross-layer quasi-clique model. Due to the NP-hardness of enumerating all (frequent) cross-layer quasi-cliques, exact branch and bound approaches with several pruning methods are proposed. Boden et al. [5] introduced an MLCS cluster model for MPNs with edge labels, which is a variant of cross-layer quasi-clique with consideration of similarities between edge labels. A best-first search approach is presented in [5] to find qualified MLCS clusters with low redundancy.

Due to the high computational cost and low flexibility in characterizing large cohesive subgraphs of the cross-layer quasi-cliques, Zhu et al. [37] proposed a notion of d-coherent core (d-CC), which requires a vertex subset to form a d-core in a given subset of layers. Three approximation algorithms with provable guarantees are proposed in [37] to extract a fixed number of d-CCs that attain the largest diversity. Liu et al. [22] studied the d-CC decomposition problem. Galimberti et al. [11] studied the multi-layer core model, another extension of k-core. It allows using various values of k in different layers. Three decomposition algorithms based on different search orders are given in [11]. Hashemi et al. [12] introduced a relaxed version of the d-CC model called (k, λ)-FirmCore and studied the FirmCore decomposition problem. Huang et al. [14] and Behrouz et al. [3] further combined the extension scheme of d-CC and (k, λ)-FirmCore with the k-truss model, respectively. All the above models rely on the one-to-one cross-layer mappings between vertices, which are hence inapplicable to GMGs.

CSM on HINs. Fang et al. [9] surveyed the existing CSM models and algorithms designed for HINs. Here, we review several representative works. Fang et al. [10] defined a $(k, \mathcal{P})$-core model based on a given symmetric meta-path $\mathcal{P}$, which requires each vertex in a $(k, \mathcal{P})$-core is connected to at least k vertices through instances of $\mathcal{P}$. Two variants of $(k, \mathcal{P})$-core that require the meta-path instances contributing to k are vertex-disjoint and edge-disjoint are also studied in [10]. Yang et al. [34] combined the extension scheme of $(k, \mathcal{P})$-core and the vertex-disjoint $(k, \mathcal{P})$-core with the k-truss model. Jian et al. [15] proposed another extension of k-core called relational community. It allows using a series of meta-paths $\mathcal{P}$ of fixed length 2 and integers k specific to each $\mathcal{P}$. A relational community may contain heterogeneous vertices. Hu et al. [13] designed an extension of the clique model called m-Clique based on a given meta-subgraph. Due to the challenge of selecting meaningful meta-paths or relational constraints, Jiang et al. [17] introduced a (k, Ψ)-NMC model, where Ψ is a set of non-nested meta-paths. A set of vertices is a (k, Ψ)-NMC if it forms a $(k, \mathcal{P})$-core for each meta-path $\mathcal{P} \in \Psi$. Algorithms to search the (k, Ψ)-NMC containing a given set of query vertices while maximizing the size of Ψ are proposed in [17]. However, as discussed in Section 1, the above models cannot be directly applicable to GMGs that involve multiple types of relationships between homogeneous entities.

3 PRELIMINARIES

3.1 The General Multi-layer Graph Model

A simple graph $G = (V, E)$ is a pair, where V is the set of vertices, and E is the set of edges. By joining multiple simple graphs via the connections among them, we have a general multi-layer graph:

Definition 3.1 ([24]). A *general multi-layer graph* (GMG) is a pair $\mathcal{M} = (\mathcal{G}, \mathcal{C})$, where $\mathcal{G} = \{G_1, G_2, \ldots, G_l\}$ is a set of simple graphs (also called *layers*) $G_i = (V_i, E_i)$, where $i \in \{1, 2, \cdots, l\}$, and $\mathcal{C} = \{E_{i,j} | 1 \le i < j \le l\}$ is the collection of sets of edges $E_{i,j} \subseteq V_i \times V_j$ linking the vertices of G_i to the vertices of G_j.

We denote the edges in $E_1, E_2, \ldots, E_l$ as "*intra-layer edges*" and the edges in $E_{i,j}$ as "*cross-layer edges*". We use $E(\mathcal{G})$ and $E(\mathcal{C})$ to denote the set of all intra-layer edges and all cross-layer edges, respectively, i.e., $E(\mathcal{G}) = \bigcup_{i=1}^{l} E_i$ and $E(\mathcal{C}) = \bigcup_{E_{i,j} \in \mathcal{C}} E_{i,j}$.

Obviously, pillar multi-layer graphs are special cases of GMGs where all layers have identical vertex sets, and a cross-layer edge only exists between two copies of a vertex lying in different layers.

In GMGs, a vertex $v \in V_i$ can have both intra-layer and cross-layer neighbors. The set of neighbors of v in G_j is denoted by $N_j(v)$. If $j = i$, the vertices in $N_j(v)$ are called *intra-layer neighbors*; otherwise, they are called *cross-layer neighbors*. The degree of v in

G_j is denoted as $deg_j(v)$, where $deg_j(v) = |N_j(v)|$ is the *intra-layer degree* of v if $j = i$, and the *cross-layer degree* of v otherwise.

A GMG $\mathcal{M}' = (\mathcal{G}', \mathcal{C}')$ is a subgraph of $\mathcal{M}$ if $|\mathcal{G}| = |\mathcal{G}'|$, G'_i is a subgraph of G_i for $G'_i \in \mathcal{G}'$, and $E'_{i,j} \subseteq E_{i,j}$ for $E'_{i,j} \in \mathcal{C}'$. Given a simple graph $G = (V, E)$, the subgraph of G *induced* by $Q \subseteq V$ is $G[Q] = (Q, E[Q])$, where $E[Q] \subseteq E$ is the set of edges with both endpoints in Q. Similarly, let $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_l\}$, where $Q_i \subseteq V_i$ for $1 \le i \le l$, the subgraph of $\mathcal{M}$ induced by $\mathcal{Q}$ is $\mathcal{M}[\mathcal{Q}] = (\mathcal{G}[\mathcal{Q}], \mathcal{C}[\mathcal{Q}])$, where $\mathcal{G}[\mathcal{Q}] = \{G_1[Q_1], G_2[Q_2], \ldots, G_l[Q_l]\}$ and $\mathcal{C}[\mathcal{Q}] = \{E_{i,j}[Q_i, Q_j] | 1 \le i < j \le l\}$ with $E_{i,j}[Q_i, Q_j] \subseteq E_{i,j}$ being the set of edges having one endpoint in Q_i and the other in Q_j.

Besides, the presence of cross-layer edges in a GMG enables us to define cross-layer induced subgraphs. For a vertex subset $Q_i \subseteq V_i$ and $j \neq i$, let $Q_j = \bigcup_{v \in Q_i} N_j(v)$ be Q_i's cross-layer neighbors in G_j. The induced subgraph $G_j[Q_j]$ is called the *cross-layer subgraph* of G_j induced by Q_i, denoted by $G_j[Q_i]$.

3.2 gCores in General Multi-layer Graphs

As stated in Section 1, a generalized core (gCore) Q on the layer of users' interest, say G_i, is expected to show cohesiveness under various relationships, including the direct one captured by G_i and the indirect ones represented by $E_{i,j}$ and G_j for $j = 1, 2, \cdots, l$ and $j \neq i$. The former can be characterized by the elegant k-core model [29], which requires each vertex $v \in Q$ to be adjacent to at least k vertices in $G_i[Q]$, i.e., the degree of v in $G_i[Q]$ is at least k. To characterize the latter, we extend the concept of *fraction* proposed in [36] and rephrase it as *neighbor coverage fraction* in Definition 3.2.

Definition 3.2. For $v \in V_i$ and $Q_j \subseteq V_j$, the fraction of v's neighbors in G_j falling in Q_j, i.e., $\phi(v, Q_j) = \frac{|N_j(v) \cap Q_j|}{|N_j(v)|}$, is called the *neighbor coverage fraction* of v within Q_j.

A higher value of $\phi(v, Q_j)$ indicates that Q_j covers more neighbors of v in G_j, thereby capturing more information about v's cross-layer neighborhood. Based on Definition 3.2, we can define the concept of generalized core (gCore) on GMGs.

Definition 3.3. Given an l-layer graph $\mathcal{M}$, a specific layer G_i, l thresholds $k_1, k_2, \ldots, k_l \in \mathbb{N}$ and $p_{i,j} \in [0, 1]$ for $j = 1, 2, \ldots, l$ and $j \neq i$, a vertex subset $Q_i \subseteq V_i$ is a *generalized core (gCore)* in $\mathcal{M}$ if

(1) Q_i is a k_i-core in G_i.
(2) For $j = 1, 2, \ldots, l$ and $j \neq i$, there exists a nonempty k_j-core Q_j in $G_j[Q_i]$ such that $\phi(v, Q_j) \ge p_{i,j}$ for all $v \in Q_i$.
(3) None of the proper supersets of Q_i satisfies (1) and (2).

Without loss of generality, we suppose the selected layer in Definition 3.3 is G_l in the rest of the paper. For simplicity, let $\mathbf{k} = (k_1, k_2, \ldots, k_l)$ and $\mathbf{p} = (p_{l,1}, p_{l,2}, \ldots, p_{l,l-1})$. A gCore defined with respect to (w.r.t.) vectors $\mathbf{k}$ and $\mathbf{p}$ is referred to as a $(\mathbf{k}, \mathbf{p})$-core.

Example. Figure 3 shows a 3-layer GMG, where G_2 is of users' interest. Let $\mathbf{k} = (3, 3, 3)$, $\mathbf{p}_1 = (0, 0)$, $\mathbf{p}_2 = (1/2, 0)$, and $\mathbf{p}_3 = (1/2, 2/3)$. The entire vertex set of G_2, V_2, forms a $(\mathbf{k}, \mathbf{p}_1)$-core as $\mathbf{p}_1$ does not impose any constraints on the cohesiveness of the vertices on G_0 and G_1, and V_2 itself is a 3-core. The vertex set $\{1, 2, 3, 4, 5, 6, 7, 8\}$ forms a $(\mathbf{k}, \mathbf{p}_2)$-core. Vertices 9 and 10 are excluded from the $(\mathbf{k}, \mathbf{p}_2)$-core because they have no cross-layer neighbors in G_0. $Q = \{1, 2, 3, 4\}$ is a $(\mathbf{k}, \mathbf{p}_3)$-core. The sets $\{22, 23, 24, 25, 26\}$ and $\{13, 14, 15, 16, 17\}$ of

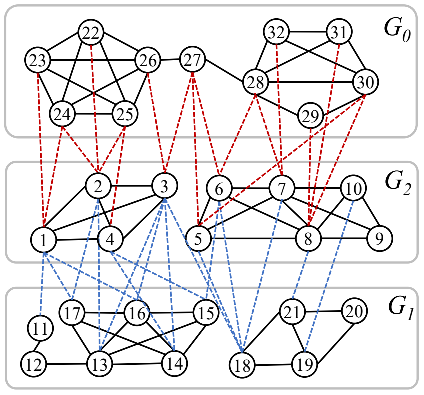


Figure 3: An example of a general multi-layer graph (GMG).

vertices on G_0 and G_1 are both 3-cores, and they cover at least 1/2 and 2/3 neighbors of each vertex in Q, respectively.

Generalization. The gCore model is a generalization of the multi-layer core model [11] and the d-CC model [37] proposed for MPNs. Given an l-layer graph $\mathcal{M}$, if $\mathcal{M}$ is an MPN, a $(\mathbf{k}, \mathbf{p})$-core in $\mathcal{M}$, where $\mathbf{p} = \mathbf{1}^{l-1}$, is a multi-layer $\mathbf{k}$-core. Moreover, when considering a subset L of layers, the gCore w.r.t. $\mathbf{k} = [k_i]_{1 \le i \le l}$ and $\mathbf{p} = \mathbf{1}^{l-1}$, where $k_i = d$ if $G_i \in L$ and $k_i = 0$ otherwise, is a d-CC on L.

Properties. The concept of gCore has several elegant properties.

PROPERTY 1. *Given* $\mathbf{k}$ *and* $\mathbf{p}$*, the* $(\mathbf{k}, \mathbf{p})$*-core is unique.*

As a notation, let $\mathbf{x}$ and $\mathbf{y}$ be two vectors of the same dimension, we say $\mathbf{x} \le \mathbf{y}$ if $\mathbf{x}[i] \le \mathbf{y}[i]$ for every dimension i, and we say $\mathbf{x} < \mathbf{y}$ if $\mathbf{x} \le \mathbf{y}$ and $\mathbf{x} \neq \mathbf{y}$.

PROPERTY 2. *Given* $\mathbf{k}_1$, $\mathbf{k}_2$*, and* $\mathbf{p}$*, if* $\mathbf{k}_1 \le \mathbf{k}_2$*, the* $(\mathbf{k}_2, \mathbf{p})$*-core is a subset of the* $(\mathbf{k}_1, \mathbf{p})$*-core.*

PROPERTY 3. *Given* $\mathbf{k}$, $\mathbf{p}_1$*, and* $\mathbf{p}_2$*, if* $\mathbf{p}_1 \le \mathbf{p}_2$*, the* $(\mathbf{k}, \mathbf{p}_2)$*-core is a subset of the* $(\mathbf{k}, \mathbf{p}_1)$*-core.*

Due to limited space, we leave the proofs of all the properties, lemmas, and theorems in Appendix M of the full paper [23].

3.3 Problem Formulation

This paper studies the following three problems related to the proposed gCore model on general multi-layer graphs:

(1) **gCore Search (GCS).** Given an l-layer graph $\mathcal{M}$, $\mathbf{k} \in \mathbb{N}^l$, and $\mathbf{p} \in [0, 1]^{l-1}$, find the $(\mathbf{k}, \mathbf{p})$-core in $\mathcal{M}$.
(2) **gCore Decomposition (GCD).** Given an l-layer graph $\mathcal{M}$, enumerate all nonempty gCores in $\mathcal{M}$.
(3) **gCore Indexing (GCI).** Given an l-layer graph $\mathcal{M}$, store all nonempty gCores in $\mathcal{M}$ in a compact data structure to support fast retrieval of the $(\mathbf{k}, \mathbf{p})$-core for given $\mathbf{k}$ and $\mathbf{p}$.

4 GCORE SEARCH (GCS)

Given an l-layer graph $\mathcal{M}$, $\mathbf{k} \in \mathbb{N}^l$, and $\mathbf{p} \in [0, 1]^{l-1}$, we propose a polynomial-time algorithm called GCS to find the $(\mathbf{k}, \mathbf{p})$-core in $\mathcal{M}$. The algorithm follows the *vertex peeling paradigm* that has been successfully used by the existing core search/decomposition

Algorithm 1 GCS (gCore Search)

```
Input: An $l$-layer graph $\mathcal{M}$, a vector $\mathbf{k} \in \mathbb{N}^l$, and a vector $\mathbf{p} \in [0,1]^{l-1}$
Output: The $(\mathbf{k}, \mathbf{p})$-core in $\mathcal{M}$
 1: $Q_l \leftarrow V_l$                              ▷ $V_l$ is the vertex set of $G_l$
 2: repeat
 3:     $Q_{before} \leftarrow Q_l$
 4:     $Q_l \leftarrow$ peel$(G_l[Q_l], \mathbf{k}[l])$
 5:     for $i \leftarrow 1, 2, \ldots, l-1$ do
 6:         $Q_i \leftarrow$ peel$(G_i[Q_l], \mathbf{k}[i])$
 7:         for $v \in Q_l$ do
 8:             if $\phi(v, Q_i) < \mathbf{p}[i]$ then
 9:                 $Q_l \leftarrow Q_l - \{v\}$
10: until $Q_l = Q_{before}$
11: return $Q_l$
```

algorithms [2], that is, it iteratively removes from G_l the vertices that violate Constraint (1) or (2) specified in Definition 3.3.

Algorithm 1 presents the pseudocode of GCS. We use Q_l to store the remaining vertices in G_l during vertex peeling. Initially, Q_l is set to the vertex set V_l of G_l in line 1. The **repeat** loop in lines 2–10 performs the peeling process on Q_l as follows. In line 3, a copy of Q_l is saved in Q_{before} to support checking the variation of Q_l in line 10. In line 4, we call the peel function to iteratively remove vertices with degrees less than $\mathbf{k}[l]$ from $G_l[Q_l]$ (i.e., violating Constraint (1) in Definition 3.3). The output is the $\mathbf{k}[l]$-core in $G_l[Q_l]$, which is used to update Q_l. Then, for each other layer G_i, where $1 \le i < l$, we compute the cross-layer subgraph of G_i induced by Q_l, i.e., $G_i[Q_l]$, and call peel to obtain the $\mathbf{k}[i]$-core in $G_i[Q_l]$. The result is saved in Q_i. In lines 7–9, we check the neighbor coverage fraction $\phi(v, Q_i)$ for each vertex $v \in Q_l$. If $\phi(v, Q_i) < \mathbf{p}[i]$, v violates Constraint (2) in Definition 3.3, so we remove v from Q_l (line 9). If Q_l is changed in this iteration, i.e., $Q_l \neq Q_{before}$, the **repeat** loop continues to peel more vertices. Otherwise, the algorithm returns Q_l and terminates.

THEOREM 4.1. *The output of Algorithm 1 is the* $(\mathbf{k}, \mathbf{p})$*-core in* $\mathcal{M}$.

Using well-designed arrays, Algorithm 1 can run in $O(|\mathcal{M}|+l|V_l|)$ time and $O(l \cdot V_{\max})$ space, where $|\mathcal{M}| = \sum_{i=1}^{l} |V_i| + |E(\mathcal{G})| + |E(C)|$ and $V_{\max} = \max_{1 \le i \le l} |V_i|$. The details are described in Appendix B of the full paper [23]. In addition, the complexity analysis for all algorithms can be found in Appendix N of [23].

5 GCORE DECOMPOSITION (GCD)

As with the multi-layer cores [11], the number of nonempty gCores can be exponential in the number of layers, and not all of them are nested into one another. Hence, we cannot expect to apply the vertex peeling paradigm used in core decomposition on simple graphs to solve the gCore decomposition (GCD) problem. To this end, we first present a naïve solution in Section 5.1. By exploiting the containment relationships among gCores as described in Properties 2 and 3, a tree-based structure called *KP-tree* is introduced to model the search space of the GCD problem in Section 5.2, which determines a good order for fast computing all gCores.

5.1 Naïve GCD Algorithm

A naïve solution to the GCD problem consists of two phases. In phase 1, we enumerate all $(\mathbf{k}, \mathbf{p})$ pairs such that the $(\mathbf{k}, \mathbf{p})$-core is nonempty. In phase 2, we use the GCS algorithm (Algorithm 1) to compute the $(\mathbf{k}, \mathbf{p})$-core for each $(\mathbf{k}, \mathbf{p})$ pair enumerated in phase 1. Obviously, phase 1 is the key to the algorithm design.

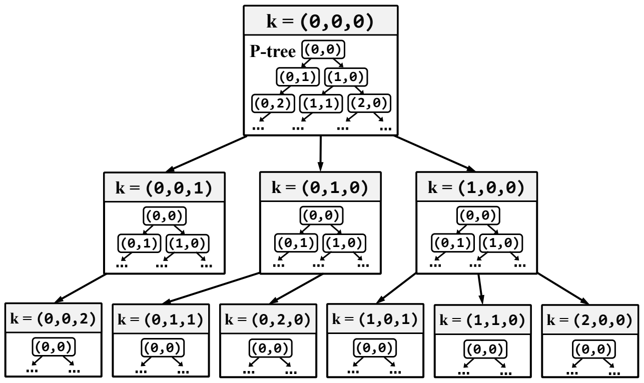


Figure 4: The structure of the KP-tree for a 3-layer graph.

All elements of $\mathbf{k}$ are integers. For $i = 1, 2, \ldots, l$, we have $0 \le \mathbf{k}[i] \le \kappa(G_i)$, where $\kappa(G_i)$ is the degeneracy of G_i, the largest integer k such that the k-core of G_i is nonempty. If $\mathbf{k}[i] > \kappa(G_i)$ for some $i \in \{1, 2, \ldots, l\}$, the $\mathbf{k}[i]$-core of G_i is certainly empty, thereby violating Constraint (1) or (2) specified in Definition 3.3. Therefore, we only need to consider possible values for $\mathbf{k}$ ranging from $(0, 0, \ldots, 0)$ to $(\kappa(G_1), \kappa(G_2), \ldots, \kappa(G_l))$.

Every element of $\mathbf{p}$ is a real number in $[0, 1]$. It is definitely infeasible to enumerate all values of $\mathbf{p}$. However, we found that it is sufficient to choose $\mathbf{p}[i]$ from the finite set F_i defined below.

LEMMA 5.1. *For any* $(\mathbf{k}, \mathbf{p})$*-core* Q *in an* l*-layer graph* $\mathcal{M}$*, there exists a vector* $\hat{\mathbf{p}}$ *with* $\hat{\mathbf{p}}[i] \in F_i$ *for* $i = 1, 2, \cdots, l-1$ *such that the* $(\mathbf{k}, \hat{\mathbf{p}})$*-core in* $\mathcal{M}$ *is identical to* Q*, where*

$$F_i = \left\{ \frac{j}{deg_i(v)} \middle| v \in V_l, j = 0, 1, \ldots, deg_i(v) \right\}. \tag{1}$$

After computing each F_i using Equation (1), we can then generate all possible values for $\mathbf{p}$.

The overall time complexity of the two-phase naïve solution is $O(\prod_{i=1}^{l} \kappa(G_i) \cdot \prod_{i=1}^{l-1} |F_i| \cdot (|\mathcal{M}| + l|V_l|) + \sum_{i=1}^{l-1} d_i^2 \log d_i)$, and the space complexity is $O(l \cdot (\prod_{i=1}^{l} \kappa(G_i) + \prod_{i=1}^{l-1} |F_i| + V_{max}))$, where d_i is the maximum cross-layer degree of vertices in V_l on layer G_i.

The naïve approach suffers from two main drawbacks. Firstly, it computes the $(\mathbf{k}, \mathbf{p})$-core for all enumerated $(\mathbf{k}, \mathbf{p})$ pairs, although many of them are empty. Secondly, each $(\mathbf{k}, \mathbf{p})$-core is independently computed by Algorithm 1 from scratch. A large number of repeated computations are carried out for different $(\mathbf{k}, \mathbf{p})$ pairs.

5.2 GCD Algorithm based on KP-trees

To overcome the disadvantages of the naïve approach, we design a "tree of trees" data structure called *KP-tree* to represent the search space of the GCD problem. We will show later that by traversing a KP-tree, it is possible to fast identify empty gCores and reduce repeated computations when computing different gCores.

5.2.1 **KP-trees.** As illustrated in Figure 4, the overall structure of a KP-tree is a tree. Each node in the KP-tree is associated with an l-dimensional vector $\mathbf{k} \in \mathbb{N}^l$. We call the node associated with a vector $\mathbf{k}$ the $\mathbf{k}$-*node*. The root of the KP-tree is the $\mathbf{0}^l$-node. In the

KP-tree, the $\mathbf{k}$-node is the parent of the $\mathbf{k}'$-node if and only if $\mathbf{k}'$ is an immediate suffix successor of $\mathbf{k}$ as defined below.

Definition 5.2. Let $\mathbf{x}$ and $\mathbf{y}$ be two integer vectors of the same dimension. $\mathbf{y}$ is an *immediate suffix successor* of $\mathbf{x}$ if

(1) $\mathbf{x}$ and $\mathbf{y}$ are different in exactly one element, say $\mathbf{x}[i]$ and $\mathbf{y}[i]$;
(2) $\mathbf{x}[i] + 1 = \mathbf{y}[i]$;
(3) $\mathbf{x}[j] = \mathbf{y}[j] = 0$ for all $j > i$.

Obviously, if $\mathbf{y}$ is an immediate suffix successor of $\mathbf{x}$, we have $\mathbf{x} < \mathbf{y}$. Let $end0(\mathbf{k})$ be the number of consecutive zeros at the end of a vector $\mathbf{k}$. It's easy to see that the $\mathbf{k}$-node in the KP-tree has $end0(\mathbf{k}) + 1$ children if $\mathbf{k} \neq \mathbf{0}^l$ and l children if $\mathbf{k} = \mathbf{0}^l$.

Inside every node of the KP-tree is nested another tree called *P-tree*. Each node in the *P*-tree is associated with an $(l-1)$-dimensional real vector $\mathbf{p} \in [0, 1]^{l-1}$, and the node is called the $\mathbf{p}$-node. In fact, as the i-th element of $\mathbf{p}$ is chosen from the set F_i of fractional numbers formulated in Equation (1), we actually store in each $\mathbf{p}[i]$ the index of the corresponding fractional value in F_i after sorting F_i in increasing order. It helps define the P-tree and save storage. Specifically, in a P-tree, the root is the $\mathbf{0}^{l-1}$-node, the $\mathbf{p}$-node is the parent of the $\mathbf{p}'$-node if and only if $\mathbf{p}'$ is an immediate suffix successor of $\mathbf{p}$. Thus, the $\mathbf{p}$-node has $end0(\mathbf{p})+1$ children if $\mathbf{p} \neq \mathbf{0}^{l-1}$ and $l-1$ children if $\mathbf{p} = \mathbf{0}^{l-1}$.

The following lemma ensures that the KP-tree is a systematic organization of gCores: a $\mathbf{p}$-node nested in a $\mathbf{k}$-node represents a distinct $(\mathbf{k}, \mathbf{p})$ pair that uniquely corresponds to the $(\mathbf{k}, \mathbf{p})$-core.

LEMMA 5.3. *For each possible value of* $\mathbf{k}$, *there is exactly one* $\mathbf{k}$-*node in the KP-tree. And in every* $\mathbf{k}$-*node, for each possible value of* $\mathbf{p}$, *there is exactly one* $\mathbf{p}$-*node in the P-tree nested in the* $\mathbf{k}$-*node.*

5.2.2 **KP-tree-based GCD Algorithm.** It's easy to see that both KP-tree and each P-tree inside the KP-tree maintain an ordering relationship "<" between the vectors associated with any pair of parent and child nodes. The following lemma utilizes these orders to overcome the disadvantages of the naïve method.

LEMMA 5.4. *Given an* l-*layer graph* $\mathcal{M}$, *for any* $(\mathbf{k}, \mathbf{p})$-*core* Q *and* $(\mathbf{k}', \mathbf{p}')$-*core* Q' *in* $\mathcal{M}$, $\mathcal{M}_{\mathbf{k}'}[Q']$ *is a subgraph of* $\mathcal{M}_{\mathbf{k}}[Q]$ *if* $\mathbf{k} \leq \mathbf{k}'$ *and* $\mathbf{p} \leq \mathbf{p}'$. *Here,* $\mathcal{M}_{\mathbf{k}}[Q]$ *denotes the subgraph of* $\mathcal{M}$ *induced by the set* $\{Q_1, Q_2, \cdots, Q_l\}$, *where* Q_i *is the* $\mathbf{k}[i]$-*core of* $G_i[Q]$ *for* $1 \leq i < l$, *and* $Q_l = Q$.

The lemma has two implications: (I1) If $Q = \emptyset$, we also have $Q' = \emptyset$, so it is unnecessary to compute Q' by the GCS algorithm (Algorithm 1); (I2) If $Q \neq \emptyset$, Q' can be computed on the subgraph $\mathcal{M}_{\mathbf{k}}[Q]$ instead of on the entire $\mathcal{M}$. Implication I1 enables fast identification of empty gCores, and Implication I2 helps reduce repeated computations when computing different gCores.

The two implications and the orders between vectors associated with parent and child nodes in the KP-tree and P-trees pave the way to a depth-first-search (DFS) order generation of all gCores, which computes gCores during the DFS on the KP-tree and the DFS on the P-tree nested in each visited $\mathbf{k}$-node (Algorithm 2). The pseudocode is self-explanatory. Note that the function GCS used in line 5 needs to be simply adapted to return Q_l as well as $Q_1, Q_2, \ldots, Q_{l-1}$, where each Q_i for $1 \leq i < l$ is obtained in line 6 of Algorithm 1.

THEOREM 5.5. *Algorithm 2 performs generalized core decomposition, returning all nonempty gCores in* $\mathcal{M}$.

Algorithm 2 GCD+ (KP-tree-based gCore Decomposition)

```
Input: An $l$-layer graph $\mathcal{M}$
Output: The collection $R$ of all nonempty gCores
 1: $R \leftarrow \emptyset$
 2: KPTreeDFS($\mathcal{M}, \mathbf{0}^l, R$)
 3: return $R$
 4: procedure PTreeDFS($\mathcal{M}, \mathbf{k}, \mathbf{p}, R$)
 5:     $\{Q_1, Q_2, \ldots, Q_l\} \leftarrow$ GCS($\mathcal{M}, \mathbf{k}$, ToFrac($\mathbf{p}$))
 6:     if $Q_l \neq \emptyset$ then
 7:         $R \leftarrow R \cup \{(Q_l, \mathbf{k}, \mathbf{p})\}$
 8:         for $i \leftarrow l-1, l-2, \ldots, l - end0(\mathbf{p}) - 1$ do
 9:             if $i > 0$ and $\mathbf{p}[i] < |F_i|$ then
10:                 $\mathbf{p}' \leftarrow \mathbf{p}$
11:                 $\mathbf{p}'[i] \leftarrow \mathbf{p}'[i] + 1$
12:                 PTreeDFS($\mathcal{M}[\{Q_1, Q_2, \ldots, Q_l\}], \mathbf{k}, \mathbf{p}', R$)
13:     return $\{Q_1, Q_2, \ldots, Q_l\}$
14: procedure KPTreeDFS($\mathcal{M}, \mathbf{k}, R$)
15:     $\{Q_1, Q_2, \ldots, Q_l\} \leftarrow$ PTreeDFS($\mathcal{M}, \mathbf{k}, \mathbf{0}^{l-1}, R$)
16:     if $Q_l \neq \emptyset$ then
17:         for $i \leftarrow l, l-1, \ldots, l - end0(\mathbf{k})$ do
18:             if $i > 0$ and $\mathbf{k}[i] < \kappa(G_i)$ then
19:                 $\mathbf{k}' \leftarrow \mathbf{k}$
20:                 $\mathbf{k}'[i] \leftarrow \mathbf{k}'[i] + 1$
21:                 KPTreeDFS($\mathcal{M}[\{Q_1, Q_2, \ldots, Q_l\}], \mathbf{k}', R$)
22: procedure ToFrac($\mathbf{p}$)
23:     return $(F_1[\mathbf{p}[1]], F_2[\mathbf{p}[2]], \ldots, F_{l-1}[\mathbf{p}[l-1]])$    ▷ Convert $\mathbf{p}$ to its fractional form
```

The time complexity of Algorithm 2 is $O(\prod_{i=1}^{l} \kappa(G_i) \cdot \prod_{i=1}^{l-2} |F_i| \cdot (|\mathcal{M}| + l|V_l|) + l \cdot \prod_{i=1}^{l} \kappa(G_i) \cdot \prod_{i=1}^{l-1} |F_i| + \sum_{i=1}^{l-1} d_i^2 \log d_i)$. The space complexity of Algorithm 2 is $O(l \cdot (V_{max} + \sum_{i=1}^{l-1} |F_i|))$.

6 GCORE INDEXING (GCI)

In this section, we study how to store and index the results of the GCD problem to enable fast search of any $(\mathbf{k}, \mathbf{p})$-core.

6.1 Storage and Index Structure

When GCD+ (Algorithm 2) terminates, it conceptually generates a subtree of the KP-tree where a $\mathbf{p}$-node nested in a $\mathbf{k}$-node uniquely represents a nonempty $(\mathbf{k}, \mathbf{p})$-core. By materializing this subtree of KP-tree, a storage and index structure can be designed.

Specifically, we use a hash table to map $\mathbf{k}$ to the $\mathbf{k}$-node to support fast locating each $\mathbf{k}$-node. To store all nonempty gCores in the P-tree nested in the $\mathbf{k}$-node, say T, we augment the structure of T in the following way:

(1) For any leaf node N in T, we add a dummy node L to represent an empty gCore and designate N as the parent of L.
(2) Let N and N' be two nodes in T representing gCores Q and Q', respectively, and N' is the *leftmost* child of N. We store the set $Q - Q'$ along with the edge between N and N'. N' is said to be the leftmost child of N if $N' \prec N''$ for all siblings N'' of N', where $\prec$ is a total order of N's children as given below.

Definition 6.1. Let N' and N'' be two children of a node N. We have $N' \prec N''$ if $end0(\mathbf{p}') < end0(\mathbf{p}'')$, where $\mathbf{p}'$ and $\mathbf{p}''$ are the vectors associated with N' and N'', respectively.

Figure 5(a) illustrates an augmented P-tree. Theorem 6.2 ensures that the augmented P-tree is a compact storage of all gCores corresponding to the nodes in the P-tree. In the rest of the paper, when we mention a KP-tree or a P-tree, we are referring to its augmented version that can serve as a storage and index structure.

THEOREM 6.2. *Let* N *be the node representing the gCore* Q. *We have that* Q *equals the union of the vertex sets associated with the edges on the leftmost path from* N *to a leaf node, where the leftmost*

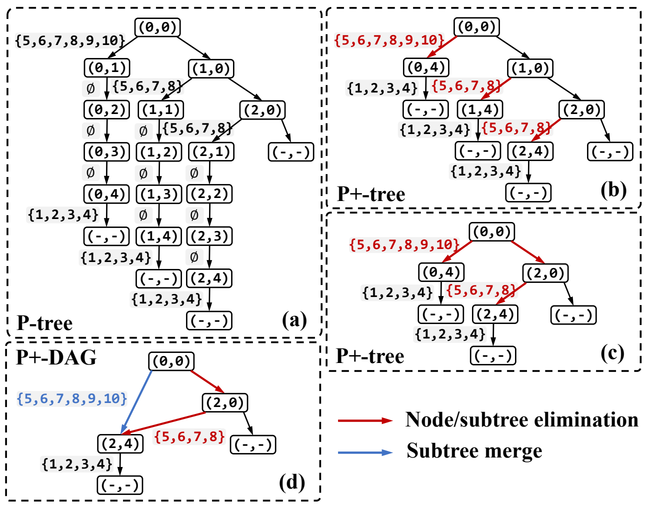


Figure 5: Augmented structures nested in the k-node of the KP-tree index of the example GMG in Figure 3, where k = (3, 3, 3). (a) The original augmented P-tree. (b)-(d) Compact structures obtained using different compaction techniques.

Algorithm 3 GCS+ (Index-based gCore Search)

```
Input: The KP-tree for an $l$-layer graph $\mathcal{M}$, a vector $\mathbf{k} \in \mathbb{N}^l$, and a vector $\mathbf{p} \in [0,1]^{l-1}$
Output: The $(\mathbf{k},\mathbf{p})$-core in $\mathcal{M}$
 1: Find the $\mathbf{k}$-node from the hash table of the KP-tree with the key $\mathbf{k}$
 2: $T \leftarrow$ the P-tree nested in the $\mathbf{k}$-node
 3: $N \leftarrow$ SEARCH$(T, \mathbf{p})$
 4: return RECOVER$(N)$
 5: procedure SEARCH$(T, \mathbf{p})$
 6:     $N \leftarrow$ the root of the P-tree $T$
 7:     $i \leftarrow 1$
 8:     while $i < l$ do
 9:         $(N, i) \leftarrow$ FORWARD$(N, i, \mathbf{p})$
10:     return $N$
11: procedure FORWARD$(N, i, \mathbf{p})$
12:     if $\mathbf{p}'[i] < \mathbf{p}[i]$ then        ▷ $\mathbf{p}'$ is the vector (in the fractional form) associated with $N$
13:         for each child $N'$ of $N$ do
14:             if the vectors associated with $N'$ and $N$ are different in their $i$-th element then
15:                 return $(N', i)$
16:     else
17:         return $(N, i+1)$
18: procedure RECOVER$(N)$
19:     $Q \leftarrow \emptyset$
20:     while $N$ is not a dummy leaf do
21:         $N' \leftarrow$ the leftmost child of $N$
22:         $S \leftarrow$ the set of vertices associated with the edge between $N$ and $N'$ in the P-tree
23:         $Q \leftarrow Q \cup S$
24:         $N \leftarrow N'$
25:     return $Q$
```

path of N is obtained by following the leftmost child of N, the leftmost child of the leftmost child of N … until a leaf node is reached.

6.2 Index-based GCS Algorithm

The KP-tree index proposed in Section 6.1 enables a fast two-phase gCore search (GCS) algorithm (Algorithm 3). Given vectors $\mathbf{k} \in \mathbb{N}^l$ and $\mathbf{p} \in [0,1]^{l-1}$, it retrieves the $(\mathbf{k},\mathbf{p})$-core as follows:

(1) Find a node N in the P-tree nested in the $\mathbf{k}$-node of the KP-tree that represents the $(\mathbf{k},\mathbf{p})$-core.
(2) Compute the union of all vertex sets associated with the edges on the leftmost path of N (Theorem 6.2).

The $\mathbf{k}$-node can be easily found from the hash table of the KP-tree with the key $\mathbf{k}$ (line 1). Since $\mathbf{p}$ is a real vector, it is very likely that $\mathbf{p} \notin F_1 \times F_2 \times \cdots \times F_{l-1}$, and therefore the $\mathbf{p}$-node is not indexed in the KP-tree. To handle this, we find a $\hat{\mathbf{p}}$-node in the P-tree such that the $(\mathbf{k},\mathbf{p})$-core is identical to the $(\mathbf{k},\hat{\mathbf{p}})$-core. The proof of Lemma 5.1 (Appendix M of [23]) provides us a method to efficiently identify $\hat{\mathbf{p}}$: For $i = 1, 2, \ldots, l-1$, $\hat{\mathbf{p}}[i]$ is the smallest element in F_i that is not less than $\mathbf{p}[i]$. If $\mathbf{p} \in F_1 \times F_2 \times \cdots \times F_{l-1}$, we have $\hat{\mathbf{p}} = \mathbf{p}$.

After knowing $\hat{\mathbf{p}}$, Algorithm 3 calls Procedure SEARCH (lines 5–10) in line 3 to search for the $\hat{\mathbf{p}}$-node. Then, Procedure RECOVER (lines 18–25) is called in line 4 to get the union of the vertex sets on the leftmost path of the $\hat{\mathbf{p}}$-node, which is returned as the $(\mathbf{k},\mathbf{p})$-core. The pseudocode is straightforward, so the description is omitted.

THEOREM 6.3. *Algorithm 3 returns the* $(\mathbf{k},\mathbf{p})$*-core in* $\mathcal{M}$.

Algorithm 3 runs in $O(\sum_{i=1}^{l-1} |F_i| + |Q|)$ time and $O(|Q|)$ space, where $|Q|$ is the size of the result $(\mathbf{k},\mathbf{p})$-core.

7 COMPACTION OF P-TREES

The gCores represented by the nodes in a P-tree may be not distinct. For example, in Figure 5(a), the $\mathbf{p}$-nodes for $\mathbf{p} = (2,1), (2,2), (2,3)$, and $(2,4)$ all represent the gCore $\{1,2,3,4\}$. Storing these nodes incurs unnecessary storage overheads and computational costs. In this section, we explore methods to eliminate unnecessary nodes from a P-tree while ensuring that all gCores can still be correctly retrieved using Algorithm 3 on the resulting compact structure.

7.1 Foundations

In a P-tree, two nodes N and N' are considered *redundant*, denoted as $N \cong N'$, if they represent the same gCore. Obviously, the binary relation $\cong$ is reflexive, symmetric, and transitive, making it an equivalence relation. The equivalence class of a node N under $\cong$ is denoted as $[N] = \{N' | N' \cong N\}$.

Definition 7.1. The vector $\mathbf{p}$ associated with a node $N \in [N]$ is *maximal* if $\mathbf{p}' \leq \mathbf{p}$ for all vectors $\mathbf{p}'$ associated with nodes $N' \in [N]$.

The maximal vector provides a unique characterization of the equivalence class $[N]$. Below are several theoretical foundations:

THEOREM 7.2. *The maximal vector for* $[N]$ *is unique.*

THEOREM 7.3. *Let* Q *be the* $(\mathbf{k},\mathbf{p})$*-core represented by* N *in the P-tree, and* $\hat{\mathbf{p}}$ *be the maximal vector for* $[N]$*. For* $i = 1, 2, \ldots, l-1$*, we have* $\hat{\mathbf{p}}[i] = \min_{v \in Q} \phi(v, Q_i)$*, where* Q_i *is the* $\mathbf{k}[i]$*-core of* $G_i[Q]$*.*

LEMMA 7.4. *For two nodes* N *and* N' *in a P-tree, we have* $N \cong N'$ *if and only if the maximal vectors for* $[N]$ *and* $[N']$ *are identical.*

7.2 Node Elimination

For any two nodes N and N' in a P-tree, if $N \cong N'$ and N' is N's only child, we can remove N because the gCore represented by N can be identically obtained by searching for the gCore represented by N'. If N has a parent, we link N' to N's parent as a new child. We call the process "node elimination".

Example. An example of node elimination is shown in Figure 6(a). Let N_0, N_1, and N_2 be the nodes with $\mathbf{p} = (2,0), (2,1)$, and $(2,2)$, respectively. We have $N_1 \cong N_2$ and N_2 is N_1's only child. Therefore, N_1 can be removed. After removing N_1, N_0 becomes N_2's parent.

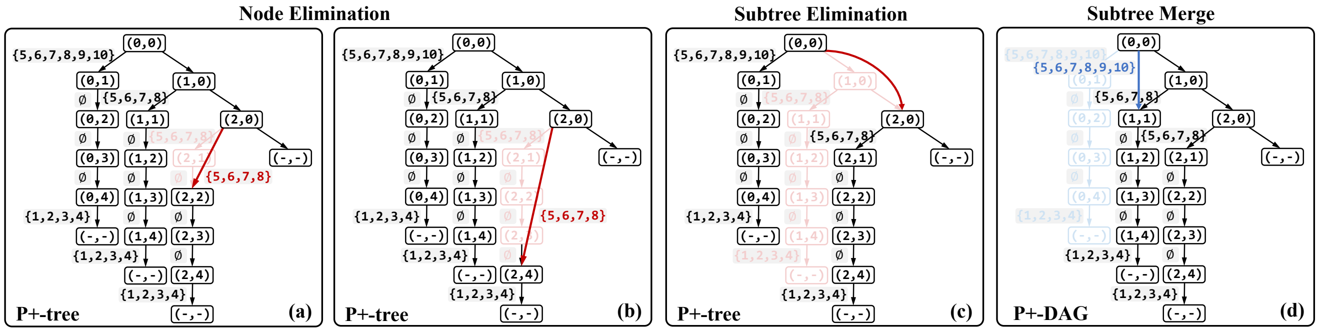


Figure 6: Illustration of P-tree compaction schemes: (a) and (b) node elimination, (c) subtree elimination, and (d) subtree merge.

After applying node elimination, the obtained tree is no longer a P-tree as the vector associated with N' is not an immediate suffix successor (Definition 5.2) of the new parent's vector. We name the new tree *P+-tree*, which satisfies the following property.

Property 4. *Let N and N' be two nodes in a P+-tree, and let $\mathbf{p}$ and $\mathbf{p}'$ be the vectors associated with N and N', respectively. If N is the parent of N', $\mathbf{p}'$ is a suffix successor of $\mathbf{p}$, that is: (1) $\mathbf{p}$ and $\mathbf{p}'$ are different in exactly one element, say $\mathbf{p}[i]$ and $\mathbf{p}'[i]$; (2) $\mathbf{p}[i] < \mathbf{p}'[i]$; and (3) $\mathbf{p}[j] = \mathbf{p}'[j] = 0$ for all $j > i$.*

The node elimination procedure presented above can be repeatedly applied to a P+-tree until no node can be removed anymore. The correctness of applying Algorithm 3 on a P+-tree to solve the GCS problem is guaranteed by the following theorem.

Theorem 7.5. *Given a P-tree and the P+-tree obtained by repeatedly applying the node elimination procedure, when the $(\mathbf{k}, \mathbf{p})$-core represented by a node N in the P-tree is queried, Procedure Search in Algorithm 3 returns a node N^* in the P+-tree. We have (1) $N \cong N^*$; (2) The sets (except $\emptyset$) on the leftmost path from N' to a leaf node in the P-tree are all retained on the leftmost path from N^* to a leaf node in the P+-tree, where N' is the copy of N^* in the P-tree.*

Example. The P+-tree shown in Figure 6(b) is obtained by eliminating the nodes with vectors $\mathbf{p} = (2, 1), (2, 2)$, and $(2, 3)$. When the $(\mathbf{k}, \mathbf{p})$-core for $\mathbf{p} = (2, 2)$ is queried, Procedure Search in Algorithm 3 returns the $\mathbf{p}^*$-node N^*, where $\mathbf{p}^* = (2, 4)$. The vertex set $\{1, 2, 3, 4\}$ on the leftmost path of N^* is returned as the $(\mathbf{k}, \mathbf{p})$-core.

Redundant Node Detection. Lemma 7.4 implies an easier way to test the equivalence between two nodes N and N' in a P-tree if N' is N's only child as described in the following theorem.

Theorem 7.6. *Let N and N' be two nodes in a P-tree, and N' is N's only child. Let $\hat{\mathbf{p}}$ be the maximal vector for $[N]$, and $\mathbf{p}'$ be the vector associated with N'. We have $N \cong N'$ if and only if $\mathbf{p}' \leq \hat{\mathbf{p}}$.*

7.3 Subtree Elimination

If a node N in a P-tree has more than one child, node elimination can never be applied to N. To overcome this limitation, we extend node elimination to subtree elimination in this subsection.

First, let us define some concepts. For a node N in a P-tree, based on the order $\prec$ (Definition 6.1) on N's children, we have a node N' is the rightmost child of N if $N'' \prec N'$ for all siblings N'' of N'. $N_1, N_2, \cdots, N_m$ is the rightmost path of N if $N_1 = N$, N_{i+1} is the rightmost child of N_i for $1 \leq i < m$, and N_m is a leaf node.

Definition 7.7. For a non-leaf node N in a P-tree, let N' be the rightmost child of N. The subtree rooted at N except the subtree rooted at N' is called the preceding subtree rooted at N.

Definition 7.8. Let T and T' be two subtrees in a P-tree rooted at R and R', respectively. T and T' are redundant if

(1) $R \cong R'$, that is, R and R' are redundant;
(2) R and R' have the same number of children, and for each i-th child N_i of R and the i-th child N_i' of R', the subtrees rooted at N_i and N_i' are redundant.

In other words, T is isomorphic to T' in terms of the node redundancy relation $\cong$. We denote it by $T \cong T'$.

If $T \cong T'$, T and T' have the same number of nodes, and there exists a bijection f from the nodes of T to the nodes of T':

$$f(N) = \begin{cases} R' & \text{if } N = R, \\ \text{the } i\text{-th child of } f(N') & \text{if } N \text{ is the } i\text{-th child of its parent } N' \end{cases} \quad (2)$$

Under the bijection f, we have: (1) $N \cong f(N)$ for all nodes N in T; and (2) N' is the parent of N in T if and only if $f(N')$ is the parent of $f(N)$ in T'. Therefore, every node N in T is redundant with regards to $f(N)$ in T', and all the redundant nodes in T make T a redundant subtree. To remove such redundant subtree from a P-tree, we propose the following subtree elimination procedure.

For two nodes N and N' in a P-tree, and N' is N's rightmost child, if the preceding subtree rooted at N, say T, is isomorphic to the preceding subtree rooted at N', say T', we remove T from the P-tree. If N has a parent, we link N' to N's parent as a new child. An example of subtree elimination is illustrated in Figure 6(c).

Clearly, node elimination presented in Section 7.2 is a special case of subtree elimination when N' is N's only child. After subtree elimination, the obtained tree is also a P+-tree. Hence, subtree elimination can also be repeatedly applied to a P+-tree. Theorem 7.5 still holds if "node elimination" is replaced by "subtree elimination".

Redundant Subtree Detection. By extending the redundant node detection method described in Theorem 7.6, we provide an efficient way to identify redundant subtrees in a P-tree. The basic idea is to compute a unique signature for a subtree as will be defined in Definition 7.9 and test if two subtrees are isomorphic by comparing their signatures using the following Theorem 7.10.

Definition 7.9. Let T be a subtree in a P-tree. The signature of T is defined as the element-wise minimum of the maximal vectors of $[N]$ for all nodes N in T.

Theorem 7.10. *Let N and N' be two nodes in a P-tree with vectors $\mathbf{p}$ and $\mathbf{p}'$, respectively, and N' is a descendant of N on N's rightmost path. Let T and T' be the preceding subtrees rooted at N and N', respectively. We have $T \cong T'$ if and only if $\mathbf{p}'[i] \leq \hat{\mathbf{p}}[i]$, where $\hat{\mathbf{p}}$ is the signature of T, and i is the dimension at which $\mathbf{p}$ differs from $\mathbf{p}'$.*

7.4 Subtree Transplant

Generating P-tree nodes in a DFS order (Algorithm 2) guarantees that for any node N and its rightmost child N' in a P-tree, the preceding subtree rooted at N, say T, is generated prior to N' and the preceding subtree T' rooted at N'. Once N' is generated, we can determine whether the subtrees T and T' are isomorphic using Theorem 7.10, even though T' has not been generated yet. If we find that $T \cong T'$, we can reuse the structure of T to directly obtain T' without removing T and generating T' from scratch. This approach is called "*subtree transplant*". Due to space limitations, the implementation details are given in Appendix F of [23].

7.5 Subtree Merge

The condition for subtree elimination is strict. For any preceding subtrees T and T' rooted at N and N', respectively, subtree elimination cannot be applied to T if N' is not N's rightmost child or T and T' are not isomorphic. The latter case is more likely to happen if N has more children. In this subsection, a *subtree merge* approach is proposed to further reduce the redundancy between T and T'.

The idea is to conceptually divide T and T' into smaller substructures called "branches", in which we seek redundant subtrees that can be eliminated. The concept of branch is defined as follows:

Definition 7.11. Given two nodes N and N' in a P-tree and N' is a child of N, the subtree formed by N and the subtree rooted at N' is called a branch, denoted by B_i^N, where i is the dimension at which the vectors associated with N and N' differ.

The *subtree merge* approach works as follows. Let N and R be two nodes in a P-tree, and N is a child of R. We consider each pair of branches B_i^R and B_i^N within the preceding subtrees of R and N, respectively. If there exist nodes R_1 and N_1 on the rightmost path of R in B_i^R and N in B_i^N, respectively, such that the subtrees rooted at them, say T and T', are isomorphic, we merge T into T'. This can be done by removing T and making N_1 a new child of R_1's parent.
Example. Figure 6(d) illustrates an example of subtree merge. Let R, R_1, N, N_1 be the nodes with vectors $\mathbf{p} = (0,0), (0,1), (1,0), (1,1)$, respectively. The subtrees rooted at R_1 and N_1 are denoted as T and T', respectively. R and T form the branch B_2^R, while N and T' form the branch B_2^N. In this example, B_2^R and B_2^N are exactly the preceding subtrees rooted at R and N, respectively. Although $B_2^R \ncong B_2^N$, the subtrees T and T' within them are isomorphic. Therefore, we merge T into T' by removing T and making R the parent of N_1.

Applying subtree merge, as shown in the example above, leads to a node having more than one parent, so the obtained data structure is no longer a tree but a directed acyclic graph (DAG). We call it P+-DAG. Obviously, subtree merge can be repeatedly applied to a P+-DAG until no subtrees can be merged.

Table 1: Properties of graphs used in experiments.

Graph	$\lvert\mathcal{V}\rvert$	$\lvert E(\mathcal{G})\rvert$	$\lvert E(C)\rvert$	#Vertex Types	l
SacchCere (SC) [11]	6750	247,152	39,420	1	7
ObamaInIsrael (Oii) [27]	2,279,535	3,827,964	4,559,070	1	3
Friendfeed (FF)[8]	505,104	18,673,521	1,010,208	1	3
6-NG [26]	4,500	15,787	24,001	5	5
9-NG [26]	6,750	24,264	36,015	5	5
DBLP [30]	41,892	280,707	381,176	2	2
Twitter[1]	47,280	445,287	89,775	3	3
Movie[2]	251,742	1,183,167	502,821	2	4
Aminer-5 [32]	2,890,443	14,536,094	7,730,034	3	5
Aminer-10 [32]	4,650,693	118,763,984	14,384,941	3	5

Notably, subtree elimination and subtree merge complement each other. By applying subtree merge to a P+-tree obtained by applying the subtree elimination procedure, we can get a more succinct storage of all gCores. The following theorem ensures that Algorithm 3 can be applied to solve the GCS problem on a P+-DAG.

Theorem 7.12. *Given a P+-tree and the P+-DAG obtained by repeatedly applying the subtree merge procedure, when the gCore represented by a node N in the P+-tree is queried, Procedure* Search *in Algorithm 3 returns a node N^* in the P+-DAG. We have (1) $N \cong N^*$; (2) The sets (except $\emptyset$) on the leftmost path from N' to a leaf node in the P+-tree are all retained on the leftmost path from N^* to a leaf node in the P+-DAG, where N' is the copy of N^* in the P+-tree.*

Redundant Subtree Detection. By utilizing the subtree signatures (Definition 7.9), we can determine the redundancy between two subtrees within separate branches. For the details, please refer to Appendix G of the full paper [23], where we also demonstrate an efficient implementation of subtree merge that runs in linear time in the lengths of the rightmost paths of two given branches.

Putting It All Together. Consider the P-tree shown in Figure 5(a), which consists of 19 nodes and 15 edges associated with vertex sets. Figure 5(b), (c), and (d) show the P+-tree/P+-DAG obtained by applying the proposed compaction techniques. Through repeated node elimination, 10 nodes and 6 edges associated with vertex sets are retained in the P+-tree, as shown in Figure 5(b). Subsequent applications of subtree elimination further reduce the size of the P+-tree, which has 7 nodes and 4 edges with sets (see Figure 5(c)). By continuing applying the subtree merge, we finally have a P+-DAG in Figure 5(d), in which only 5 nodes and 3 sets are retained.

8 EXPERIMENTS

8.1 Experimental Setup

We obtained the source code for (k, Ψ)-NMC search from [17] and implemented other algorithms in C++. All experiments were performed on a server with an Intel Xeon Gold 5218R processor and 754GB of RAM, running 64-bit Ubuntu 22.04.
Datasets. We select 3 real-world pillar multi-layer graphs and construct 7 GMGs using publicly available datasets. The statistics of these graphs are presented in Table 1, where $|\mathcal{V}|$ denotes the total number of distinct vertices among all layers, and $|E(C)|$ counts cross-layer edges with one endpoint on the layer of users' interest. The details of these graphs are given in Appendix H of [23].

[1]https://www.twitter.com/
[2]https://www.themoviedb.org/

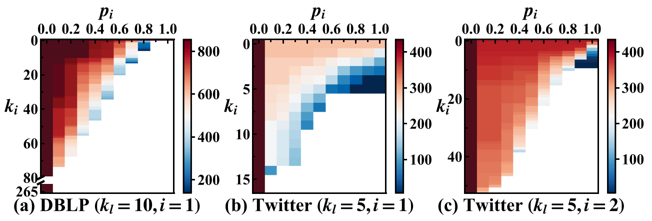


Figure 7: Size matrices of gCores on DBLP and Twitter.

Algorithms. To the best of our knowledge, no existing cohesive subgraph models can directly handle GMGs. Therefore, we compare our gCore model with the k-core [2], multi-layer core (d-CC) [37], and relational community [15] models designed for simple graphs, MPNs, and HINs, respectively. Besides, the meta-path-based (k, Ψ)-NMC [17] model is also compared in the effectiveness evaluation. Specifically, we compare the following algorithms:

- KC: k-core computation algorithm in [2]. By default, we run KC only on the layer of users' interest.
- DCC: d-CC computation algorithm in [37]. In our implementation, we extend DCC to compute the multi-layer $\mathbf{k}$-core.
- RCD: Relational community detection algorithm in [15]. We adapt the algorithm to GMGs by designating vertices in each layer G_i a type t_i and using constraints (t_i, t_i, k_i) for $1 \le i \le l$ and $(t_l, t_i, 1)$ for $1 \le i < l$ as the community schema. The result is denoted by $\mathbf{k}$-rc, where $\mathbf{k} = (k_1, k_2, \cdots, k_l)$.
- CSSH: The (k, Ψ)-NMC search algorithm in [17], which is adapted to GMGs as described in Appendix I of the full paper [23].
- GCS: Our gCore search algorithm (Algorithm 1).
- GCS+: Our KP-tree-based gCore search algorithm (Algorithm 3).
- TN: GCD+ (Algorithm 2) with KP-tree constructed (Section 6.1).
- TE: TN with node and subtree elimination (Sections 7.2–7.3).
- TM: TN with subtree merge (Section 7.5).
- TEM: TN with both subtree elimination and subtree merge.

Note that we do not evaluate the naïve GCD algorithm proposed in Section 5.1 because it is slow and cannot generate a KP-tree index to support gCore search. Besides, the subtree transplant technique introduced in Section 7.4 is incorporated in the implementations of subtree elimination and subtree merge.

8.2 Effectiveness Evaluation

8.2.1 gCore Sizes. To demonstrate how the cohesiveness constraint imposed on different layers affects gCores, we examine the size of the $(\mathbf{k}, \mathbf{p})$-core w.r.t. k_i and $p_{l,i}$ (p_i for short) for $1 \le i < l$. Figure 7 visualizes the size matrices obtained on DBLP and Twitter with rows and columns representing k_i and p_i, respectively, and each cell (x, y) representing the size of the $(\mathbf{k}, \mathbf{p})$-core with $k_i = x$ and $p_i = y$. When varying k_i and p_i, we fix $k_l = 10$ for DBLP and 5 for Twitter, and fix $k_j = 0$ and $p_j = 0$ for $1 \le j < l$ and $j \ne i$.

We can observe that the size of the $(\mathbf{k}, \mathbf{p})$-core is always monotonically decreasing when either k_i or p_i becomes larger. It's consistent with the containment relationships among gCores given in Properties 2 and 3. Note that all cells in the first column correspond to $p_i = 0$, meaning no constraint is imposed on the neighbor coverage fraction and thereby the cohesiveness of the vertices shown on G_i. In this case, the $(\mathbf{k}, \mathbf{p})$-core is exactly the k_l-core. A significant drop in size exhibits when increasing p_i from 0 to 0.1. This is because the k_l-core contains massive vertices with seldom, even no, cross-layer neighbors. By properly setting p_i, the $(\mathbf{k}, \mathbf{p})$-core can well filter out these vertices. Besides, a trade-off between higher intra-layer cohesiveness and higher neighbor coverage fraction, i.e., a larger k_i and a larger p_i, can be seen. Each boundary cell corresponding to the last nonempty $(\mathbf{k}, \mathbf{p})$-core as k_i or p_i increases exhibits the highest cohesiveness w.r.t. G_i for different k_i/p_i trade-offs.

8.2.2 Closeness. We compare the closeness of vertices in k-core, $\mathbf{k}$-rc, (k, Ψ)-NMC, and $(\mathbf{k}, \mathbf{p})$-core exhibited on different layers. To measure this, we introduce two metrics:

(1) **k-number**: Given a vertex subset $Q \subseteq V_l$ and a real number $p^* \in [0, 1]$, the k-number of a vertex $v \in Q$ w.r.t. G_i is defined as the P-th percentile of the coreness of vertices in $N_i(v)$ within $G_i[Q]$, where $P = (1-p^*) \times 100$. Here, the coreness of a vertex is the largest k such that there is a nonempty k-core containing the vertex in the graph. If a fraction p^* of v's cross-layer neighbors in G_i can capture enough information about v's neighborhood in G_i, a large k-number of v indicates that v's neighbors strongly engage in the community formed by neighbors of other vertices in Q, implying the close interactions between v and other vertices in Q w.r.t. G_i.

(2) **p-number**: Given a vertex subset $Q \subseteq V_l$ and an integer k^*, the p-number of a vertex $v \in Q$ w.r.t. G_i is defined as the neighbor coverage fraction of v within the k^*-core on $G_i[Q]$. Assume that the k^*-core is enough to characterize a desired cohesiveness in G_i, when the p-number of v is large, many neighbors of v cohesively interact with neighbors of other vertices in Q within G_i, which also implies v's close interactions with other vertices in Q w.r.t. G_i.

We inspect the k-number and p-number for vertices in the cohesive subgraphs characterized by these models on DBLP and Twitter. For DBLP, we set $k = 10$, $\mathbf{k} = (10, 10)$, and $\mathbf{p} = (0.7)$; and for Twitter, we use $k = 5$, $\mathbf{k} = (5, 5, 5)$, and $\mathbf{p} = (0.5, 0.5)$. When computing the k-number (resp. p-number) of vertices w.r.t G_i, we set $p^* = \mathbf{p}[i]$ (resp. $k^* = \mathbf{k}[i]$).

The distributions of the k-numbers are given in Figure 8. We can observe that the k-core contains massive vertices with small k-numbers, especially on DBLP and Twitter ($i = 1$), where vertices with k-numbers smaller than 6 and 4 account for 23% and 37% of all vertices, respectively. The $\mathbf{k}$-rc and (k, Ψ)-NMC also have vertices with small k-numbers, accounting for 21% and 7% on DBLP and Twitter ($i = 1$), respectively. These vertices show weak interactions with other vertices w.r.t. G_i, which may be broken by only removing several edges in G_i. However, all vertices in the $(\mathbf{k}, \mathbf{p})$-core have relatively large k-numbers (no less than 10 for DBLP and 5 for Twitter). Note that the other three models may contain a larger proportion of vertices with large k-numbers than the $(\mathbf{k}, \mathbf{p})$-core (see Figure 8(c)). It is because the massive vertices with small k-numbers enrich the subgraph induced by their cross-layer neighbors in G_i, and thereby many vertices have the coreness values increased.

Figure 9 shows the results on p-numbers, which exhibit similar distributions to the k-numbers. As DBLP is very dense in G_0 (a term similarity layer), only a small proportion of vertices in the k-core, $\mathbf{k}$-rc, and (k, Ψ)-NMC have small p-numbers (about 3% vertices with p-numbers less than 0.5). The k-core on Twitter has 28% and 7% vertices with p-numbers equal to 0 for $i = 1$ and 2, respectively.

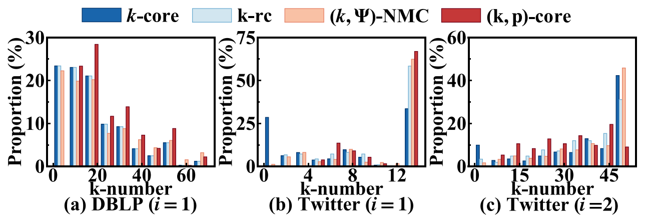


Figure 8: Distributions of the k-numbers of vertices in the k-core, k-rc, (k, Ψ)-NMC, and $(\mathbf{k}, \mathbf{p})$-core on DBLP and Twitter.

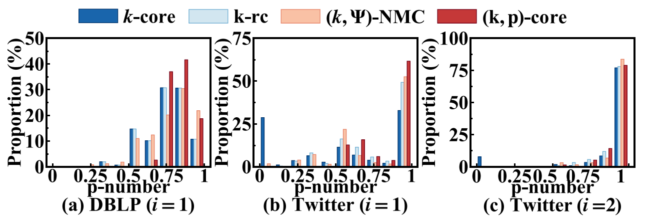
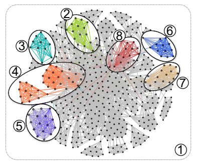


Figure 9: Distributions of the p-numbers of vertices in the k-core, k-rc, (k, Ψ)-NMC, and $(\mathbf{k}, \mathbf{p})$-core on DBLP and Twitter.



No.	Size	Top-5 Terms
①	566	**data, system, database, query, databases**
②	11	demonstration, repository, xml, active, views
③	11	application, design, technology, oracle, replication
④	26	manager, performance, storage, database, memory
⑤	11	system, incomplete, advanced, inconsistent, infomix
⑥	11	hosting, sql, server, runtime, microsoft
⑦	11	distributed, continuous, server, media, database
⑧	12	distributed, stream, engine, operation, borealis

Figure 10: Comparison between the k-core and $(\mathbf{k}, \mathbf{p})$-core on DBLP, where $k = 10$, $\mathbf{k} = (10, 10)$, and $\mathbf{p} = (0.757)$.

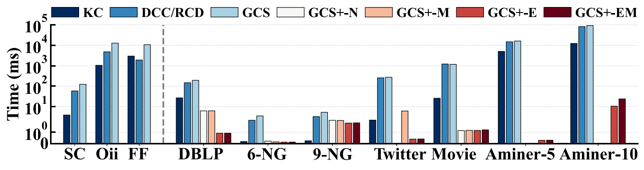


Figure 11: Runtime of cohesive subgraph search algorithms.

These vertices have no interactions with other vertices w.r.t. G_i, making the k-core show very sparse connections in terms of G_i. It is consistent with the significant $(\mathbf{k}, \mathbf{p})$-core size drop we observed in Section 8.2.1 when increasing p_i from 0 to 0.1. The $\mathbf{k}$-rc and (k, Ψ)-NMC have a smaller proportion of vertices with small p-numbers than the k-core, while each vertex in the $(\mathbf{k}, \mathbf{p})$-core has a large p-number (no less than 0.7 for DBLP and 0.5 for Twitter).

8.2.3 *Case Study on DBLP.*

We apply the gCore model to conduct collaboration analysis on the DBLP dataset. We set $\mathbf{k} = (10, 10)$ and $\mathbf{p} = (p)$, where p is the largest real number such that the $(\mathbf{k}, \mathbf{p})$-core is nonempty. In this case, $p = 0.757$. Figure 10 visualizes the largest connected component (CC) of the 10-core (No. ①) and all CCs of the $(\mathbf{k}, \mathbf{p})$-core within it. These CCs correspond to groups of cohesively collaborated authors. For each group, we compute its size and the top-5 frequently connected terms to represent the research interests of the authors, which are also presented in the figure.

We can observe that the CC of the 10-core characterizes an extremely large group of authors with research interests in data management and database systems, which are both broad fields. In contrast, each 7 small CC of the $(\mathbf{k}, \mathbf{p})$-core within it captures a group of authors collaborating on a specific research field, e.g., the one labeled No. ④ corresponds to an author group working in the field of database storage management. These focused groups are ideal for inviting to present tutorials on certain topics.

8.2.4 *Case Study on Last.Fm.*

We apply the gCore model to discover users sharing friendships and similar musical tastes on Last.Fm, an online music sharing platform (https://www.last.fm/), and show how the result guides artist recommendation in Appendix K of [23].

8.3 Efficiency Evaluation

8.3.1 *Cohesive Subgraph Search.*

We compare the efficiency of our gCore search algorithm with other core-based cohesive subgraph search algorithms on both pillar multi-layer graphs and GMGs. On pillar multi-layer graphs, KC, DCC, and GCS are compared; and on GMGs, KC, RCD, GCS, and GCS+ are tested. The total runtime of 100 queries is reported. Queries are generated by randomly sampling 100 $(\mathbf{k}, \mathbf{p})$ pairs, and for each pair, we apply $\mathbf{k}[l]$ to KC, $\mathbf{k}$ to DCC and RCD, and $(\mathbf{k}, \mathbf{p})$ to GCS and GCS+. To avoid excessive empty results, we restrict $\mathbf{k}[i] \leq \kappa(G_i)/4$ for $1 \leq i \leq l$. Besides, we distinguish GCS+ using different KP-trees. GCS+-N, GCS+-M, GCS+-E, and GCS+-EM represent GCS+ searching the KP-tree generated by TN, TM, TE, and TEM, respectively. Figure 11 reports the results. Empty bars mean that the KP-tree used by GCS+ cannot be generated within 12 hours or due to exceeded memory.

On pillar multi-layer graphs, KC usually uses the least time, which is expected as it only considers one layer. DCC runs faster than GCS because it can directly access the cross-layer neighbor of a vertex by its ID. However, GCS, designed for GMGs, has to scan the adjacent list to obtain all cross-layer neighbors of a vertex.

On GMGs, KC also runs the fastest among all algorithms without using indexes. GCS takes a slightly longer runtime than RCD as it has to check whether each vertex meets the constraints imposed on the neighbor coverage fractions. With a pre-computed KP-tree, GCS+ significantly outperforms GCS by $1-4$ orders of magnitude on the execution time on most datasets, except for 9-NG where GCS+ achieves a speedup of 2.4 to 2.9. We next inspect the efficiency of GCS+. GCS+-E and GCS+-EM generally run faster than GCS+-N and GCS+-M. This is because subtree elimination reduces the height of P+-trees, leading to fewer traversed tree nodes during gCore retrieval. Moreover, GCS+-EM is usually marginally slower than GCS+-E. In our implementation, nodes of P+-trees/P+-DAGs are stored in linked blocks in the DFS order, with each node followed by its rightmost child. GCS+-E benefits from better cache locality during traversal on P+-trees, while GCS+-EM incurs more cache misses as subtree merge disrupts the sequential storage of the nodes.

8.3.2 *KP-tree Construction.*

We evaluate our gCore decomposition (indexing) algorithm and P-tree compaction techniques. Specifically, we compare TN, TM, TE, and TEM in terms of the runtime as

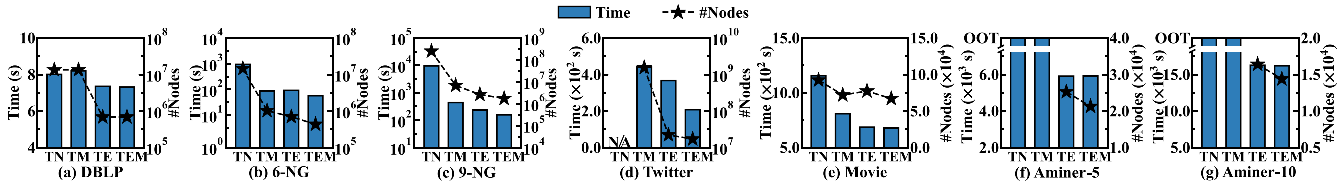


Figure 12: Construction time and scale of the KP-tree.

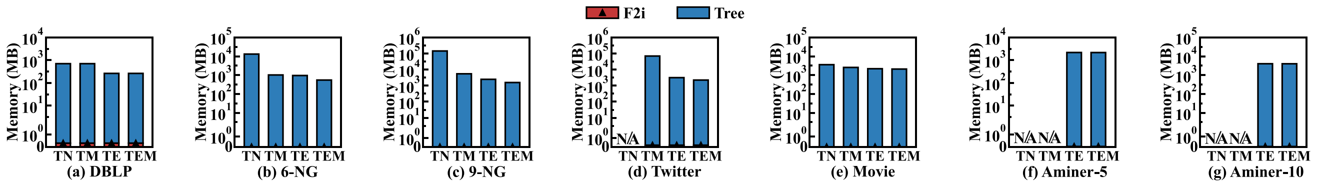


Figure 13: Memory consumption of the KP-tree.

well as the scale (number of nodes) and storage overhead of the output KP-tree index. Since the Movie, Aminer-5, and Aminer-10 datasets have about 10^5, 10^9, and 10^{11} distinct values of $\mathbf{k}$, each of which corresponds to a P-tree, constructing a complete KP-tree is infeasible under limited time and memory constraints. Moreover, not all values of $\mathbf{k}$ are of users' interests. Therefore, we randomly sample 1000 values of $\mathbf{k}$ and compute corresponding P-trees for these 3 datasets. Each element $\mathbf{k}[i]$ of $\mathbf{k}$ is chosen in $[0, \kappa(G_i)/4]$. Total runtime, scale, and storage overhead are reported.

Runtime and Scale. Figure 12 reports the runtime and the scale of the generated KP-trees. We set a timeout (OOT) of 12 hours. Execution of TN on Twitter is aborted due to exceeded memory. We have the following observations. 1) TE outperforms TN in both runtime and scale of the output, confirming the promising compaction ability of subtree elimination. 2) TM runs significantly faster than TN on 6-NG, 9-NG, and Movie, while falls behind on DBLP. This is because subtree merge only works for graphs with more than 2 layers. Additional redundant subtree detection operations slow down the execution on DBLP. 3) TEM consistently yields the smallest KP-tree, and in most cases, uses the shortest time.

Storage Overhead. Figure 13 reports the storage overhead of the generated KP-tree index, including the tree structure and the mapping that maps each <u>f</u>raction in F_i to its <u>i</u>ndex, denoted by "f2i". We can observe that: 1) F2i has neglected space cost, which is $2-8$ orders of magnitude smaller than the KP-tree for all tested datasets. 2) Both subtree merge and subtree elimination contribute to reducing the storage overhead. By integrating both of them into TN, the index generated by TEM achieves a $41\% - 98\%$ space reduction.

8.3.3 Scalability. We test GCS, GCS+, and TEM, the champion in solving the GCD and GCI problems, using different versions of the Aminer-10 dataset obtained by selecting variable numbers of layers and subsets of vertices. The detailed results are presented in Appendix J of the full paper [23]. Here are several key findings: 1) GCS and GCS+ exhibit linear and sub-linear scalability, respectively, in terms of the runtime, with the size of the multi-layer graph when the number of layers is fixed. 2) Introducing a new layer has both positive and negative impacts on the efficiency of GCS and GCS+. 3) The scale of the KP-tree generated by TEM grows linearly with the number of vertices on the layer of users' interest. 4) Introducing a new layer increases both the runtime of TEM and the scale of the generated KP-tree.

9 CONCLUSIONS AND FUTURE WORK

The generalized core (gCore) model is proposed on general multi-layer graphs (GMGs) by extending the k-core model designed for simple graphs. The gCore model has several elegant properties including uniqueness and containment hierarchy. Three related problems, gCore search (GCS), gCore decomposition (GCD), and gCore indexing (GCI), based on the gCore model are solved by designing efficient algorithms and data structures. The polynomial-time GCS algorithm is comparable to other algorithms for searching core-based cohesive subgraphs in multi-layer graphs in terms of execution time. The KP-tree structure represents the whole search space of the GCD problem and can serve as a compact storage and index (GCI) of all gCores. The KP-tree supports fast retrieval of any gCore Q in linear time in the size of Q and the height of the KP-tree. The compaction techniques, including node/subtree elimination, subtree transplant, and subtree merge, significantly speed up the construction of GCI and reduce the storage overhead of the KP-tree. Extensive experiments show that the vertices in gCores attain high closeness with each other, and the proposed algorithms are efficient in solving the three problems studied in the paper.

We outline two promising research directions for future work. First, developing efficient update mechanisms for the KP-tree index against frequent graph dynamics. By addressing this issue, we can further enhance the applicability of the gCore model and related algorithms in real-world scenarios. Second, exploring more extension schemes such as the one used in the (k, λ)-FirmCore. These extensions have the potential to provide further insights and solutions to the ML-CSM problem on GMGs.

ACKNOWLEDGMENTS

This work was partially supported by the National Natural Science Foundation of China (No. 62072138).

REFERENCES

[1] José Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat, and Alessandro Vespignani. 2005. k-core decomposition: A tool for the visualization of large scale networks. *arXiv preprint cs/0504107* (2005).

[2] Vladimir Batagelj and Matjaz Zaversnik. 2003. An o (m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049* (2003).

[3] Ali Behrouz, Farnoosh Hashemi, and Laks V. S. Lakshmanan. 2022. FirmTruss Community Search in Multilayer Networks. *Proc. VLDB Endow.* 16, 3 (2022), 505–518.

[4] Stefano Boccaletti, Ginestra Bianconi, Regino Criado, Charo I Del Genio, Jesendiús Gómez-Gardenes, Miguel Romance, Irene Sna-Nadal, Zhen Wang, and Massimiliano Zanin. 2014. The structure and dynamics of multilayer networks. *Physics reports* 544, 1 (2014), 1–122.

[5] Brigitte Boden, Stephan Günnemann, Holger Hoffmann, and Thomas Seidl. 2017. MiMAG: mining coherent subgraphs in multi-layer graphs with edge labels. *Knowledge and Information Systems* 50 (2017), 417–446.

[6] Lijun Chang and Lu Qin. 2019. Cohesive subgraph computation over large sparse graphs. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2068–2071.

[7] Annie P Chiang and Atul J Butte. 2009. Systematic evaluation of drug–disease relationships to identify leads for novel drug uses. *Clinical Pharmacology & Therapeutics* 86, 5 (2009), 507–510.

[8] Mark E Dickison, Matteo Magnani, and Luca Rossi. 2016. *Multilayer social networks*. Cambridge University Press.

[9] Yixiang Fang, Kai Wang, Xuemin Lin, and Wenjie Zhang. 2022. *Cohesive Subgraph Search Over Large Heterogeneous Information Networks*. Springer.

[10] Yixiang Fang, Yixing Yang, Wenjie Zhang, Xuemin Lin, and Xin Cao. 2020. Effective and efficient community search over large heterogeneous information networks. *Proceedings of the VLDB Endowment* 13, 6 (2020), 854–867.

[11] Edoardo Galimberti, Francesco Bonchi, and Francesco Gullo. 2017. Core decomposition and densest subgraph in multilayer networks. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 1807–1816.

[12] Farnoosh Hashemi, Ali Behrouz, and Laks VS Lakshmanan. 2022. FirmCore Decomposition of Multilayer Networks. In *Proceedings of the ACM Web Conference 2022*. 1589–1600.

[13] Jiafeng Hu, Reynold Cheng, Kevin Chen-Chuan Chang, Aravind Sankar, Yixiang Fang, and Brian YH Lam. 2019. Discovering maximal motif cliques in large heterogeneous information networks. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 746–757.

[14] Hongxuan Huang, Qingyuan Linghu, Fan Zhang, Dian Ouyang, and Shiyu Yang. 2021. Truss Decomposition on Multilayer Graphs. In *2021 IEEE International Conference on Big Data (Big Data)*. IEEE, 5912–5915.

[15] Xun Jian, Yue Wang, and Lei Chen. 2020. Effective and efficient relational community detection and search in large dynamic heterogeneous information networks. *Proceedings of the VLDB Endowment* 13, 10 (2020), 1723–1736.

[16] Daxin Jiang and Jian Pei. 2009. Mining frequent cross-graph quasi-cliques. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 2, 4 (2009), 1–42.

[17] Yangqin Jiang, Yixiang Fang, Chenhao Ma, Xin Cao, and Chunshan Li. 2022. Effective community search over large star-schema heterogeneous information networks. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2307–2320.

[18] Jungeun Kim and Jae-Gil Lee. 2015. Community detection in multi-layer graphs: A survey. *ACM SIGMOD Record* 44, 3 (2015), 37–48.

[19] Mikko Kivelä, Alex Arenas, Marc Barthelemy, James P Gleeson, Yamir Moreno, and Mason A Porter. 2014. Multilayer networks. *Journal of complex networks* 2, 3 (2014), 203–271.

[20] Victor E Lee, Ning Ruan, Ruoming Jin, and Charu C Aggarwal. 2010. A Survey of Algorithms for Dense Subgraph Discovery. *Managing and mining graph data* 40 (2010), 303–336.

[21] Yong Li and Pankaj Agarwal. 2009. A pathway-based view of human diseases and disease relationships. *PloS one* 4, 2 (2009), e4346.

[22] Boge Liu, Fan Zhang, Chen Zhang, Wenjie Zhang, and Xuemin Lin. 2019. Corecube: Core decomposition in multilayer graphs. In *Web Information Systems Engineering–WISE 2019: 20th International Conference, Hong Kong, China, January 19–22, 2020, Proceedings 20*. Springer, 694–710.

[23] Dandan Liu and Zhaonian Zou. 2023. *gCore: Exploring Cross-layer Cohesiveness in Multi-layer Graphs (technique report)*. https://github.com/SSSuperDan/gCore/blob/master/full_version.pdf

[24] Matteo Magnani and Luca Rossi. 2011. The ml-model for multi-layer social networks. In *2011 International conference on advances in social networks analysis and mining*. IEEE, 5–12.

[25] Hideo Matsuda, Tatsuya Ishihara, and Akihiro Hashimoto. 1999. Classifying molecular sequences using a linkage graph with their pairwise similarities. *Theoretical Computer Science* 210, 2 (1999), 305–325.

[26] Jingchao Ni, Shiyu Chang, Xiao Liu, Wei Cheng, Haifeng Chen, Dongkuan Xu, and Xiang Zhang. 2018. Co-regularized deep multi-network embedding. In *Proceedings of the 2018 world wide web conference*. 469–478.

[27] Elisa Omodei, Manlio De Domenico, and Alex Arenas. 2015. Characterizing interactions in online social networks during exceptional events. *Frontiers in Physics* 3 (2015), 59.

[28] Jian Pei, Daxin Jiang, and Aidong Zhang. 2005. On mining cross-graph quasi-cliques. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. 228–238.

[29] Stephen B Seidman. 1983. Network structure and minimum degree. *Social networks* 5, 3 (1983), 269–287.

[30] Yizhou Sun, Jiawei Han, Jing Gao, and Yintao Yu. 2009. itopicmodel: Information network-integrated topic modeling. In *2009 Ninth IEEE international conference on data mining*. IEEE, 493–502.

[31] Silpa Suthram, Joel T Dudley, Annie P Chiang, Rong Chen, Trevor J Hastie, and Atul J Butte. 2010. Network-based elucidation of human disease similarities reveals common functional modules enriched for pluripotent drug targets. *PLoS computational biology* 6, 2 (2010), e1000662.

[32] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 990–998.

[33] Jianyong Wang, Zhiping Zeng, and Lizhu Zhou. 2006. Clan: An algorithm for mining closed cliques from large dense graph databases. In *22nd International Conference on Data Engineering (ICDE'06)*. IEEE, 73–73.

[34] Yixing Yang, Yixiang Fang, Xuemin Lin, and Wenjie Zhang. 2020. Effective and efficient truss computation over large heterogeneous information networks. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 901–912.

[35] Zhiping Zeng, Jianyong Wang, Lizhu Zhou, and George Karypis. 2006. Coherent closed quasi-clique discovery from large dense graph databases. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. 797–802.

[36] Chen Zhang, Fan Zhang, Wenjie Zhang, Boge Liu, Ying Zhang, Lu Qin, and Xuemin Lin. 2020. Exploring finer granularity within the cores: Efficient (k, p)-core computation. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 181–192.

[37] Rong Zhu, Zhaonian Zou, and Jianzhong Li. 2018. Diversified coherent core search on multi-layer graphs. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 701–712.