Synergetic Community Search over Large Multilayer Graphs

Chengyang Luo
Zhejiang University
Hangzhou, China
luocy1017@zju.edu.cn

Qing Liu
Zhejiang University
Hangzhou, China
qingliucs@zju.edu.cn

Yunjun Gao
Zhejiang University
Hangzhou, China
gaoyj@zju.edu.cn

Jianliang Xu
Hong Kong Baptist University
Hong Kong, China
xujl@comp.hkbu.edu.hk

ABSTRACT

Community search is a fundamental problem in graph analysis and has attracted much attention for its ability to discover personalized communities. In this paper, we focus on community search over multilayer graphs. We design a novel cohesive subgraph model called *synergetic core* for multilayer graphs, which requires both *local* and *global* cohesiveness. Specifically, the synergetic core mandates that the vertices within the subgraph are not only densely connected on some individual layers but also form more cohesive connections on the projected graph that considers all layers. The local and global cohesiveness collectively ensure the superiority of the synergetic core. Based on this new model, we formulate the problem of *synergetic community search.* To efficiently retrieve the community, we propose two algorithms. The first is a progressive search algorithm, which enumerates potential layer combinations to compute the synergetic core. The second is a *trie-based search algorithm*, leveraging our novel index called *dominant layers-based trie* (DLT). DLT compactly stores synergetic cores within the trie structure. By traversing the DLT, we can efficiently identify the synergetic core. We conduct extensive experiments on ten real-world datasets. Experimental results demonstrate that (1) the synergetic core can find communities with the best quality among the state-of-the-art models, and (2) our proposed algorithms are up to five orders of magnitude faster than the basic method.

PVLDB Reference Format:

Chengyang Luo, Qing Liu, Yunjun Gao, and Jianliang Xu. Synergetic Community Search over Large Multilayer Graphs. PVLDB, 18(5): 1412 - 1424, 2025.

doi:10.14778/3718057.3718069

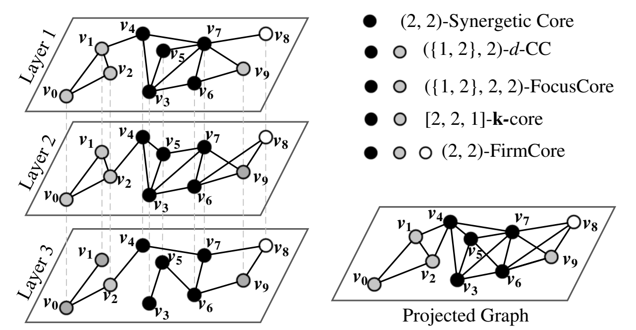
PVLDB Artifact Availability:

The source code, data, and/or other artifacts have been made available at https://github.com/ZJU-DAILY/SynCore.

1 INTRODUCTION

Real-world graphs, such as social networks [2], financial networks [9], and biological networks [21], often encompass multiple types of relationships between entities. Multilayer graphs, denoted $G = (V, E, L)$, effectively model these relationships [13]. In G, the vertex set V is shared across all layers in L, with edges on different layers representing distinct relationships. Figure 1 shows a multilayer graph G with three layers.

Proceedings of the VLDB Endowment, Vol. 18, No. 5 ISSN 2150-8097.
doi:10.14778/3718057.3718069



(a) A multilayer graph G (b) The projected graph G_P of G

Figure 1: An example of the multilayer graph

Community search, an important tool for graph analysis, aims to find a cohesive subgraph that includes a specified set of query vertices [16, 25, 33, 42]. Prior research has extensively explored community search across various graph types, such as bipartite graphs [49], directed graphs [17, 38], temporal graphs [32, 35], and heterogeneous information networks [18, 27]. In this paper, we focus on community search over multilayer graphs [4].

The cohesiveness is a crucial metric for community search [16, 25, 45]. To identify cohesively connected communities, numerous cohesive subgraph models have been proposed, including k-core [43], k-truss [26], and k-clique [10]. The k-core model, in particular, has attracted significant attention for its computational efficiency and effectiveness in identifying densely connected communities. Several core-based cohesive subgraph models tailored for multilayer graphs have also been proposed, such as **k**-core [19], d-CC [36, 53], FirmCore [20], and FocusCore [50]. Specifically, a **k**-core with $\mathbf{k} = [k_1, k_2, ..., k_{|L|}]$ requires that each vertex in the **k**-core has at least k_l neighbors on layer $l \in L$. A d-CC, given an integer d and a layer set $L' \subseteq L$, is a subgraph where every vertex connects to at least d neighbors on each layer $l \in L'$. For the FirmCore, given parameters k and λ, every vertex is connected to no less than k neighbors across at least λ layers, with the specific layers varying per vertex. The FocusCore integrates elements of d-CC and FirmCore, taking three parameters: $(\mathcal{F}, k, \lambda)$. In a $(\mathcal{F}, k, \lambda)$-FocusCore, each vertex has (1) at least k neighbors on each layer in $\mathcal{F}$, and (2) at least k neighbors across a minimum of λ layers.

However, existing core-based models for multilayer graphs have notable limitations. First, models like **k**-core, d-CC, and FocusCore require users to set multiple parameters, such as specifying k_l for each layer $l \in L$ in **k**-core or selecting layers that meet degree constraints in d-CC and FocusCore. This can be challenging for users unfamiliar with the graph's structure. Second, while FirmCore simplifies parameter settings, it often results in less cohesive communities by not enforcing a k-core structure within individual

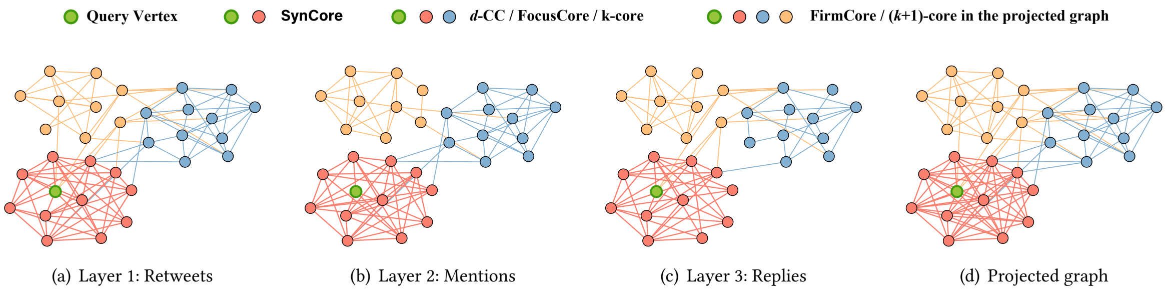


Figure 2: Case study on Twitter

layers. Last but not least, all existing models emphasize local cohesiveness within layers but overlook the crucial concept of *global cohesiveness* in multilayer graphs. Specifically, a set of vertices may form dense subgraphs on certain layers. However, when all layers are considered together, these vertices might constitute an even denser subgraph, potentially revealing a more cohesive community.

To address these limitations, we introduce the *synergetic core* (SynCore), a cohesive subgraph model that ensures density within individual layers and enhances cohesion across all layers. To capture global cohesiveness, we use the *projected graph*, a single-layer representation of a multilayer graph G. In the projected graph G_P, an edge exists between vertices u and v if they are connected in at least one layer of G. Figure 1(b) illustrates the projected graph of G from Figure 1(a). Based on the projected graph, the SynCore is defined as follows. Given a multilayer graph G, an integer k, and a support threshold $s \in [1, |L|]$, the (k, s)-SynCore is defined as the maximum subgraph $H \subseteq G$ that satisfies: (i) there are at least s layers in which H is a k-core. (ii) in the projected graph of G, H is a k'-core with $k' > k$. Notably, if H is a k-core in some layers, it must also be a k-core in the projected graph. Thus, for condition (ii), we require $k' > k$. We demonstrate the advantage of SynCore through a case study on a real-world multilayer graph, Twitter [52], as shown in Figure 2. Twitter is a multilayer social network extracted from data during the Cannes Film Festival. The graph contains three layers, each representing a different type of interaction: retweets, mentions, and replies. The green vertex denotes the query vertex. Communities returned by different models include vertices of varying colors. The community identified by SynCore, composed of red and green vertices, represents a group of users who frequently interact and likely share similar film preferences with the query vertex. SynCore offers two key advantages. First, its communities are more cohesive than those from other models. Second, SynCore effectively excludes irrelevant vertices. For instance, blue and yellow vertices, unrelated to the query vertex, appear in communities returned by other models but are successfully pruned by SynCore. Consequently, SynCore outperforms other models.

Based on SynCore, we systematically study the problem of *synergetic community search* (SynCS) over large multilayer graphs. Specifically, given a query vertex set Q, SynCS aims to find a maximum connected subgraph which contains Q and qualifies as a SynCore. The SynCS problem has diverse applications, such as friend recommendation [7, 25], cooperative team identification [34], and fraud detection [9]. For instance, in social networks, multilayer graphs model user friendships across platforms. SynCS identifies communities where users have strong connections on specific platforms and also demonstrate closer friendships on the whole. These communities are more likely to consist of people with similar interests, improving friend recommendations. For example, transaction networks from different financial institutions can form a multilayer graph. SynCS can find hidden accounts that frequently interact with suspicious ones across multiple platforms. These accounts exhibit high activity individually but trade more overall, potentially signaling a criminal group.

A basic approach for SynCS is to enumerate all possible combinations of s layers and compute the (k, s)-SynCore for each, resulting in $\binom{|L|}{s}$ combinations. However, this process is inefficient. To improve search efficiency, we propose a progressive search algorithm. First, we introduce the Quasi-SynCore, a relaxed version of SynCore, which reduces the graph to a smaller subgraph in linear time. It significantly pruning the search space. Second, we design rules to selectively enumerate layer combinations likely to contain the final result, avoiding exhaustive enumeration. All SynCores in a multilayer graph can be precomputed, allowing more efficient community search. To further accelerate this process, we introduce the *dominant layers-based trie* (DLT). Storing all SynCores is storage-intensive due to the large number of layer combinations, so we propose the concept of *dominant layers*. It represent the maximal set of layers where a vertex v is contained in a SynCore. DLT uses these dominant layers to store only a fraction of SynCores, reducing storage significantly. It organizes these layers in a trie, with each node representing a set of dominant layers and associated vertices. This enables efficient search through DLT for SynCS.

Overall, we make the following contributions in this paper.

- We propose a novel cohesive subgraph model called synergetic core for multilayer graphs. Based on it, we formally define and study the synergetic community search problem.
- We propose a progressive search algorithm for the SynCS problem that greatly prunes the search space and improves search efficiency.
- To further enhance efficiency, we develop an efficient trie-based search algorithm for the SynCS problem by designing a novel index called DLT. Additionally, we propose efficient algorithms to construct and maintain DLT.
- We conduct extensive experimental studies on ten real-world graphs. Experimental results validate the quality of the synergetic core and demonstrate the efficiency of the proposed algorithms.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 formally defines the synergetic core and the synergetic community search problem. Section 4 presents the progressive search algorithm. Section 5 proposes Dominant Layers Trie and the DLT-based search algorithm. Experimental results are reported in Section 6. Finally, Section 7 concludes the paper.

2 RELATED WORK

We review the related work from two aspects, including *cohesive subgraph models for multilayer graph*, and *community search*.

Cohesive Subgraph Models for Multilayer Graph. Various core-based cohesive subgraph models have been proposed for multilayer graphs by extending the k-core [43]. The $\mathbf{k}$-core [19] with a vector $\mathbf{k} = [k_1, k_2, ..., k_{|L|}]$ requires the identified subgraph on each layer $l \in L$ to be a k_l-core. The d-CC [36, 53] is a d-core on each layer of a given subset $L' \subseteq L$. The FirmCore [20] ensures each vertex has at least k neighbors on at least λ layers. The FocusCore [50] is a combination of d-CC and FirmCore. Moreover, gCore [37] and CrossCore [46] are proposed for the multilayer heterogeneous information graphs by extending the $\mathbf{k}$-core and d-CC, respectively. The multilayer heterogeneous information graph is a variant of the multilayer graph where vertices differ across layers and cross-layer connections exist. Other cohesive models for multilayer graphs include the frequent cross-graph γ-quasi-clique [28], which identifies the subgraph forming a γ-quasi-clique on some layers. The FirmTruss [4] ensures each edge is part of at least $k-2$ triangles on at least λ layers. The TrussCube [23] finds subgraphs where each edge is part of at least k-2 triangles on all layers of L'.

Community Search. Sozio and Gionnis [45] first introduced the community search problem. Since then, various cohesive subgraph models have been proposed for community search over simple graphs, such as k-core [3, 11], k-truss [1, 24, 26], k-clique [10], k-plex [51], and k-ECC [5]. Community search has also been studied for more complex graphs, including directed graphs [17, 38], bipartite graphs [49], keyword-based graphs [15, 39], location-based graphs [6, 14], temporal graphs [32, 35], and heterogeneous information networks [18, 27]. Recently, several learning-based methods have been proposed for community search. QD-GNN [29], AQD-GNN [29], and ALICE [48] address the community search in a supervised manner. COCLEP [31] employs contrastive learning and tackles community search in a semi-supervised manner. TransZero [47] supports unsupervised community search, which trains a graph transformer in a self-supervised manner. However, community search over multilayer graphs has received less attention. Behrouz et al. [4] studied the FirmTruss community search problem. Differing from existing works, we propose a new core-based model for community search over multilayer graphs.

3 PRELIMINARIES

A multilayer graph is denoted by $G = (V, E, L)$, where V, L, and $E \subseteq V \times V \times L$ are the sets of vertices, layers, and edges, respectively. We denote the set of edges on layer l as E_l. For a vertex $v \in V$, if $\exists l \in L, (u, v, l) \in E$, we call u a neighbor of v. Let $N_l(v)$ represent the neighbors of v on layer l. The degree of vertex v on layer l in G is denoted by $deg_l^G(v) = |N_l(v)|$. Moreover, for a subset of vertices $H \subseteq V$, $G_l[H]$ denotes the subgraph of G induced by H on layer l.

Definition 3.1. (Projected Graph). Given a multilayer graph $G = (V, E, L)$, the projected graph of G is denoted as $G_P = (V_P, E_P)$, where $V_P = V$, $E_P = \{(u, v) \mid \exists l \in L, (u, v, l) \in E\}$.

According to Definition 3.1, the projected graph G_P of the multilayer graph G is a single-layer graph that shares the same set of vertices as the multilayer graph. The edges in the projected graph are the union of edges from all layers of the multilayer graph. For a vertex $v \in V_P$, $N_P(v)$ represents the neighbors of v in the projected graph. For a set of vertices $H \subseteq V$, $G_P[H]$ denotes the subgraph of G_P induced by H, and $deg_P^H(v)$ denotes the degree of vertex v in this subgraph. For the sake of simplicity, when the context is clear, we will denote $deg_l^H(v)$ and $deg_P^H(v)$ as $deg_l(v)$ and $deg_P(v)$, respectively. Based on the multilayer graph and projected graph, we formally define the synergetic core as follows.

Definition 3.2. (Synergetic Core (SynCore)). Given a multilayer graph $G = (V, E, L)$, an integer $k \geq 0$, a support threshold $1 \leq s \leq |L|$, the SynCore of G, denoted by SC_s^k, is a maximal vertex set $H \subseteq V$ satisfying:

(i) **Local cohesiveness**: there is a set of layers $L' \subseteq L$ with $|L'| \geq s$ such that $\forall v \in H, \forall l \in L'$, $deg_l^H(v) \geq k$;

(ii) **Global cohesiveness**: in the projected graph G_P of G, $\forall v \in H$, $deg_P^H(v) > k$.

We highlight two points for the SynCore. (1) Local cohesiveness only ensures each vertex v has at least k neighbors in s layers, but it does not guarantee $k+1$ neighbors. Global cohesiveness helps to prune the vertices with exactly k neighbors in the projected graph. (2) We use the same k for both local and global cohesiveness to reduce the number of parameters.

In the following, if a SynCore forms a k-core on each layer in $L' \subseteq L$, it is denoted by $SC_{L'}^k$.

Differences from Existing Models. First, the most significant difference between SynCore and other models is that SynCore considers both the local (i.e., on individual layers) and global (i.e., in the projected graph) cohesivenesses of multilayer graphs. Second, SynCore only requires two parameters, namely k and s. In contrast, the $\mathbf{k}$-core model necessitates $|L|$ different parameters, one for each layer; the d-CC and FocusCore require a manual specification of the layers that satisfy the degree requirements. Fewer parameters can relieve users from the burdensome task of setting parameters. Third, unlike FirmCore, SynCore ensures the k-core structure.

Based on SynCore, we formally define the studied problem.

Problem 1. **(Synergetic Community Search).** Given a multilayer graph $G = (V, E, L)$, an integer $k \geq 0$, a support threshold $1 \leq s \leq |L|$, and a set of query vertices $Q \subseteq V$, the synergetic community search (SynCS) is to find a vertex set $H \subseteq V$ satisfying:

(i) **Containment**: $Q \subseteq H$;
(ii) **Cohesiveness**: H is a SynCore;
(iii) **Connectivity**: H is connected in the projected graph G_P;
(iv) **Maximum**: H is maximum.

Example 1. In Figure 1, suppose we set $k = 2$, $s = 2$, and $Q = \{v_5\}$. The black vertices, i.e., $\{v_3, v_4, v_5, v_6, v_7\}$, represent the result of SynCS. These vertices constitute a 2-core on both layer 1 and layer 2, and a 3-core in the projected graph.

A basic method to handle SynCS is to enumerate all possible combinations of s layers. For each layer combination L', we can employ the peeling method to find SynCore containing the query vertex set, i.e., to iteratively delete the vertices that do not satisfy the degree constraints. Finally, the maximum SynCore is returned. However, this basic method suffers from performance issues. Specifically, for each layer combination L', computing the SynCore requires $O(|E|+|V|)$ time. Since there are $\binom{L}{s}$ possible layer combinations, the overall time complexity amounts to $O(\binom{L}{s}\cdot(|E|+|V|))$. Clearly, the basic method is inefficient when $\binom{L}{s}$ is large. To address this, we propose more efficient algorithms in Sections 4 and 5.

4 PROGRESSIVE SEARCH ALGORITHM

The basic method should enumerate all layer combinations. Actually, some layer combinations do not contain the SynCore, and thus enumerating them is unnecessary. If we enumerate as few layer combinations as possible, the algorithm's performance will improve. Motivated by it, in this section, we propose a progressive search algorithm (PSA). The basic idea of PSA is to first identify candidate vertices that may contain the SynCore. Then, PSA iteratively enumerates the potential layer combinations for the candidate vertices that may contain the SynCore, and prunes the candidate vertices until the maximum SynCore is found. There are two issues that PSA needs to address. (1) What vertices constitute the candidate vertices? (2) How to enumerate the potential layer combinations for the candidate vertices? In the following, we address these two issues in detail.

4.1 Candidate Vertices Identification

It is costly to compute the exact SynCore for a graph. An alternative is to find candidate vertices and then refine them to the final results. To this end, we introduce the concept of Quasi-SynCore.

Definition 4.1. (Quasi-SynCore). Given a multilayer graph $G = (V, E, L)$, an integer k, and a support threshold $1 \leq s \leq |L|$, the Quasi-SynCore of G on layer set L, denoted by QSC_s^k, is a maximal vertex set $H \subseteq V$ such that for each vertex $v \in H$:

(i) there are at least s layers $L' \subseteq L$ satisfying $deg_l^H(v) \geq k$ on each layer $l \in L'$;

(ii) in the projected graph G_P of G, $deg_P^H(v) \geq k+1$.

The Quasi-SynCore does not require specific layers to meet the k-core structure. Instead, each vertex may satisfy the degree constraints on different s layers. For example, in Figure 1, given $k=2$ and $s=2$, the Quasi-SynCore is $\{v_3, v_4, v_5, v_6, v_7, v_8, v_9\}$. The Quasi-SynCore and SynCore have the following relationship.

Lemma 4.1. *Given integers k and $s \in [1, |L|]$, and layers $L' \subseteq L$ with $|L'| = s$. Then, $SC_{L'}^k \subseteq QSC_s^k$.*

According to Lemma 4.1, QSC_s^k is a superset of $SC_{L'}^k$. Next, we introduce how to compute the Quasi-SynCore efficiently. First, we introduce the concepts of degree vector, *Top-s* degree, and synergetic degree. Specifically, the degree vector of a vertex v on a set of layers L', denoted by $\mathbf{deg}_{L'}(v) = [deg_{l_1}(v), deg_{l_2}(v), \ldots, deg_{l_{|L'|}}(v)]$, consists of the degrees of v on all layers in L'. The *Top-s* degree of $\mathbf{deg}_{L'}(v)$, denoted by $Top\text{-}s(v)$, is the s-th largest degree in the degree vector $\mathbf{deg}_{L'}(v)$. The synergetic degree of v, denoted by $syn\text{-}deg(v)$, is the minimum of the *Top-s* degree and the degree in the projected graph minus 1, i.e., $syn\text{-}deg(v) = \min(Top\text{-}s(v), deg_P(v) - 1)$.

Algorithm 1: Quasi-SynCore Computation

```
   Input: A multilayer graph G(V, E, L), projected graph G_P,
          integers k and s
   Output: the Quasi-SynCore
1  foreach v ∈ V do
2      compute deg_P(v) − 1;
3      Top-s(v) ← the s-th largest degree among all layers in L;
4      syn-deg(u) ← min(Top-s(v), deg_P(v) − 1);
5  while V ≠ ∅ do
6      v ← vertex with minimum syn-deg(u) in V;
7      if syn-deg(u) >= k then
8          break;
9      V ← V \ v; V' ← ∅; // V' stores affected vertices
10     foreach (v, u, l) ∈ E and syn-deg(u) ≥ k do
11         if deg_l(u) = syn-deg(u) then
12             V' ← V' ∪ u;
13         deg_l(u) ← deg_l(u) − 1;
14     foreach (v, u) ∈ E_P and syn-deg(u) ≥ k do
15         if deg_P(u) − 1 = syn-deg(u) then
16             V' ← V' ∪ u;
17         deg_P(u) ← deg_P(u) − 1;
18     foreach u ∈ V' do
19         recompute Top-s(u);
20         update syn-deg(u);
21 return V;
```

Lemma 4.2. *Given two integers k and s, for a vertex $v \in V$, if $syn\text{-}deg(v) < k$, then $v \notin QSC_s^k$.*

According to Lemma 4.2, if the synergetic degree of a vertex v is less than k, v is not included in the Quasi-SynCore. Thus, v can be safely deleted, which leads to an update of synergetic degrees for v's neighbors. However, recomputing the synergetic degrees for all v's neighbors is costly due to the *Top-s* degrees calculation. To tackle this issue, we propose the following lemma to identify v's neighbors whose synergetic degrees do not need updating.

Lemma 4.3. *Assume a vertex $v \in V$ is deleted from a multilayer graph G, and u is a neighbor of v. If $syn\text{-}deg(u)$ satisfies (i) $syn\text{-}deg(u) \neq deg_P(u)-1$, and (ii) $\forall l \in L$ with $(v, u, l) \in E$, $syn\text{-}deg(u) \neq deg_l(u)$, $syn\text{-}deg(u)$ does not need to be updated.*

Lemma 4.3 reduces the number of neighbors whose synergetic degree needs to be recomputed. Only the neighbor u whose degree on a particular layer equals $syn\text{-}deg(u)$ or whose degree in the projected graph equals $syn\text{-}deg(u) + 1$ could be affected.

Based on the above discussion, we propose an algorithm for Quasi-SynCore computation. Algorithm 1 shows the pseudo-code. Specifically, firstly, Algorithm 1 computes the degree in the projected graph, *Top-s* degree, and synergetic degree for each vertex (lines 3-4). Then, Algorithm 1 iteratively deletes the vertex with the minimal synergetic degree (lines 5-21). After a vertex is deleted (lines 6 and 9), Algorithm 1 updates the synergetic degree of the deleted vertex's neighbors. In particular, for the v's neighbor u, Algorithm 1 updates u's degree on each layer (lines 10-13) and the projected graph (lines 14-17). Suppose $deg_l(u) = syn\text{-}deg(u)$ or $deg_P(u) - 1 = syn\text{-}deg(u)$, u will be added to the set of affected vertices (lines 11-12, 15-16). Then, the *Top-s* degree and synergetic degree of each affected vertex is updated (lines 18-20). Finally, the Quasi-SynCore is returned (line 21).

Complexity Analysis. The time complexity of of Algorithm 1 is $O(|L|\cdot(|E|+|V|))$. Algorithm 1 takes $O(|L|)$ time to get the s-largest value in a degree vector using a divide and conquer method. Hence, computing the Top-s degree for all vertices takes $O(|L|\cdot|V|)$ time. Each vertex is removed at most once, which takes $O(|V|)$ time. In addition, the update of Top-s degrees takes $O(|L|\cdot|E|)$ time. Therefore, the total time complexity is $O(|L|\cdot(|E|+|V|))$.

4.2 Potential Layer Combination Enumeration

In the previous subsection, we introduced Quasi-SynCore as the candidate for the final result. Here, we explain how to enumerate potential layer combinations for a given Quasi-SynCore. Given a multilayer graph $G=(V,E,L)$, let QSC_s^k be the Quasi-SynCore of G. The goal is to identify layer combinations that may contain the final result based on QSC_s^k.

According to Definition 4.1, the degree of a vertex $v\in QSC_s^k$ is not less than k on at least s layers of L. We call these layers the matched layers of v, denoted by $ML(v)$, i.e., $ML(v)=\{l\mid l\in L, deg_l(v)\geq k\}$. A potential layer combination L' should be a subset of L, making Quasi-SynCore on L' closer to a SynCore than QSC_s^k. Since the result must contain query vertex set Q, we enumerate the layer combination based on the matched layers of $q\in Q$ as follows.

Enumeration Rules. We consider two cases.

(1) If $|\bigcap_{q\in Q}ML(q)|<|L|$, then one layer combination $L'=\bigcap_{q\in Q}ML(q)$ should be enumerated.

(2) If $|\bigcap_{q\in Q}ML(q)|=|L|$, $\forall l\in L$, all layer combinations with $L'=L-l$ should be enumerated.

The $\bigcap_{q\in Q}ML(q)$ is the intersection of matched layers of all vertices in the query vertex set. On each layer of the intersection, the degrees of all query vertices are greater than or equal to k. This intersection represents a set of layers that may contain the final results. If $|\bigcap_{q\in Q}ML(q)|<|L|$, this intersection is an ideal layer combination that is a proper subset of L. Thus, only one layer combination L' needs to be enumerated, i.e., $L'=\bigcap_{q\in Q}ML(q)$. Otherwise, we need to enumerate more than one layer combination according to L. Each enumerated layer combination L' is obtained by removing one layer from L.

For the potential layer combination enumeration, one important issue it how to terminate the enumeration. To this end, we provide the termination rules.

Termination Rules. The layer combination enumeration can be terminated if one of the following conditions is satisfied.

(1) If $\bigcap_{v\in QSC_s^k}ML(v)=L$, enumeration according to L can be stopped.

(2) If $|L'|<s$, L' can be pruned.

(3) If L' has been enumerated before, then L' can be pruned.

(4) If the corresponding Quasi-SynCore of L' is empty or Q is not included in the Quasi-SynCore, then L' can be pruned.

We illustrate the termination rules one by one. (1) If all vertices in QSC_s^k share the same matched layers, QSC_s^k is a SynCore on L, so no new layer combination L' needs enumeration, even if $|L|>s$. (2) If the cardinality of L' is less than s, the Quasi-SynCore corresponding to L' cannot be a SynCore. Thus, the layer combination L' does not need to be enumerated. (3) Each layer combination is enumerated at most once. The enumeration process will not generate duplicate layer combinations. (4) If the Quasi-SynCore corresponding to L' is empty or does not contain Q, it cannot be the final result of SynCS. Hence, L' should not be enumerated.

Algorithm 2: Progressive Search Algorithm (PSA)

Input: a multilayer Graph $G=(V,E,L)$, $k\geq 0$, $s\in[1,|L|]$, query vertices $Q\subseteq V$
Output: maximum connected SynCore containing Q

$H\leftarrow\emptyset$; construct projected graph G_P of G;
$QSC_s^k\leftarrow$ Quasi-SynCore_Computation$((V,E,L),G_P,k,s)$;
$QSC_s^k.PL\leftarrow L$;
add QSC_s^k to T; // T stores Quasi-SynCores
while $T\neq\emptyset$ **do**
  $QSC_s^k\leftarrow$ pop the Quasi-SynCore with the largest size from T;
  if $|QSC_s^k|<|H|$ **then**
    break;
  $L_{QSC}\leftarrow QSC_s^k.PL$;
  compute the matched layers $ML(v)$ for each $v\in QSC_s^k$;
  if $\bigcap_{v\in QSC_s^k}ML(v)=L_{QSC}$ **then** // Termination Rule 1
    $H'\leftarrow$ the connected component of QSC_s^k containing Q;
    if $|H'|>H$ **then**
      $H\leftarrow H'$;
    continue;
  // Enumeration Rules
  $PL\leftarrow\emptyset$; // PL stores potential layer combinations
  if $|\bigcap_{q\in Q}ML(q)|<|L_{QSC}|$ **then**
    $PL\leftarrow\bigcap_{q\in Q}ML(q)$;
  else
    $PL\leftarrow$ all subsets of L_{QSC} with $|L_{QSC}|-1$ layers;
  foreach $L'\in PL$ **do**
    // Termination Rules 2 and 3
    if $|L'|<s$ *or* L' *has been enumerated* **then**
      continue;
    $V'\leftarrow$ all vertices of QSC_s^k;
    foreach $v\in V'$ **do**
      if $|ML(v)\cap L'|<s$ **then**
        $V'\leftarrow V'\setminus v$;
    $QSC_s'^k\leftarrow$Quasi-SynCore_Computation$((V',E,L'),G_P,k,s)$;
    $QSC_s'^k.PL\leftarrow L'$;
    // Termination Rule 4
    if $QSC_s'^k\neq\emptyset$ *and* $Q\subseteq QSC_s'^k$ **then**
      add $QSC_s'^k$ to T;
return H;

4.3 Progressive Search Algorithm

Based on the above discussions, we propose the algorithm PSA. Given a multilayer graph G, PSA firstly computes the Quasi-SynCore for G. Then, PSA enumerates the potential layer combinations according to the *Enumeration Rules.* For each layer combination L', PSA computes the Quasi-SynCore$'$ w.r.t., L'. Afterwards, PSA iteratively enumerates the potential layer combinations for the Quasi-SynCore$'$ and computes the new Quasi-SynCore$''$ until the termination rules are satisfied. Note that the Quasi-SynCore$'$ is not necessarily computed from scratch. We only need to take the Quasi-SynCore on the corresponding layer combination L' as a multilayer graph to compute the Quasi-SynCore$'$.

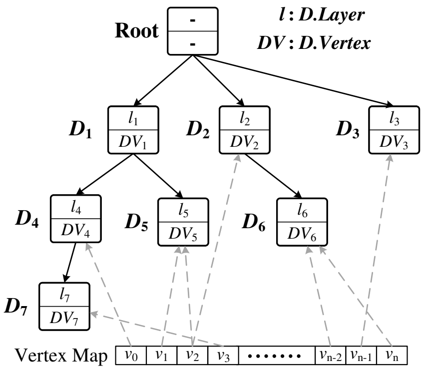


Figure 3: The structure of DLT

The pseudo-code of PSA is outlined in Algorithm 2. PSA begins with initializations (line 1), computes the Quasi-SynCore of G (line 2), sets its layer combination to L (line 3), and adds it to a heap T (line 4). It then iteratively enumerates potential layer combinations and computes corresponding Quasi-SynCores (lines 5-31). In each round, PSA examines the largest Quasi-SynCore, QSC_s^k (line 6). If its size is smaller than the current maximum SynCore, it is discarded (lines 7-8). Otherwise, PSA computes matched layers for all vertices in QSC_s^k (line 9). If all vertices share the same matched layers, QSC_s^k is a SynCore, and PSA checks its connected component containing Q as the potential result (lines 11-15). If not, PSA enumerates layer combinations based on Enumeration Rules (lines 16-20), pruning invalid ones via Termination Rules (lines 21-22). For each valid L', PSA filters invalid vertices in QSC_s^k (lines 25-27) and computes a new Quasi-SynCore, $QSC_s'^k$, using Algorithm 1 (line 28). If $QSC_s'^k$ is non-empty and includes Q, it is added to T (lines 30-31). Finally, PSA returns the result of SynCS (line 32).

Complexity Analysis. Let $\mathcal{N}$ be the total number of Quasi-SynCores computed by PSA, and $\mathcal{N}'$ be the number of Quasi-SynCores that do not generate new potential layer combinations based on termination rules. PSA primarily involves enumeration and the computation for Quasi-SynCores. For enumeration, PSA computes matched layers for each Quasi-SynCore, taking $O(|L| \cdot |V|)$ time. Thus, the total time complexity of enumeration is $O(\mathcal{N} \cdot |L| \cdot |V|)$. The computation for each Quasi-SynCore is based on an existing Quasi-SynCore, not from scratch. Therefore, the computation for all Quasi-SynCores takes $O(\mathcal{N}' \cdot |L| \cdot (|E| + |V|))$ time. Therefore, the total complexity of PSA is $O(\mathcal{N} \cdot |L| \cdot |V| + \mathcal{N}' \cdot |L| \cdot (|E| + |V|))$.

5 TRIE-BASED SEARCH ALGORITHM

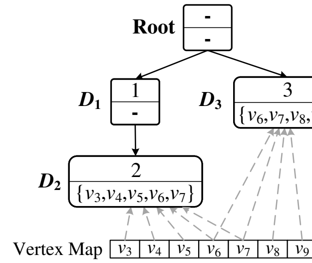
The progressive search algorithm improves the efficiency by reducing the enumeration of layer combinations. For a given multilayer graph, SynCores on different layers can be pre-computed. In light of this, we propose a novel index called Dominant Layers-based Trie (DLT) to compactly store all SynCores.

5.1 DLT Structure

Let k_{max} be the maximum k value of all non-empty SynCores in the multilayer graph. Totally, there are $k_{max} \cdot \binom{L}{s}$ SynCores. Hence, it is essential to design a compact index to reduce the index storage.

Vertex	$DL_k(v)$	Vertex	$DL_k(v)$
v_0	-	v_5	{{1,2}}
v_1	-	v_6	{{1,2},{3}}
v_2	-	v_7	{{1,2},{3}}
v_3	{{1,2}}	v_8	{{3}}
v_4	{{1,2}}	v_9	{{3}}

(a) The dominant layers



(b) The DLT

Figure 4: Examples of dominant layers and DLT

Property 1. **(Uniqueness).** Given a multilayer graph $G = (V, E, L)$, an integer k, and a layer set $L' \subseteq L$, $SC_{L'}^k$ is unique.

Property 2. **(Hierarchy).** Given a multilayer graph $G = (V, E, L)$, an integer k, and two layer combinations L' and L'' with $L'' \subseteq L' \subseteq L$. Then, $SC_{L'}^{k+1} \subseteq SC_{L'}^k$ and $SC_{L'}^k \subseteq SC_{L''}^k$.

Based on these two properties, different SynCores exhibit inclusion relationships across different layers and values of k. Motivated by this observation, we introduce the concept of dominant layers.

Definition 5.1. (Dominant Layers). Given a multilayer graph $G = (V, E, L)$, a vertex $v \in V$, and an integer $k \geq 0$. Layers $L' \subseteq L$ are dominant layers of v if the following two conditions are satisfied. (i) $v \in SC_{L'}^k$; (ii) $\nexists L'' \subseteq L$ such that $L' \subset L''$ and $v \in SC_{L''}^k$.

For a fixed k value, a vertex v may belong to the $SC_{L'}^k$ for different sets of dominant layers. We use $DL_k(v) = \{L_1, L_2, \ldots, L_n\}$ to denote all sets of dominant layers of v w.r.t., k. According to Property 2, if $v \in SC_{L'}^k$, $\forall L''' \subset L'$, $v \in SC_{L'''}^k$. Therefore, based on $DL_k(v)$, we can find all SC_L^k containing v.

In light of this, we present the Dominant Layers-based Trie (DLT) to organize all sets of dominant layers into a trie. Each DLT corresponds to a fixed k. By default, we assume that the dominant layers in a set are in ascending order of their IDs. The structure of DLT is shown in Figure 3. The root node of DLT is an empty node. The non-root node D of DLT consists of two parts: (i) $D.Layer$: a layer $l \in L$, and (ii) $D.Vertex$: a dominant vertex set DV_i of the dominant layers represented by D. Each node D_i represents a layer combination consisting of layers from the root node to D_i. We use $L(D_i)$ to denote the layer combination represented by node D_i. The dominant vertex set DV_i of D_i consists of vertices whose sets of dominant layers contains $L(D_i)$, i.e., $DV_i = \{v \in V \mid L(D_i) \in DL_k(v)\}$. Each vertex v may have more than one set of dominant layers, and it can appear in multiple DV_is. To quickly locate the DV_is containing the query vertices, a vertex map $VM_k(v)$ is employed to store which DV_is a vertex belongs to.

Example 2. We take Figure 1 as an example. Let $k = 2$. The set of dominant layers of each vertex is shown in Figure 4(a). Totally, there are two sets of dominant layers, i.e. $\{v_1, v_2\}$ and $\{v_3\}$. The corresponding DLT is depicted in Figure 4(b).

Complexity Analysis. Let θ be the average number of dominant layers sets for a vertex when k is specified. For a vertex

$v \in V$, the maximum possible k corresponding to the SynCore to which it belongs is $\sum_{l \in L} deg_l(v)$. Therefore, the total number of sets of dominant layers for all vertices is bounded by $O(\sum_{v \in V}(\theta \cdot \sum_{l \in L} deg_l(v))) = O(\theta \cdot |E|)$. The space cost of DLTs mainly consists of two parts: (i) the structure of the trie and (ii) the dominant vertex set of trie nodes. For the first part, its complexity is the same as the total number of DLT nodes, which is bounded by $O(\theta \cdot |E| \cdot |L|)$. For the second part, its complexity is the total number of sets of dominant layers for all vertices, which is $O(\theta \cdot |E|)$. Thus, the space complexity of all DLTs is $O(\theta \cdot |E| \cdot |L|)$.

5.2 DLT Construction

In this subsection, we present DLT construction method. The basic idea of DLT construction is to firstly compute all SynCores for a given multilayer graph. Then, we compute all sets of the dominant layers, based on which DLT is built. First, we introduce SynCore decomposition to compute all SynCores.

SynCore Decomposition.

We formally define the concepts of SynCoreness and SynCore decomposition as follows.

Definition 5.2. (SynCoreness). Given a multilayer graph $G = (V, E, L)$, a vertex $v \in V$, and a set of layers $L' \subseteq L$, the SynCoreness of v, denoted by $C_{L'}(v)$, is the largest k such that v is in $SC^k_{L'}$. Formally, $C_{L'}(v) = \max_{v \in SC^k_{L'}} k$.

Definition 5.3. (SynCore Decomposition). Given a multilayer graph $G = (V, E, L)$, and a set of layers $L' \subseteq L$, the SynCore decomposition computes the SynCoreness for $\forall v \in V$ and $\forall L' \subseteq L$.

Lemma 5.1. *Given a multilayer graph $G = (V, E, L)$, a vertex $v \in V$, and a set of layers $L' \subseteq L$, $C_{L'}(v) = k$ if and only if k is the largest value satisfying:*

(i) On each layer $l \in L'$, there are k neighbors $u \in N_l(v)$ such that $C_{L'}(u) \geq k$.

(ii) In the projected graph, there are $k+1$ neighbors $u_P \in N_P(v)$ such that $C_{L'}(u_P) \geq k$.

According to Lemma 5.1, $C_{L'}(v)$ can be computed according to the SynCoreness of v's neighbors. Specifically, $C_{L'}(v) = \max k$ $s.t. \forall l \in L', |\{u \in N_l(v) \mid C_{L'}(u) \geq k\}| \geq k$, and $|\{u_P \in N_P(v) \mid C_{L'}(u_P) \geq k\}| \geq k+1$. Nevertheless, we are unaware of the vertices' *SynCoreness*. An alternative is to set $C_{L'}(v)$ with an upper bound, denoted by $\overline{C}_{L'}(v)$. And then, we iteratively update the $\overline{C}_{L'}(v)$ until it converges to $C_{L'}(v)$. Here, we need to address the following issues. (1) How to set the upper bound $\overline{C}_{L'}(v)$ for v? (2) When to update the $\overline{C}_{L'}(v)$? (3) How to update the $\overline{C}_{L'}(v)$?

How to set the upper bound $\overline{C}_{L'}(v)$ for v? For a vertex $v \in V$, the SynCoreness satisfies $C_{L'}(v) \leq deg_P(v)$. In addition, based on the hierarchy property (i.e., Property 2), for $L'' \subset L'$, $SC^k_{L'}$ is a subset of $SC^k_{L''}$. Hence, we can conclude that $C_{L'}(v) \leq C_{L''}(v)$. Therefore, $\overline{C}_{L'}(v)$ can be set as follows. If $\exists L'' \subset L'$, $\overline{C}_{L'}(v)$ can initially be set to $\min_{L'' \subset L'} C_{L''}(v)$. Otherwise, $\overline{C}_{L'}(v) = deg_P(v)$.

When to update the $\overline{C}_{L'}(v)$? First, we give some useful definitions. For a layer $l \in L'$, we use $sup_l(v)$ to denote the the number of v's neighbors on layer l, whose upper bound of SynCoreness is no less than $\overline{C}_{L'}(v)$, i.e. $sup_l(v) = |\{u \in N_l(v) \mid \overline{C}_{L'}(u) \geq \overline{C}_{L'}(v)\}|$. Likewise, $sup_P(v)$ is defined as the number of v's neighbors in the projected graph, whose upper bound of SynCoreness is no less than $\overline{C}_{L'}(v)$, i.e. $sup_P(v) = |\{u \in N_P(v) \mid \overline{C}_{L'}(u) \geq \overline{C}_{L'}(v)\}|$. $\overline{C}_{L'}(v)$ needs to be updated when it satisfies the following lemma.

Lemma 5.2. *Given a multilayer graph $G = (V, E, L)$, a vertex $v \in V$, and layers $L' \subset L$, if $\exists l \in L'$ such that $sup_l(v) < \overline{C}_{L'}(v)$, or $sup_P(v) < \overline{C}_{L'}(v) + 1$, $\overline{C}_{L'}(v)$ needs to be updated.*

How to update the $\overline{C}_{L'}(v)$? When v satisfies Lemma 5.2, $\overline{C}_{L'}(v)$ is updated as follows. (1) For each layer $l \in L'$, we compute k_l, which is the maximum k satisfying that v has at least k neighbors $u \in N_l(v)$ such that $\overline{C}_{L'}(u) \geq k$. Formally, $k_l = \max k$ $s.t.$ $|\{u \in N_l(v) \mid \overline{C}_{L'}(u) \geq k\}| \geq k\}$. (2) In the projected graph, we compute k_P. Specifically, $k_P = \max k$ $s.t. |\{u \in N_P(v) \mid \overline{C}_{L'}(u) \geq k\}| \geq k+1$. After computing k_ls and k_P, $\overline{C}_{L'}(v)$ is updated to $\min_{l \in L'}\{k_l, k_P\}$.

In what follows, we give the pseudocode of SynCore decomposition, which is shown in Algorithm 3. The algorithm enumerates all

Algorithm 3: SynCore Decomposition

Input: multilayer graph G
Output: the SynCorenesses of all vertices
construct projected graph G_P of G;
foreach $s = 1$ *to* $|L|$ **do**
  foreach *layers* $L' \subseteq L$ *and* $|L'| = s$ **do**
    foreach $v \in V$ **do**
      if $s = 1$ **then**
        $\overline{C}_{L'}(v) \leftarrow deg_P(v)$;
      else
        $\overline{C}_{L'}(v) \leftarrow \min_{L'' \subset L'} C_{L''}(v)$;
    $C'_L \leftarrow$ Compute_SynCoreness$(G, G_P, L', \overline{C}_{L'})$
return C'_L for all $L' \subseteq L$

Procedure Compute_SynCoreness$(G, G_P, L', \overline{C}_{L'})$
foreach $v \in V$ **do**
  foreach $l \in L'$ **do**
    $sup_l(v) \leftarrow |\{u \in N_l(v) \mid \overline{C}_{L'}(u) \geq \overline{C}_{L'}(v)\}|$;
  $sup_P(v) \leftarrow |\{u \in N_P(v) \mid \overline{C}_{L'}(u) \geq \overline{C}_{L'}(v)\}|$;
while $\exists v \in V, l \in L'$ *such that* $sup_l(v) < \overline{C}_{L'}(v)$ *or* $sup_P(v) < \overline{C}_{L'}(v) + 1$ **do**
  $ub \leftarrow \overline{C}_{L'}(v)$;
  foreach $l \in L'$ **do**
    $k_l(v) \leftarrow \max k$ $s.t. |\{u \in N_l(v) \mid \overline{C}_{L'}(u) \geq k\}| \geq k$;
    $\overline{C}_{L'}(v) \leftarrow \min(k_l(v), \overline{C}_{L'}(v))$;
  $k_P(v) \leftarrow \max k$ $s.t. |\{u \in N_P(v) \mid \overline{C}_{L'}(u) \geq k\}| \geq k+1$;
  $\overline{C}_{L'}(v) \leftarrow \min(k_P(v), \overline{C}_{L'}(v))$;
  foreach $l \in L'$ **do**
    $sup_l(v) \leftarrow |\{u \in N_l(v) \mid \overline{C}_{L'}(u) \geq \overline{C}_{L'}(v)\}|$;
    foreach $u \in N_l(v)$ **do**
      if $\overline{C}_{L'}(u) \leq ub$ *and* $\overline{C}_{L'}(u) > \overline{C}_{L'}(v)$ **then**
        $sup_l(u) \leftarrow sup_l(u) - 1$;
  $sup_P(v) \leftarrow |\{u \in N_P(v) \mid \overline{C}_{L'}(u) \geq \overline{C}_{L'}(v)\}|$;
  foreach $u \in N_P(v)$ **do**
    if $\overline{C}_{L'}(u) \leq ub$ *and* $\overline{C}_{L'}(u) > \overline{C}_{L'}(v)$ **then**
      $sup_P(u) \leftarrow sup_P(u) - 1$;
return $\overline{C}_{L'}$;

layer combinations with the cardinality from 1 to $|L|$ (lines 2-3). For each enumerated layer combination L', if L' only contains one layer, the upper bound $\overline{C}_{L'}(v)$ is set to the degree in the projected graph (line 6). Otherwise, $\overline{C}_{L'}(v)$ is set to the minimum SynCoreness of v among all previously enumerated layer combinations $L'' \subset L'$ (line 8). Then, Algorithm 3 computes the SynCorenesses of vertices for L' employing the function *Compute_SynCoreness* (line 9).

The function *Compute_SynCoreness* is outlined in lines 11-31 of Algorithm 3. It firstly computes $sup_l(v)$ and $sup_P(v)$ for each vertex (lines 11-14), and then identifies the vertex v, whose $\overline{C}_{L'}(v)$ needs to be updated, according to Lemma 5.2 (line 15). Next, $\overline{C}_{L'}(v)$ is updated (lines 16-21). Subsequently, $sup_l(v)$ and $sup_P(v)$ of v, and $sup_l(u)$ and $sup_P(u)$ of v's neighbor u are updated according to the new $\overline{C}_{L'}(v)$ (lines 22-30). If there does not exist a vertex v, whose $\overline{C}_{L'}(v)$ needs to be updated, the function terminates.

Dominant layers Computation and DLTs Construction

When completing the SynCore decomposition, we should compute all sets of the dominate layers for every vertex. Specifically, consider a layer combination $L' \subseteq L$, an integer $k \geq 0$, and a vertex $v \in V$. If $C_{L'}(v) \geq k$ and there is no $L' \subset L''$ such that the SynCoreness of v on L'' is greater than or equal to k, L' is a set of dominant layers of v. Therefore, all sets of dominant layers of v for a given k are $DL_k(v) = \{L' \subseteq L \mid C_{L'}(v) \geq k \wedge \nexists L'' \subseteq L$ such that $L' \subset L''$ and $C_{L''}(v) \geq k\}$.

After computing all sets of dominant layers, the DLT for k is then constructed by accessing all sets of dominant layers of all vertices. We assume that the layers in dominant layers are in ascending order of their ID. Hence, each set of dominant layers can be treated as a string and we can build a trie for all sets of dominant layers. One point should be highlighted is that, when building trie, each node of trie is augmented with the dominant vertex set DV_i.

Based on the above discussions, the procedure for DLT Construction is outlined in Algorithm 4. The algorithm first employs the SynCore decomposition algorithm to obtain all SynCorenesses for all vertices (line 1). Once the decomposition is complete, the algorithm enumerates k from 0 to its maximum value to build the DLTs. For each k, a trie node is initialized as the root for a DLT (lines 4-5). Then, all sets of dominant layers for each vertex v are computed (line 7). For each set of dominant layers of v, its layers are traversed, building the DLT by visiting from the root node (lines 8-19). Consider a layer number l and a visited DLT node D. If there is a child node of D' with layer number l, the child node will be visited (lines 11-12). Otherwise, a new DLT node with l will be created (lines 14-15). If l is the last number of the set of dominant layers, v is added to the dominant vertex set of the node, and the node is added to the vertex map of v (lines 16-18). Finally, DLTs for all possible k are returned.

Complexity Analysis. Let θ be the average number of dominant layers for each vertex when k is specified, and U_{max} be the maximum upper bound of SynCoreness for any vertex. The time complexity of Algorithm 4 is $O(2^{|L|} \cdot (|L| \cdot |V| + U_{max} \cdot |E|) + \theta \cdot |E| \cdot |L|)$. The time cost of Algorithm 4 consists of two parts: (i) SynCore decomposition and (ii) DLT building. For the decomposition phase on layers $L' \subseteq L$, it takes $O(|L| \cdot |V|)$ time to set the upper bound of SynCoreness and $O(U_{max} \cdot |E|)$ time to perform the decomposition. With $2^{|L|} - 1$ combinations, the total time is $O(2^{|L|} \cdot (|L| \cdot |V| + U_{max} \cdot |E|)$.

Algorithm 4: DLT Construction Algorithm

Input: multilayer graph G
Output: DLT
1 All SynCorenesses ←SynCore_Decomposition(G);
2 $DLT \leftarrow \emptyset$;
3 **foreach** $k = 0$ *to* k_{max} **do**
4   initialize a root trie node D_{root};
5     $DLT[k] \leftarrow D_{root}$;
6   **foreach** $v \in V$ **do**
7         $DL_k(v) = \{L' \subseteq L \mid C_{L'}(v) \geq k \wedge \nexists L'' \subseteq L$ such that $L' \subset L''$ and $C_{L''}(v) \geq k\}$;
8   **foreach** $L' \in DL_k(v)$ **do**
9             $D \leftarrow DLT[k]$; // D is a root node
10   **foreach** $l \in L'$ **do**
11   **if** $\exists\, D' \in D.child$ *such that* $D'.Layer = l$ **then**
12                    $D \leftarrow D'$;
13   **else**
14   create a new node D'; $D'.Layer = l$;
15                    $D.child.add(D')$; $D \leftarrow D'$;
16   **if** l *is the last number of* L' **then**
17                    $D.Vertex.add(v)$;
18                    $VM_k(v).add(D)$;
19 **return** DLT;

For the DLT building phase, each set of dominant layers of vertices is accessed for a specific k, taking $O(\theta \cdot |E| \cdot |L|)$ time. Therefore, The time complexity of Algorithm 4 is $O(2^{|L|} \cdot (|L| \cdot |V| + U_{max} \cdot |E|) + \theta \cdot |E| \cdot |L|)$.

5.3 DLT-based Search Algorithm

Based on the DLT index, we develop an efficient DLT-based search algorithm (DSA) to handle SynCS. The main idea of DSA is as follows. It firstly finds all dominant layer sets $DL_k(q)$ for each query vertex $q \in Q$, and uses dominant layer sets $DL_k(q)$ to enumerate all layer combinations that contain s layers for q. If a layer combination L' is enumerated for all query vertices, Q is contained in $SC^k_{L'}$. All such layer combinations constitute candidate layer combinations. Next, for each candidate layer combination L', DSA traverses the DLT to find all vertices of $SC^k_{L'}$ via matching. Finally, $SC^k_{L'}$ with the maximum size is returned.

The pseudocode of DSA is shown in Algorithm 5. First, DSA obtains all sets of dominant layers for each query vertex q by utilizing the vertex map $VM_k(q)$ (line 5). For each set of dominant layers, different combinations of s layers are stored in $tempSetL$ (line 7). The $setL$ stores the intersection of layer combinations for s layers of all query vertices (line 8). Then, for each combination $L' \in setL$, a stack S is used to perform DFS on the DLT. DSA pushes the root node into the stack with the matched number $i = 0$ (line 10). For each element popped from the stack, i.e., DLT node D and the matched number i, DSA matches the node and pushes potential child nodes into S according to the *Matching Mechanism* (lines 14-23). When DSA gets the $SC^k_{L'}$, it searches the connected component that contains the query vertex set (line 24). The maximum connected component is the result of SynCS (line 27).

Example 3. In the DLT of Figure 4(b), given $k = 2$, $s = 2$, and $Q = \{v_5\}$, v_5 is in the dominant vertex set of node D_2, so the possible layer combination is $\{1, 2\}$. DSA starts from the root. Nodes D_1 and

Algorithm 5: DLT-based Search Algorithm (DSA)

Input: multilayer graph G, projected graph G_P, $k \geq 0$, $s \in [1, |L|]$, query vertices Q, DLTs
Output: the SynCore H

$H \leftarrow \emptyset$; $setL \leftarrow$ all combinations of s layers;
foreach $q \in Q$ **do**
  $tempSetL \leftarrow \emptyset$;
  foreach $D \in VM_k(q)$ **do**
    $L' \leftarrow$ layer combination represented by D;
    foreach $L'' \subseteq L'$ *and* $|L''| = s$ **do**
      $tempSetL.add(L'')$;
  $setL \leftarrow tempSetL \cap setL$;
foreach $L' \in setL$ **do**
  $S.push((DLT[k], 0))$ // S is a stack
  $H' \leftarrow \emptyset$;
  while $S \neq \emptyset$ **do**
    $(D, i) \leftarrow$ pop the top element from S;
    if $D.Layer = L'[i]$ **then**
      if $i < |L'| - 1$ **then**
        $i \leftarrow i + 1$;
      else
        foreach D' *in the subtree of* D **do**
          add $D'.Vertex$ to H';
        continue;
    foreach $D' \in D.Child$ **do**
      if $D'.Layer \leq L'[i]$ **then**
        $S.push((D', i))$;
  $H' \leftarrow$ connected component of H' that contains Q;
  if $|H'| \geq |H|$ **then**
    $H \leftarrow H'$;
return H;

D_2 are matched sequentially, adding $\{v_3, v_4, v_5, v_6, v_7\}$ to the result. Node D_3 cannot be matched. The final result is $\{v_3, v_4, v_5, v_6, v_7\}$.

Complexity Analysis. Let $\mathcal{H}$ be the number of vertices of the final result, and λ be the number of candidate layer combinations of the query vertex set. There are $O(\lambda \cdot |L|)$ DLT nodes to be accessed. Choosing child nodes for each DLT node takes $O(\log |L|)$ time. So traversing the DLT takes $O(\lambda \cdot |L| \cdot \log(|L|))$ time. The time cost of aggregating results is proportional to the total number of vertices in the result, i.e., $O(\mathcal{H})$. Therefore, the total time complexity of Algorithm 5 is $O(\lambda \cdot (\mathcal{H} + |L| \cdot \log(|L|)))$.

5.4 DLT Maintenance

In this subsection, we discuss DLT maintenance. After some vertices or edges are inserted or deleted from the multilayer graph, the SynCoreness of some vertices may change. Let V^* denote the vertices whose SynCoreness has changed. Let $C^*_{L'}(v)$ denote the new SynCoreness of $v \in V^*$ on layers $L' \subseteq L$.

For each affected vertex, we first compute its new sets of dominant layers when k is specified, denoted as $DL^*_k(v)$. Specifically, $DL^*_k(v) = \{L' \subseteq L \mid C^*_{L'}(v) \geq k \wedge \nexists L'' \supset L' \text{ s.t. } C^*_{L''}(v) \geq k\}$. Once we obtain the new sets of dominant layers for a vertex, the collection of newly added sets of dominant layers is $DL^+_k(v) = DL^*_k(v) \setminus DL_k(v)$, and the collection of deleted sets of dominant layers is $DL^-_k(v) = DL_k(v) \setminus DL^*_k(v)$. For each newly added set of dominant layers L^+, let $V(L^+)$ represent the vertices whose newly added dominant layers include L^+. Formally, $V(L^+) = \{v \in V \mid L^+ \in DL^+_k(v)\}$. Then, L^+ is treated as a string, and inserted into the DLT. If a node corresponding to the prefix of L^+ does not exist, a new DLT node is created. For the DLT node whose represented prefix matches L^+, we add $V(L^+)$ to the dominant vertex set of that node. For each set of deleted dominant layers L^-, let $V(L^-)$ denote the vertices whose deleted dominant layers set contains L^-, i.e., $V(L^-) = \{v \in V \mid L^- \in DL^-_k(v)\}$. For the node whose represented prefix matches L^-, $V(L^-)$ is removed from its dominant vertex set. If a leaf node has an empty dominant vertex set, it is deleted.

Table 1: Statistics of networks ($\overline{|DL(v)|}$: the average number of dominant layer sets for a vertex; θ: the average number of dominant layer sets per vertex for a given k; K: 10^3; M: 10^6; B: 10^9)

Dataset	Abbr.	$\|V\|$	$\|E\|$	$\|L\|$	$\overline{\|DL(v)\|}$	θ
Slashdot [20]	SD	51**K**	139**K**	6	1.3	0.33
Homo [52]	HM	18**K**	153**K**	7	6.1	0.64
SacchCere [52]	SC	6.5**K**	247**K**	7	29.0	1.22
DBLP [19]	DB	512**K**	1.0**M**	10	1.4	0.44
Obama [52]	OB	2.2**M**	3.8**M**	3	0.5	0.15
YEAST [52]	YE	4.5**K**	8.5**M**	4	1635.2	0.99
Higgs [52]	HI	456**K**	13**M**	4	27.5	0.89
StackOverflow [44]	SO	2.6**M**	64.5**M**	9	8.5	0.69
Wiki [30]	WK	6.9**M**	129**M**	10	5.9	0.66
MAG [22]	MG	53**M**	1.1**B**	8	7.2	0.78

6 EXPERIMENTS

This section evaluates the performance of the index structure and algorithms we propose. The experiments use ten real-world multilayer graphs, details of which are summarized in Table 1. All algorithms were implemented in C++ and compiled by GCC 9.3.0 with -O3 optimization. All experiments were conducted on a machine running on Ubuntu server 20.04.2 LTS version with an Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz and 128GB memory. In each experiment, we randomly generate 1,000 sets of query vertices, and report the average results. The experimental results for all datasets are available in our technical report [41].

6.1 Effectiveness Evaluation

First, we evaluate the effectiveness of our proposed SynCore model.

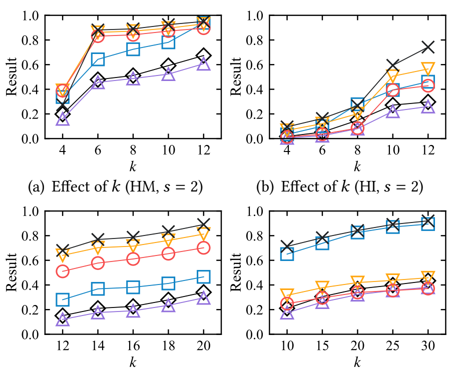
Exp-1: Community Quality Comparison of Different Models. We assess the quality of communities retrieved by different models under their optimal parameter settings.

Competitors. We compare SynCore with seven cohesive subgraph models for multilayer graphs, i.e., d-CC [36, 53], FirmCore [20], FocusCore [50], CrossCore[46], **k**-core [19], and FirmTruss [4]. We also compare the community returned by $(k+1)$-core in the projected graph.

Metrics. To evaluate the quality of the discovered communities, we employ two types of metrics: the *density* ($\frac{2\cdot|E|}{|V|\cdot(|V|-1)}$) [8] and the *global clustering coefficient* ($\frac{3\cdot|\Delta|}{|triplet|}$) [40]. Let H be the returned community; $D_l[H]$ and $GCC_l[H]$ (rep. $D_P[H]$ and $GCC_P[H]$) are the density and global clustering coefficient of H on layer l (rep.

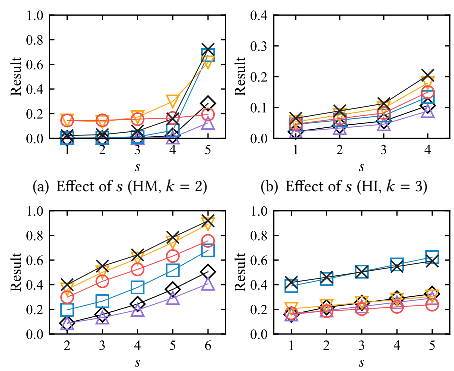
Table 2: Community quality comparison for different models under the optimal parameter settings (Metrics: ①: *D-Avg*, ②: *D-Min*, ③: *D-P*, ④: *GCC-Avg*, ⑤: *GCC-Min*, ⑥: *GCC-P*)

Datasets	HM						DB						HI						WK						MG					
Metrics	①	②	③	④	⑤	⑥	①	②	③	④	⑤	⑥	①	②	③	④	⑤	⑥	①	②	③	④	⑤	⑥	①	②	③	④	⑤	⑥
SynCore	**0.78**	**0.73**	**0.88**	**0.83**	**0.81**	**0.91**	**0.66**	**0.60**	**0.72**	**0.84**	**0.73**	0.85	**0.50**	**0.43**	0.80	**0.65**	**0.55**	**0.74**	**0.53**	**0.40**	**0.79**	**0.86**	**0.71**	**0.93**	**0.64**	**0.45**	**0.76**	**0.51**	**0.40**	**0.97**
d-CC	0.49	0.48	0.67	0.69	0.68	0.81	0.42	0.42	0.61	0.70	0.69	0.71	0.41	0.30	0.71	0.42	0.36	0.55	0.37	0.34	0.73	0.72	0.63	0.79	0.44	0.38	0.60	0.39	0.36	0.79
FirmCore	0.40	0.07	0.54	0.53	0.08	0.63	0.28	0.02	0.45	0.40	0.09	0.54	0.24	0.03	0.47	0.25	0.03	0.33	0.28	0.02	0.57	0.47	0.04	0.69	0.32	0.05	0.35	0.31	0.03	0.63
FocusCore	0.69	0.62	0.75	0.80	0.73	0.83	0.53	0.52	0.64	0.72	0.72	0.79	0.46	0.37	**0.81**	0.46	0.41	0.55	-	-	-	-	-	-	-	-	-	-	-	-
CrossCore	0.49	0.48	0.67	0.69	0.68	0.81	0.42	0.42	0.61	0.70	0.69	0.71	0.41	0.30	0.71	0.42	0.36	0.55	0.37	0.34	0.73	0.72	0.63	0.79	-	-	-	-	-	-
k-core	0.55	0.48	0.69	0.74	0.72	0.82	0.43	0.43	0.62	0.75	0.72	0.78	0.42	0.31	0.73	0.43	0.36	0.59	-	-	-	-	-	-	-	-	-	-	-	-
$(k+1)$-core	0.31	0.05	0.42	0.33	0.06	0.49	0.21	0.01	0.31	0.25	0.07	0.33	0.16	0.02	0.29	0.18	0.02	0.20	0.22	0.02	0.45	0.28	0.03	0.50	0.20	0.03	0.34	0.20	0.02	0.47
FirmTruss	0.52	0.09	0.64	0.64	0.16	0.73	0.47	0.06	0.52	0.61	0.20	**0.88**	0.41	0.06	0.70	0.36	0.14	0.66	-	-	-	-	-	-	-	-	-	-	-	-



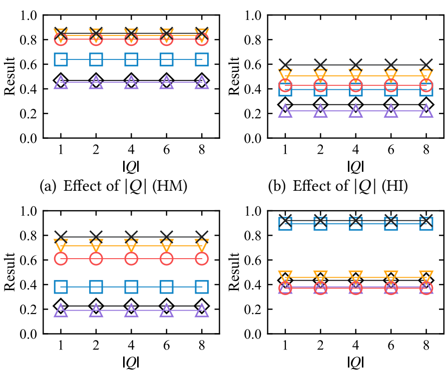
(a) Effect of k (HM, $s = 2$)
(b) Effect of k (HI, $s = 2$)
(c) Effect of k (WK, $s = 3$)
(d) Effect of k (MG, $s = 3$)

Figure 5: Effect of k on community quality



(a) Effect of s (HM, $k = 2$)
(b) Effect of s (HI, $k = 3$)
(c) Effect of s (WK, $k = 10$)
(d) Effect of s (MG, $k = 15$)

Figure 6: Effect of s on community quality



(a) Effect of $|Q|$ (HM)
(b) Effect of $|Q|$ (HI)
(c) Effect of $|Q|$ (WK)
(d) Effect of $|Q|$ (MG)

Figure 7: Effect of $|Q|$ on community quality

projected graph), respectively; and L_c is the set of layers that satisfy the degree constraints of cohesive subgraph models.

(1) The density-based metrics include *average density* (*D-Avg*), *minimum density* (*D-Min*), and *density in the projected graph* (*D-P*) of H, which are defined as follows.

$$D\text{-}Avg = \frac{\sum_{l \in L_c} D_l[H]}{|L_c|},\ D\text{-}Min = \min_{l \in L_c} D_l[H],\ D\text{-}P = D_P[H]$$

(2) The global clustering coefficient-based metrics consist of *average global clustering coefficient* (*GCC-Avg*), *minimum global clustering coefficient* (*GCC-Min*), and *global clustering coefficient in the projected graph* (*GCC-P*). Specifically,

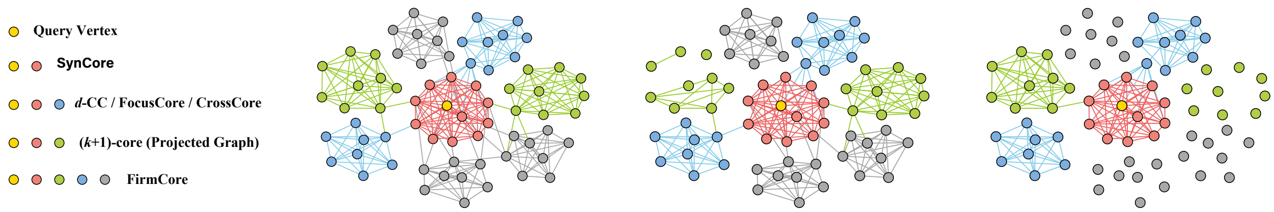
$$GCC\text{-}Avg = \frac{\sum_{l \in L_c} GCC_l[H]}{|L_c|},$$

$$GCC\text{-}Min = \min_{l \in L_c} GCC_l[H], \quad GCC\text{-}P = GCC_P[H]$$

Results. The community quality of all models under their optimal parameter settings on five datasets are shown in Table 2. Note that "-" represents that evaluating the quality of all parameter combinations for the model could not be completed within 24 hours. Overall, SynCore shows superior quality compared to other models. This is because SynCore considers both local and global cohesiveness, enabling it to identify highly cohesive communities. Other models like d-CC, CrossCore, and **k**-core only consider cohesiveness on partial layers, resulting in lower densities and clustering coefficients compared to SynCore. As for FirmCore and FirmTruss, they do not require dense structures across layers, leading to lower-quality communities. FocusCore, which simply combines d-CC and FirmCore, does not enhance community quality. The $(k+1)$-core in the projected graph performs the worst, showing that the quality of SynCore is not solely reliant on projected graph constraints. These results demonstrate the effectiveness of SynCore's design.

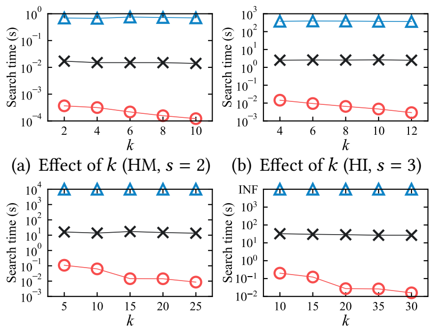
Exp-2: Effect of Parameters on Community Quality. In this set of experiments, we investigate the effects of different parameters on SynCore, including k, s, and $|Q|$. Figures 5 to 7 presents the results. **Effect of k** (Figure 5): As k increases, all metrics consistently improve, indicating that higher k values lead to more cohesive subgraphs. This is because increasing k results in more neighbors, thereby enhancing community cohesiveness. **Effect of s** (Figure 6): In general, increasing s improves the community quality. This is because larger s requires more layers to meet the degree constraint, resulting in denser community structures. **Effect of $|Q|$** (Figure 7): We can observe that when $|Q|$ varies, the qualities of communities almost keep the same. It is because the query vertex sets of different sizes belong to the same community.

Exp-3: Case Study on DBLP. In this experiment, we conduct a case study on DBLP [12]. We collected papers from VLDB, SIGMOD, ICDE, and KDD to construct a four-layer graph. Each vertex represents an author, and each layer corresponds to a conference. An edge exists between two authors within a layer if they have coauthored a paper in that conference. The core models used in this study are SynCore, d-CC, FocusCore, CrossCore, FirmCore,



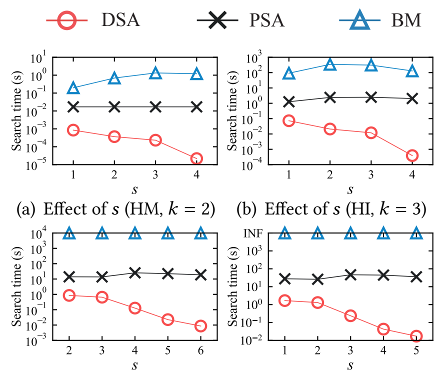
(a) Projected graph (b) Layer of SIGMOD (c) Layer of ICDE

Figure 8: Case study on DBLP



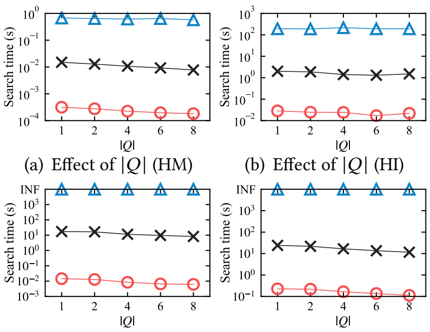
(a) Effect of k (HM, s = 2) (b) Effect of k (HI, s = 3)
(c) Effect of k (WK, s = 3) (d) Effect of k (MG, s = 3)

Figure 9: Effect of k on search time



(a) Effect of s (HM, k = 2) (b) Effect of s (HI, k = 3)
(c) Effect of s (WK, k = 10) (d) Effect of s (MG, k = 15)

Figure 10: Effect of s on search time



(a) Effect of $|Q|$ (HM) (b) Effect of $|Q|$ (HI)
(c) Effect of $|Q|$ (WK) (d) Effect of $|Q|$ (MG)

Figure 11: Effect of $|Q|$ on search time

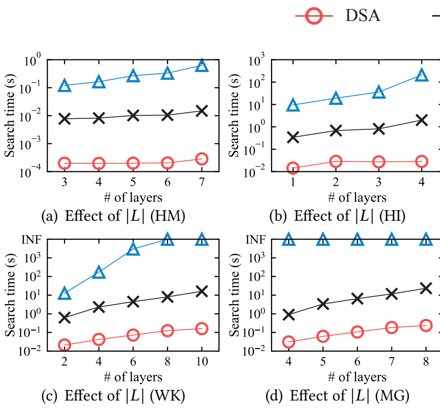
and $(k+1)$-core in the projected graph. To conduct community search, we set the query vertex Q = Samuel Madden; $k = k_{max} = 7$ to ensure the returned community is non-empty; and $s = 2$. For d-CC and FocusCore, we specify SIGMOD and ICDE as the layers of interest. For FirmCore, we set $\lambda = 2$. Figure 8 shows the connected subgraphs returned by these models. All subgraphs contain the query vertex, which is highlighted in yellow. Specificaaly, red vertices represent authors returned by SynCore; blue and red vertices represent authors identified by d-CC, FocusCore, and CrossCore; green, blue, and red vertices represent authors returned by FirmCore. The community identified by SynCore (red and yellow vertices) stands out as the most cohesive, evidently constituting a close network of academic collaborators. The collaborators exhibit strong cohesiveness not only within the SIGMOD and ICDE, but also form a tighter network across the four data-related conferences. For the communities identified by other models, some authors who do not closely collaborate with the query vertex are included in the result. For example, in d-CC / FocusCore / CrossCore (blue, red, and yellow vertices), the set of blue vertices in the bottom left has only one edge connecting to the main red cluster, which should not be included in the result.

6.2 Efficiency Evaluation

We evaluate the efficiency of our proposed algorithms, including the *DLT-based Search Algorithm* (DSA), the *Progressive Search Algorithm* (PSA), and the *Basic Method* (BM). We test parameters including k, s, $|Q|$, $|L|$, and $|V|$. Each experiment varies one parameter while keeping the others at default values. k and s are set by the datasets, $|Q|$ defaults to 1, and $|L|$ and $|V|$ use original dataset values. We report running time, using INF for any that exceed 10^4 seconds.

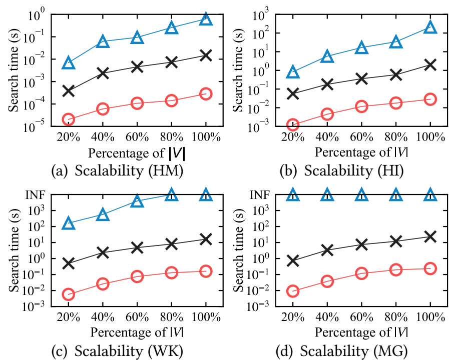
Exp-4: Effect of Parameters on Search Time. In this set of experiments, we investigate the effects of different parameters on search time, including k, s, $|Q|$, and $|L|$. Results are shown in Figures 9 to 12, respectively. Overall, DSA consistently outperforms PSA and BM by 1-5 orders of magnitude while PSA is better than BM. This is because DSA leverages the DLT index to avoid multiple graph traversals. In contrast, BM must enumerate all possible layer combinations. PSA improves upon BM by pruning invalid combinations. However, it still needs traversals the original graph to compute SynCore, which makes PSA slower than DSA. **Effect of k** (Figure 9): As k increases, DSA becomes faster because a larger k results in a smaller SynCore, reducing the number of DLT nodes it needs to traverse. PSA and BM are not significantly affected by k as k does not impact the number of layer combinations. **Effect of s** (Figure 10): Increasing s reduces DSA's search time by decreasing the size of SynCore. For BM, increasing s results in more layer combinations, leading to longer search times. PSA maintains stable performance by pruning combinations effectively. **Effect of $|Q|$** (Figure 11): As $|Q|$ increases, DSA and PSA show slight decreases in search time due to fewer layer combinations. BM's search time remains constant since it must compute SynCore for all combinations regardless of $|Q|$. **Effect of $|L|$** (Figure 12): Increasing $|L|$ deteriorates the performance of all algorithms due to an expanded search space from more layer combinations.

Exp-5: Scalability The scalability of the three algorithms is shown in Figure 13. In this experiment, we randomly sampled vertices



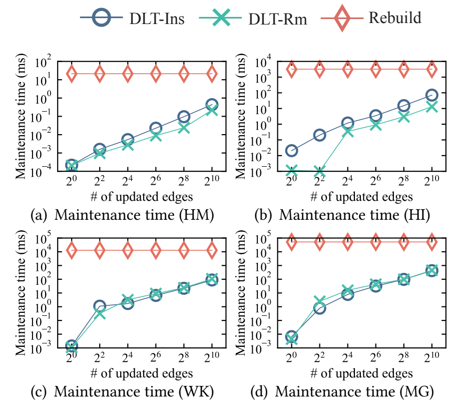
(a) Effect of $|L|$ (HM) (b) Effect of $|L|$ (HI) (c) Effect of $|L|$ (WK) (d) Effect of $|L|$ (MG)

Figure 12: Effect of $|L|$ on search time



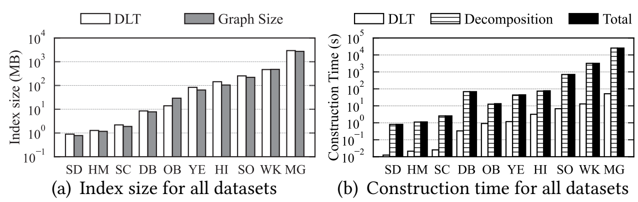
(a) Scalability (HM) (b) Scalability (HI) (c) Scalability (WK) (d) Scalability (MG)

Figure 13: Scalability



(a) Maintenance time (HM) (b) Maintenance time (HI) (c) Maintenance time (WK) (d) Maintenance time (MG)

Figure 14: DLT maintenance evaluation



(a) Index size for all datasets (b) Construction time for all datasets

Figure 15: DLT construction evaluation

from the original graph to generate graphs with varying sizes. As expected, the performance of all algorithms declines as the graph size increases, since larger graphs lead to larger communities. However, DSA and PSA consistently outperform BM by several orders of magnitude, highlighting their superior efficiency. On the MG dataset, the proposed DSA and PSA methods demonstrate high efficiency, while the baseline BM fails to finish within 10^4 seconds. The remarkable efficiency of DSA and PSA can be attributed to their highly effective pruning capabilities.

6.3 Index Evaluation

In this section, we examine the performance of DLT, focusing on both its construction and maintenance.

Exp-6: DLT Construction. In this experiment, we test the DLT size and construction time on all datasets. The results are shown in Figure 15. Figure 15(a) shows that the index size of DLT is comparable to the graph size in all datasets. For example, the sizes of DLT and the original graph of MG are 2944 MB and 2729 MB, respectively. The compactness of DLT benefits from the design of dominant layers, which allows DLT to store only a small subset of layer combinations. The construction time of DLT is shown in Figure 15(b), where *Decomposition* denotes the `SynCore` decomposition time, *DLT* denotes the time of dominant layers computation and DLT construction, and *Total* is the sum of *Decomposition* and *DLT*. We can observe that the majority of the time is spent on `SynCore` decomposition. Building the DLT itself takes only a small fraction of the total time. On MG dataset, *Decomposition* and *DLT* take 25,477s and 51s, respectively, with DLT Construction accounting for 0.2% of the total time. The reason behind this is that *Decomposition* needs to compute `SynCores` for all possible layer combinations, which takes a lot of time. Notably, even on large dataset MG, *Decomposition* still shows good performance. This is because *Decomposition* employs the reuse technique, which avoids recomputing `SynCore` for each layer combination from scratch.

Exp-7: DLT Maintenance. Finally, we evaluate the performance of DLT maintenance by randomly inserting and removing different numbers of edges, ranging from 2^0 to 2^{10}. Results are shown in Figure 14. *DLT-Ins* and *DLT-Rm* denote the maintenance algorithms for edge insertion and removal, respectively. *Rebuild* denotes to build the index from scratch whenever the multilayer graph updates. We have two key observations. First, both *DLT-Ins* and *DLT-Rm* are more efficient than *Rebuild*. Second, the performance of both *DLT-Ins* and *DLT-Rm* degenerates as the number of inserted/removed edges increases. This is because, as more edges are updated, more `SynCores` are affected, resulting in more sets of dominant layers being changed. Therefore, *DLT-Ins* and *DLT-Rm* require more time to maintain the DLT.

7 CONCLUSION

In this paper, we study the problem of synergetic community search over large multilayer graphs. First, we devise a new cohesive model called synergetic core (`SynCore`). Based on `SynCore`, we formally define the synergetic core community search problem. To solve the problem efficiently, we propose a progressive search algorithm that substantially reduces the search space. To enhance the search efficiency further, we design a novel index called dominant layers-based trie (DLT). We develop a DLT-based search algorithm by traversing DLT to return the community. Extensive experimental results on large real-world datasets validate the effectiveness and efficiency of our proposed model and algorithms. In the future, we would like to investigate the truss-based cohesive subgraph model for multilayer graphs to handle the synergetic community search.

ACKNOWLEDGMENTS

This work was supported in part by the NSFC under Grants No. (62025206, U23A20296, and 62302444), Zhejiang Province's "Lingyan" R&D Project under Grant No. 2024C01259, and HK-RGC Grant C2003-23Y. Yunjun Gao is the corresponding author of the work.

REFERENCES

[1] Esra Akbas and Peixiang Zhao. 2017. Truss-based community search: A truss-equivalence based indexing approach. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1298–1309.

[2] Asim Ansari, Oded Koenigsberg, and Florian Stahl. 2011. Modeling multiple relationships in social networks. *Journal of Marketing Research* 48, 4 (2011), 713–728.

[3] Nicola Barbieri, Francesco Bonchi, Edoardo Galimberti, and Francesco Gullo. 2015. Efficient and effective community search. *Data mining and knowledge discovery* 29 (2015), 1406–1433.

[4] Ali Behrouz, Farnoosh Hashemi, and Laks VS Lakshmanan. 2022. FirmTruss community search in multilayer networks. *Proceedings of the VLDB Endowment* 16, 3 (2022), 505–518.

[5] Lijun Chang, Xuemin Lin, Lu Qin, Jeffrey Xu Yu, and Wenjie Zhang. 2015. Index-based optimal algorithms for computing steiner components with maximum connectivity. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 459–474.

[6] Lu Chen, Chengfei Liu, Rui Zhou, Jiajie Xu, Jeffrey Xu Yu, and Jianxin Li. 2020. Finding effective geo-social group for impromptu activities with diverse demands. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 698–708.

[7] Chaitali Choudhary, Inder Singh, and Manoj Kumar. 2023. Community detection algorithms for recommendation systems: Techniques and metrics. *Computing* 105, 2 (2023), 417–453.

[8] Thomas F Coleman and Jorge J Moré. 1983. Estimation of sparse jacobian matrices and graph coloring blems. *SIAM journal on Numerical Analysis* 20, 1 (1983), 187–209.

[9] Andrea Fronzetti Colladon and Elisa Remondi. 2017. Using social network analysis to prevent money laundering. *Expert Systems with Applications* 67 (2017), 49–58.

[10] Wanyun Cui, Yanghua Xiao, Haixun Wang, Yiqi Lu, and Wei Wang. 2013. Online search of overlapping communities. In *Proceedings of the 2013 ACM SIGMOD international conference on Management of data*. 277–288.

[11] Wanyun Cui, Yanghua Xiao, Haixun Wang, and Wei Wang. 2014. Local search of communities in large graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 991–1002.

[12] DBLP. 2024. https://dblp.uni-trier.de/xml/

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

[14] Yixiang Fang, Reynold Cheng, Xiaodong Li, Siqiang Luo, and Jiafeng Hu. 2017. Effective community search over large spatial graphs. *Proceedings of the VLDB Endowment* 10, 6 (2017), 709–720.

[15] Yixiang Fang, Reynold Cheng, Siqiang Luo, and Jiafeng Hu. 2016. Effective community search for large attributed graphs. *Proceedings of the VLDB Endowment* 9, 12 (2016).

[16] Yixiang Fang, Xin Huang, Lu Qin, Ying Zhang, Wenjie Zhang, Reynold Cheng, and Xuemin Lin. 2020. A survey of community search over big graphs. *The VLDB Journal* 29 (2020), 353–392.

[17] Yixiang Fang, Zhongran Wang, Reynold Cheng, Hongzhi Wang, and Jiafeng Hu. 2018. Effective and efficient community search over large directed graphs. *IEEE Transactions on Knowledge and Data Engineering* 31, 11 (2018), 2093–2107.

[18] 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.

[19] 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.

[20] Farnoosh Hashemi, Ali Behrouz, and Laks VS Lakshmanan. 2022. Firmcore decomposition of multilayer networks. In *Proceedings of the ACM Web Conference 2022*. 1589–1600.

[21] Haiyan Hu, Xifeng Yan, Yu Huang, Jiawei Han, and Xianghong Jasmine Zhou. 2005. Mining coherent dense subgraphs across massive biological networks for functional discovery. *Bioinformatics* 21, suppl_1 (2005), i213–i221.

[22] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. 2021. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430* (2021).

[23] 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)*. 5912–5915.

[24] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying k-truss community in large and dynamic graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 1311–1322.

[25] Xin Huang, Laks VS Lakshmanan, and Jianliang Xu. 2019. *Community search over big graphs*. Morgan & Claypool Publishers.

[26] Xin Huang, Laks VS Lakshmanan, Jeffrey Xu Yu, and Hong Cheng. 2015. Approximate closest community search in networks. *Proceedings of the VLDB Endowment* 9, 4 (2015), 276–287.

[27] 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.

[28] 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.

[29] Yuli Jiang, Yu Rong, Hong Cheng, Xin Huang, Kangfei Zhao, and Junzhou Huang. 2022. Query driven-graph neural networks for community search: From non-attributed, attributed, to interactive attributed. *Proceedings of the VLDB Endowment* 15, 6 (2022), 1243–1255.

[30] KONECT. 2013. http://konect.uni-koblenz.de/networks

[31] Ling Li, Siqiang Luo, Yuhai Zhao, Caihua Shan, Zhengkui Wang, and Lu Qin. 2023. COCLEP: Contrastive learning-based semi-supervised community search. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. 2483–2495.

[32] Rong-Hua Li, Jiao Su, Lu Qin, Jeffrey Xu Yu, and Qiangqiang Dai. 2018. Persistent community search in temporal networks. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. 797–808.

[33] Yuqi Li, Tao Meng, Zhixiong He, Haiyan Liu, and Keqin Li. 2024. A biased edge enhancement method for truss-based community search. *Frontiers of Computer Science* 18, 3 (2024), 183610.

[34] Kar Wai Lim and Wray Buntine. 2016. Bibliographic analysis on research publications using authors, categorical labels and the citation network. *Machine Learning* 103, 2 (2016), 185–213.

[35] Longlong Lin, Pingpeng Yuan, Rong-Hua Li, Chunxue Zhu, Hongchao Qin, Hai Jin, and Tao Jia. 2024. QTCS: Efficient query-centered temporal community search. *Proceedings of the VLDB Endowment* 17, 6 (2024), 1187–1199.

[36] 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.

[37] Dandan Liu and Zhaonian Zou. 2023. gCore: Exploring cross-Layer cohesiveness in multi-layer graphs. *Proceedings of the VLDB Endowment* 16, 11 (2023), 3201–3213.

[38] Qing Liu, Minjun Zhao, Xin Huang, Jianliang Xu, and Yunjun Gao. 2020. Truss-based community search over large directed graphs. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2183–2197.

[39] Qing Liu, Yifan Zhu, Minjun Zhao, Xin Huang, Jianliang Xu, and Yunjun Gao. 2020. VAC: Vertex-centric attributed community search. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 937–948.

[40] R Duncan Luce and Albert D Perry. 1949. A method of matrix analysis of group structure. *Psychometrika* 14, 2 (1949), 95–116.

[41] Chengyang Luo, Qing Liu, Yunjun Gao, and Jianliang Xu. 2024. Synergetic Community Search over Large Multilayer Graphs. https://github.com/ZJU-DAILY/SynCore/blob/main/SynCore-VLDB25-FullVersion.pdf

[42] Fragkiskos D Malliaros, Christos Giatsidis, Apostolos N Papadopoulos, and Michalis Vazirgiannis. 2020. The core decomposition of networks: Theory, algorithms and applications. *The VLDB Journal* 29, 1 (2020), 61–92.

[43] David W Matula and Leland L Beck. 1983. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM (JACM)* 30, 3 (1983), 417–427.

[44] SNAP. 2014. http://snap.stanford.edu/data

[45] Mauro Sozio and Aristides Gionis. 2010. The community-search problem and how to plan a successful cocktail party. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. 939–948.

[46] Longxu Sun, Xin Huang, Zheng Wu, and Jianliang Xu. 2024. Efficient Cross-layer Community Search in Large Multilayer Graphs. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 2959–2971.

[47] Jianwei Wang, Kai Wang, Xuemin Lin, Wenjie Zhang, and Ying Zhang. 2024. Efficient unsupervised community search with pre-trained graph transformer. *Proceedings of the VLDB Endowment* 17, 9 (2024), 2227–2240.

[48] Jianwei Wang, Kai Wang, Xuemin Lin, Wenjie Zhang, and Ying Zhang. 2024. Neural attributed community search at billion scale. *Proceedings of the ACM on Management of Data* 1, 4 (2024), 1–25.

[49] Kai Wang, Wenjie Zhang, Xuemin Lin, Ying Zhang, Lu Qin, and Yuting Zhang. 2021. Efficient and effective community search on large-scale bipartite graphs. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. 85–96.

[50] Run-An Wang, Dandan Liu, and Zhaonian Zou. 2024. FocusCore decomposition of multilayer graphs. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 2792–2804.

[51] Yue Wang, Xun Jian, Zhenhua Yang, and Jia Li. 2017. Query optimal k-plex based community in graphs. *Data Science and Engineering* 2 (2017), 257–273.

[52] Manlio De Domenico's website. 2017. https://manliodedomenico.com/data.php

[53] 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)*. 701–712.