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 the 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 lay 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 nine 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.

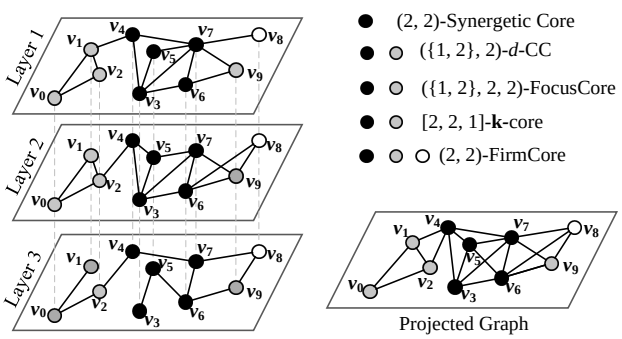
PVLDB Reference Format:
Chengyang Luo, Qing Liu, Yunjun Gao, and Jianliang Xu. Synergetic Community Search over Large Multilayer Graphs. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

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 by $G = (V, E, L)$, can effectively model such graphs with different types of relationships [13]. Specifically, in G, the vertex set V is consistent across all layers in L, with edges on different layers representing different types of relationships. For example, Figure 1(a) depicts a multilayer graph G consisting of three layers.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX



(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, 24, 32, 40]. Prior research has extensively explored community search across various graph types, such as bipartite graphs [46], directed graphs [17, 37], temporal graphs [31, 34], and heterogeneous information networks [18, 26]. In this paper, we focus on community search over multilayer graphs [4].

Numerous cohesive subgraph models have been developed to facilitate community search, including k-core [41], k-truss [25], 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. Consequently, several core-based cohesive subgraph models tailored for multilayer graphs have also been proposed, such as **k**-core [19], d-CC [35, 50], FirmCore [20], and FocusCore [47]. 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, considering 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 model, given parameters k and λ, every vertex is connected to no less than k neighbors across at least λ different layers, with the specific layers varying per vertex. The FocusCore model 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, the existing core-based models for multilayer graphs have their limitations. First, models such as **k**-core, d-CC, and FocusCore require users to specify layers that must satisfy specific degree constraints. For example, the **k**-core necessitates setting k_l for each layer $l \in L$. This requirement to configure multiple parameters can be a considerable challenge, particularly for users who are not well-acquainted with the graph's structure. Second, while FirmCore simplifies parameter setting, it does not consistently ensure cohesiveness within the resulting community. As demonstrated in

our experiments, the community identified by FirmCore is the least cohesive compared to the other three models. This shortcoming stems from FirmCore's design, which does not require a k-core within any individual layer. Last but not least, all existing models focus solely on the local cohesiveness of the multilayer graphs, i.e., only considering the k-core for certain individual layers. They overlook the equally important concept of *global cohesiveness* in multilayer graphs. To be specific, a set of vertices may form dense subgraphs on particular layers. However, when all layers are taken into account concurrently, these vertices may constitute an even denser subgraph, which could lead to the discovery of a significantly more cohesive community. Consider a group of individuals: among them, some may be colleagues, others classmates, and still others friends. From a global perspective, they might all know each other, irrespective of the nature of their relationships.

To address these limitations, we propose a new cohesive subgraph model called *synergetic core* (SynCore), which requires that the subgraph is not only dense within individual layers but also exhibits greater cohesion across all layers. Notably, as observed in our experiments, if we consider only local or global cohesiveness, the resulting communities show inferior densities, confirming the necessity of taking both local and global cohesivenesses into account simultaneously. To this end, we employ the concept of *projected graph* to represent the global aspect of a multilayer graph. Specifically, the projected graph G_P of a multilayer graph G is a simple, single-layer graph that shares the same vertex set as G. In G_P, an edge exists between vertices u and v if there is at least one layer of G in which u and v are connected by an edge. Figure 1(b) shows the projected graph of G in 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 $k' > k$. For instance, the (2, 2)-SynCore in Figure 1 consists of $\{v_3, v_4, v_5, v_6, v_7\}$, which is a 2-core on layers 1 and 2 and a 3-core in the projected graph.

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 a wide range of applications, such as friend recommendation [7, 24], cooperative team identification [33], and fraud detection [9]. For example, in social networks, the friendships of users from different social platforms can be modeled as a multilayer graph, where each layer represents a specific social platform. SynCS can be used to identify a community in which users not only have strong connections on specific platforms but also demonstrate closer friendships on the whole. Such a community is more likely to be a group of people with similar interests, thereby enhancing the effectiveness of friend recommendations within the community. As another example, consider transaction networks from different financial institutions, which can be combined into a multilayer graph. For a suspicious account, SynCS can identify a group of accounts that frequently engage in business activities with this account across multiple institutions. Moreover, this group exhibits higher connectivity from a global perspective, potentially indicating a criminal network.

A basic method to handle SynCS is to enumerate all possible combinations of s layers and compute the (k, s)-SynCore for each combination. In total, there are $\binom{|L|}{s}$ combinations, which is far from efficient. To improve the search efficiency, we propose a progressive search algorithm that reduces the number of layer combinations to be enumerated through two strategies. First, we introduce the concept of Quasi-SynCore, a relaxed version of SynCore. By computing Quasi-SynCores, the original graph can be reduced to a small subgraph that contains the final result in linear time, thereby significantly pruning the search space. Second, we design a set of rules to enumerate potential layer combinations for Quasi-SynCores. These rules accurately identify the layer combinations that may contain the final result, thereby reducing the inefficiency of blindly enumerating all possible combinations.

Given a multilayer graph, all SynCores can be precomputed. By searching for communities within these precomputed SynCores, we can enhance the efficiency of synergetic community search. Thus, we design a novel index called the *dominant layers-based trie* (DLT) to further accelerate the synergetic community search. Directly storing all SynCores requires substantial storage due to the tremendous number of layer combinations. To make the index more compact, we propose a new concept called *dominant layers*, based on the containment relationships between different SynCores. Specifically, a vertex v's dominant layers denote the maximal set of layers, where the SynCore on these layers contains v. With the help of dominant layers, DLT stores only a small fraction of SynCores, significantly reducing storage requirements. Specifically, DLT resorts to SynCore decomposition to compute all sets of dominant layers and organizes them into a trie. Each node in the trie represents a set of dominant layers and is augmented with a vertex set to indicate that the vertex has the dominant layers represented by the node. Based on DLT, we design an efficient trie-based search algorithm to handle SynCS by traversing DLT.

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 nine 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 [41]. The **k**-core [19] with a vector $\mathbf{k} = [k_1, k_2, ..., k_{|L|}]$ requires each layer $l \in L$ to be a k_l-core. The d-CC [35, 50] 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 [47] is a combination of d-CC and FirmCore. Moreover, gCore [36] is proposed for the multilayer heterogeneous information graphs by extending the **k**-core. Other cohesive subgraph models for multilayer graphs include the frequent cross-graph γ-quasi-clique [27], 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 [22] finds subgraphs where each edge is part of at least $k-2$ triangles on all layers of L'. It is worth mentioning that in this paper, we focus on the core-based model. Moreover, existing models only consider local cohesiveness while our proposed model encompass both local and global cohesivenesses.

Community Search. Sozio and Gionnis [43] 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, 23, 25], k-clique [10], k-plex [48], and k-ECC [5]. Community search has also been studied for more complex graphs, including directed graphs [17, 37], bipartite graphs [46], keyword-based graphs [15, 38], location-based graphs [6, 14], temporal graphs [31, 34], and heterogeneous information networks [18, 26]. Recently, several learning-based methods have been proposed for community search. QD-GNN [28], AQD-GNN [28], and ALICE [45] address the community search in a supervised manner. COCLEP [30] employs contrastive learning and tackles community search in a semi-supervised manner. TransZero [44] 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. Unlike 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 is the set of vertices, L represents the set of layers, and $E \subseteq V \times V \times L$ is the set of edges. 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 the neighbor of v. $N_l(v)$ represents the neighbors of v on layer l. $deg_l^G(v) = |N_l(v)|$ denotes the degree of vertex v on layer l in G. Moreover, for a subset of vertices $H \subseteq V$, $G_l[H]$ denotes the subgraph of G induced by H on layer l. Based on the concept of a multilayer graph, we define the projected graph.

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 graphs and projected graphs, 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 are 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, $\forall v \in H, deg_P^H(v) > k$.

A SynCore SC_s^k is k-core on at least s layers of the multilayer graph and a $(k+1)$-core in the projected graph. The vertices in a SynCore synergistically form a tighter subgraph in the projected graph, hence the name. These constraints ensure that the SynCore is not only dense on some layers of the multilayer graph but also exhibits stronger cohesion in the projected graph. If a SynCore forms k-core on each layer in $L' \subseteq L$, we denote it 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., single layer) and global (i.e., the projected graph) cohesivenesses of the multilayer graphs. Second, SynCore only requires two parameters, i.e., k and s. In contrast, the **k**-core model requires $|L|$ different parameters for all layers. The d-CC and FocusCore necessitate manual specification of which layers meet the degree requirements. Less parameters can free the users from the burdensome parameter settings. Third, unlike the FirmCore, SynCore ensures the k-core structure.

Based on the 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, let $k = 2$, $s = 2$, and $Q = \{v_5\}$. The black vertices, i.e., $\{v_3, v_4, v_5, v_6, v_7\}$, represent the result of the SynCS. These vertices form 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. Then, for each layer combination L', we can employ the peeling method to find the 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 the performance issue. Specifically, for each layer combination L', computing the SynCore requires $O(|E| + |V|)$ time. Totally, there are $\binom{L}{s}$ possible

layer combinations. Hence, the total time complexity is $O(\binom{L}{s} \cdot (|E| + |V|))$. Obviously, the basic method is inefficient when $\binom{L}{s}$ is large. Therefore, 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 iterativly 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 $1 \le s \le |L|$, 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 $deg_l^H(v) \ge k$ on each layer $l \in L'$;

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

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, $SC_{L'}^k \subseteq QSC_s^k$.*

Proof. For the $SC_{L'}^k$, it is a k-core on each layer $l \in L'$. Thus, the degrees of any vertex in $SC_{L'}^k$ are no less than k on each layers in L', meeting condition (i) in Definition 4.1. In the projected graph, the degree of each vertex in $SC_{L'}^k$ is greater than k, satisfying condition (ii) in Definition 4.1. Hence, $SC_{L'}^k \subseteq QSC_s^k$. □

According to Lemma 4.1, QSC_s^k is a superset of $SC_{L'}^k$. Hence, we can take the vertices of QSC_s^k as candidate vertices. For example, given $k = 2$, $s = 2$, and $L' = \{1, 2\}$ for multilayer graph in Figure 1. $SC_{\{1,2\}}^2 = \{v_3, v_4, v_5, v_6, v_7\}$, it is a subset of $QSC_2^2 = \{v_3, v_4, v_5, v_6, v_7, v_8, v_9\}$.

Next, we introduce how to compute the Quasi-SynCore efficiently. First, we introduce the concepts of degree vector, $Top\text{-}s$ 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 $Top\text{-}s$ degree of $\mathbf{deg}_{L'}(v)$, denoted by $Top\text{-}s(v)$, is the s-th largest degree in the degree vector $deg_{L'}(v)$. The synergetic degree of v, denoted by $syn\text{-}deg(v)$, is the minimum of the $Top\text{-}s$ degree and the degree in the projected graph minus 1, i.e., $syn\text{-}deg(v) = \min(Top\text{-}s(v), deg_P(v) - 1)$. For example, the degree vector of v_4 in Figure 1 is $\mathbf{deg}_L(v_4) = [3, 3, 2]$, and its degree in the projected graph is 5. So the synergetic degree of v_4 when $s = 2$ is $syn\text{-}deg(v_4) = 3$.

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

Proof. If $syn\text{-}deg(v) < k$, it means $Top\text{-}s(v) < k$ or $deg_P(v) - 1 < k$. If $Top\text{-}s(v) < k$, there does not exist s layers, on which v has k neighbors. If $deg_P(v) - 1 < k$, it does not satisfy the degree constraint on the projected graph. Therefore, $v \notin QSC_s^k$. □

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 $Top\text{-}s$ 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) $syn\text{-}deg(u) \ne deg_P(u) - 1$, and (ii) $\forall l \in L$ with $(v, u, l) \in E$, $syn\text{-}deg(u) \ne deg_l(u)$, $syn\text{-}deg(u)$ does not need to be updated.*

Proof. If $syn\text{-}deg(u) = deg_P(v) - 1$, the deletion of v will decrease u's degree in the projected graph, resulting in a change of $syn\text{-}deg(u)$. Hence, the necessary condition for $syn\text{-}deg(u)$ not to be updated is $syn\text{-}deg(u) \ne deg_P(v) - 1$. In other words, $syn\text{-}deg(u) = Top\text{-}s(u)$.

If $\forall l \in L$ with $(v, u, l) \in E$, $syn\text{-}deg(u) \ne deg_l(u)$, $Top\text{-}s(u) = deg_{l'}(u)$, where $(v, u, l') \notin E$. After deleting v, $Top\text{-}s(u) = deg_{l'}(u)$ and $Top\text{-}s(u) \le deg_P(v) - 2$. Therefore, $syn\text{-}deg(u) = Top\text{-}s(u)$. □

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, $Top\text{-}s$ 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 $Top\text{-}s$ degree and synergetic degree of each affected vertex is updated (lines 18-20). Finally, the Quasi-SynCore is returned (line 21).

Complexity Analysis. Next, we analyze the time complexity of the Quasi-SynCore computation.

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;
```

Theorem 4.4. *The time complexity of of Algorithm 1 is* $O(|L| \cdot (|E| + |V|))$.

Proof. 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. During the computation process, 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 introduce Quasi-SynCore as the candidate for the final result. In this subsection, we present 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 potential layer combination enumeration aims to identify the layer combinations that may contain the final result according to 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 final 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 $|\bigcap_{q \in Q} ML(q)| < |L|$, 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 have the same matched layers, QSC_s^k is a SynCore on L. Therefore, there is no need to enumerate any new layer combination L'. Note that $|L|$ may be larger than 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.

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$'$.

The pseudo-code of PSA is outlined in Algorithm 2. First, PSA performs some initializations (line 1); computes the Quasi-SynCore of the graph G (line 2); set the layer combination of Quasi-SynCore to L (line 3); and adds it to a heap T (line 4). Subsequently, PSA iteratively enumerates potential layer combinations and computes the corresponding Quasi-SynCores (lines 5-31). In each round, PSA checks the Quasi-SynCore QSC_s^k with the largest size (line 6). If the size of QSC_s^k is smaller than the currently found maximum SynCore, QSC_s^k can not contain the solution (lines 7-8). Otherwise, PSA computes the matched layers for all vertices in QSC_s^k based on the (line 9). If all vertices QSC_s^k share the same matched layers, the QSC_s^k is a SynCore. Hence, PSA gets the connected component of QSC_s^k containing the query vertices Q to check whether it is the

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

1 $H \leftarrow \emptyset$; construct projected graph G_P of G;
2 $QSC_s^k \leftarrow$ Quasi-SynCore_Computation$((V, E, L), G_P, k, s)$;
3 $QSC_s^k.PL \leftarrow L$;
4 add QSC_s^k to T; // T stores Quasi-SynCores
5 **while** $T \neq \emptyset$ **do**
6 $QSC_s^k \leftarrow$ pop the Quasi-SynCore with the largest size from T;
7 **if** $|QSC_s^k| < |H|$ **then**
8 **break**;
9 $L_{QSC} \leftarrow QSC_s^k.PL$;
10 compute the matched layers $ML(v)$ for each $v \in QSC_s^k$;
11 **if** $\bigcap_{v \in QSC_s^k} ML(v) = L_{QSC}$ **then** // Termination Rule 1
12 $H' \leftarrow$ the connected component of QSC_s^k containing Q;
13 **if** $|H'| > H$ **then**
14 $H \leftarrow H'$;
15 **continue**;
// Enumeration Rules
16 $PL \leftarrow \emptyset$; // PL stores potential layer combinations
17 **if** $|\bigcap_{q \in Q} ML(q)| < |L_{QSC}|$ **then**
18 $PL \leftarrow \bigcap_{q \in Q} ML(q)$;
19 **else**
20 $PL \leftarrow$ all subsets of L_{QSC} with $|L_{QSC}| - 1$ layers;
21 **foreach** $L' \in PL$ **do**
// Termination Rules 2 and 3
22 **if** $|L'| < s$ *or* L' *has been enumerated* **then**
23 **continue**;
24 $V' \leftarrow$ all vertices of QSC_s^k;
25 **foreach** $v \in V'$ **do**
26 **if** $|ML(v) \cap L'| < s$ **then**
27 $V' \leftarrow V' \setminus v$;
28 $QSC_s'^k \leftarrow$Quasi-SynCore_Computation$((V', E, L'), G_P, k, s)$;
29 $QSC_s'^k.PL \leftarrow L'$;
// Termination Rule 4
30 **if** $QSC_s'^k \neq \emptyset$ *and* $Q \subseteq QSC_s'^k$ **then**
31 add $QSC_s'^k$ to T;
32 **return** H;

final result (lines 11-15). Otherwise, PSA enumerates the layer combinations according to Enumeration Rules (lines 16-20), in which the invalid layer combinations are pruned according to Termination Rules (2) and (3) (lines 21-22). For each remaining potential layer combination L', PSA filters invalid vertices in QSC_s^k (lines 25-27), and employs Algorithm 1 to compute the new Quasi-SynCore $QSC_s'^k$ on the layer L' (line 28). If $QSC_s'^k$ is not empty or Q is included in $QSC_s'^k$, $QSC_s'^k$ will be added to T for further processing (lines 30-31). Finally, PSA returns SynCS (line 32).

Example 2. We use Figure 1 to illustrate the process of PSA. Let $k = 2$, $s = 2$, and $Q = \{v_4\}$. First, PSA computes the Quasi-SynCore for G, i.e., $QSC_2^2 = \{v_3, v_4, v_5, v_6, v_7, v_8, v_9\}$. Then, PSA enumerates potential layer combinations. According to the enumeration rules, only one layer combination $L' = \{1, 2\}$ is enumerated. Thus, PSA computes the Quasi-SynCore $QSC_2'^2$ for $L' = \{1, 2\}$, $QSC_2'^2 = \{v_3, v_4, v_5, v_6, v_7\}$. $QSC_s'^k$ is the SC_2^2 of G, and thus is returned as the final result.

Complexity Analysis. We analyze the time complexity of PSA. 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.

Theorem 4.5. *The time complexity of PSA is* $O(\mathcal{N} \cdot |L| \cdot |V| + \mathcal{N}' \cdot |L| \cdot (|E| + |V|))$.

Proof. 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 $O(\mathcal{N} \cdot |L| \cdot |V| + \mathcal{N}' \cdot |L| \cdot (|E| + |V|))$. □

In practice, $\mathcal{N}$ and $\mathcal{N}'$ are much smaller than $\binom{L}{s}$. Consequently, the efficiency of the progressive search algorithm is significantly better than the basic method, as demonstrated in experiments.

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 for efficient synergetic community search.

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. Firstly, we introduce some features of the SynCore, which are useful for index design.

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 have inclusion relations for different layers and values of k. Inspired by this, we propose 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 example, in Figure 1, $v_4 \in SC_{\{1\}}^2$, $v_4 \in SC_{\{1,2\}}^2$, and $v_4 \notin SC_{\{1,2,3\}}^2$. Hence, $\{1, 2\}$ are the dominant layers of v_4. 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

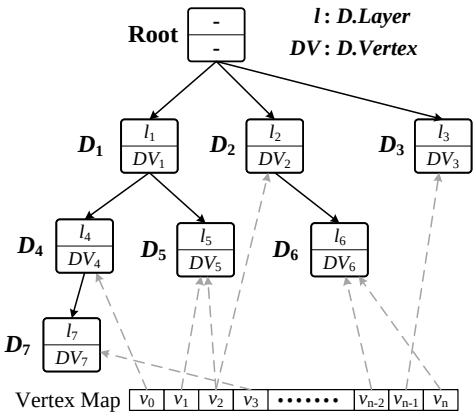


Figure 2: The structure of DLT

DLT is shown in Figure 2. 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 3. We take Figure 1 as an example. Let $k = 2$. The set of dominant layers of each vertex is shown in Figure 3(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 3(b).

Complexity Analysis. We analyze the space complexity of the DLT. Let θ be the average number of sets of dominant layers for a vertex when k is specified.

Theorem 5.1. *The space complexity of all DLTs is* $O(\theta \cdot |E| \cdot |L|)$.

Proof. 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

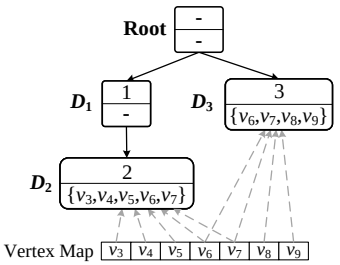
In this subsection, we present DLT construction method. The basic idea of DLT construction is to firstly compute all SynCores for a given multilayer graph. Then, we compute all sets of the dominant layers, based on which DLT is built. First, we introduce SynCore decomposition to compute all SynCores.

SynCore Decomposition.

We formally define the SynCore decomposition as follows.

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 3: Examples of Dominant Layers and DLT

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 $\forall L' \subseteq L$.

According to Definition 5.3, a straightforward way for SynCore decomposition is to compute SynCore for all possible L's, which is costly. To efficiently perform SynCore decomposition, we exploit the local property of SynCore.

Lemma 5.2. *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 $C_{L'}(u_P) \geq k$.

Proof. (1) *Necessity.* If $C_{L'}(v) = k$, then $v \in SC^k_{L'}$. On each layer $l \in L'$, there exists k neighbors belonging to $SC^k_{L'}$. In the projected graph, there exists $k+1$ neighbors included in $SC^k_{L'}$. Suppose k is not the largest value that satisfies the conditions, then $v \in SC^{k+1}_{L'}$. This contradicts to $C_{L'}(v) = k$.

(2) *Sufficiency.* According to conditions (i) and (ii), v and its neighbors form a k-core on each layer $l \in L$, and they form a $(k+1)$-core in the projected graph. Therefore, $v \in SC^k_{L'}$. Since k is the largest value that satisfies the conditions, the SynCoreness of v is k. □

According to Lemma 5.2, $C_{L'}(v)$ 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, $C_{L'}(v) \leq C_{L''}(v)$. 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 5.3. *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.*

Proof. If $\exists l \in L'$ such that $sup_l(v) < \overline{C}_{L'}(v)$, v and its neighbors cannot form a $\overline{C}_{L'}(v)$-core on layer l. Similarly, if $sup_P(v) < \overline{C}_{L'}(v) + 1$, v and its neighbors cannot form a $(\overline{C}_{L'}(v) + 1)$-core in the projected graph. Therefore, the $\overline{C}_{L'}(v)$ should be updated. □

How to update the $\overline{C}_{L'}(v)$? When v satisfies Lemma 5.3, $\overline{C}_{L'}(v)$ 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 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 5.3 (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.

Algorithm 3: SynCore Decomposition

Input: multilayer graph G
Output: the SynCorenesses of all vertices
1 construct projected graph G_P of G;
2 **foreach** $s = 1$ *to* $|L|$ **do**
3 **foreach** *layers* $L' \subseteq L$ *and* $|L'| = s$ **do**
4 **foreach** $v \in V$ **do**
5 **if** $s = 1$ **then**
6 $\overline{C}_{L'}(v) \leftarrow deg_P(v)$;
7 **else**
8 $\overline{C}_{L'}(v) \leftarrow \min_{L'' \subset L'} C_{L''}(v)$;
9 $C'_L \leftarrow$ Compute_SynCoreness$(G, G_P, L', \overline{C}_{L'})$
10 **return** C'_L for all $L' \subseteq L$

Procedure Compute_SynCoreness$(G, G_P, L', \overline{C}_{L'})$
11 **foreach** $v \in V$ **do**
12 **foreach** $l \in L'$ **do**
13 $sup_l(v) \leftarrow |\{u \in N_l(v) \mid \overline{C}_{L'}(u) \geq \overline{C}_{L'}(v)\}|$;
14 $sup_P(v) \leftarrow |\{u \in N_P(v) \mid \overline{C}_{L'}(u) \geq \overline{C}_{L'}(v)\}|$;
15 **while** $\exists v \in V, l \in L'$ *such that* $sup_l(v) < \overline{C}_{L'}(v)$ *or* $sup_P(v) \leq \overline{C}_{L'}(v) + 1$ **do**
16 $ub \leftarrow \overline{C}_{L'}(v)$;
17 **foreach** $l \in L'$ **do**
18 $k_l(v) \leftarrow \max k \ s.t. |\{u \in N_l(v) \mid \overline{C}_{L'}(u) \geq k\}| \geq k$;
19 $\overline{C}_{L'}(v) \leftarrow \min(k_l(v), \overline{C}_{L'}(v))$;
20 $k_P(v) \leftarrow \max k \ s.t. |\{u \in N_P(v) \mid \overline{C}_{L'}(u) \geq k\}| \geq k + 1$;
21 $\overline{C}_{L'}(v) \leftarrow \min(k_P(v), \overline{C}_{L'}(v))$;
22 **foreach** $l \in L'$ **do**
23 $sup_l(v) \leftarrow |\{u \in N_l(v) \mid \overline{C}_{L'}(u) \geq \overline{C}_{L'}(v)\}|$;
24 **foreach** $u \in N_l(v)$ **do**
25 **if** $\overline{C}_{L'}(u) \leq ub$ *and* $\overline{C}_{L'}(u) > \overline{C}_{L'}(v)$ **then**
26 $sup_l(u) \leftarrow sup_l(u) - 1$;
27 $sup_P(v) \leftarrow |\{u \in N_P(v) \mid \overline{C}_{L'}(u) \geq \overline{C}_{L'}(v)\}|$;
28 **foreach** $u \in N_P(v)$ **do**
29 **if** $\overline{C}_{L'}(u) \leq ub$ *and* $\overline{C}_{L'}(u) > \overline{C}_{L'}(v)$ **then**
30 $sup_P(u) \leftarrow sup_P(u) - 1$;
31 **return** $\overline{C}_{L'}$;

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. Next, we analyze the time and space complexity of the DLT construction algorithm. Let θ be the average

Algorithm 4: DLT Construction Algorithm

Input: multilayer graph G
Output: DLT

All SynCorenesses ←SynCore_Decomposition(G);
$DLT \leftarrow \emptyset$;
foreach $k = 0$ *to* k_{max} **do**
  initialize a root trie node D_{root};
  $DLT[k] \leftarrow D_{root}$;
  foreach $v \in V$ **do**
    $DL_k(v) = \{L' \subseteq L \mid C_{L'}(v) \geq k \wedge \nexists L'' \subseteq L$ such that $L' \subset L''$ and $C_{L''}(v) \geq k\}$;
    foreach $L' \in DL_k(v)$ **do**
      $D \leftarrow DLT[k]$; // D is a root node
      foreach $l \in L'$ **do**
        if $\exists\ D' \in D.child$ *such that* $D'.Layer = l$ **then**
          $D \leftarrow D'$;
        else
          create a new node D'; $D'.Layer = l$;
          $D.child.add(D')$; $D \leftarrow D'$;
        if l *is the last number of* L' **then**
          $D.Vertex.add(v)$;
          $VM_k(v).add(D)$;
return DLT;

number of dominant layers for each vertex when k is specified, and U_{max} be the maximum upper bound of SynCoreness for any vertex.

Theorem 5.4. *The time complexity of Algorithm 4 is* $O(2^{|L|} \cdot (|L| \cdot |V| + U_{max} \cdot |E|) + \theta \cdot |E| \cdot |L|)$.

Proof. 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 $O(2^{|L|} \cdot (|L| \cdot |V| + U_{max} \cdot |E|)$. 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 combiantion 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

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

1 $H \leftarrow \emptyset$; $setL \leftarrow$ all combinations of s layers;
2 **foreach** $q \in Q$ **do**
3 $tempSetL \leftarrow \emptyset$;
4 **foreach** $D \in VM_k(q)$ **do**
5 $L' \leftarrow$ layer combination represented by D;
6 **foreach** $L'' \subseteq L'$ *and* $|L''| = s$ **do**
7 $tempSetL.add(L'')$;
8 $setL \leftarrow tempSetL \cap setL$;
9 **foreach** $L' \in setL$ **do**
10 $S.push((DLT[k], 0))$ // S is a stack
11 $H' \leftarrow \emptyset$;
12 **while** $S \neq \emptyset$ **do**
13 $(D, i) \leftarrow$ pop the top element from S;
14 **if** $D.Layer = L'[i]$ **then**
15 **if** $i < |L'| - 1$ **then**
16 $i \leftarrow i + 1$;
17 **else**
18 **foreach** D' *in the subtree of* D **do**
19 add $D'.Vertex$ to H';
20 **continue**;
21 **foreach** $D' \in D.Child$ **do**
22 **if** $D'.Layer \leq L'[i]$ **then**
23 $S.push((D', i))$;
24 $H' \leftarrow$ connected component of H' that contains Q;
25 **if** $|H'| \geq |H|$ **then**
26 $H \leftarrow H'$;
27 **return** H;

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 4. In the DLT of Figure 3(b), given $k = 2$, $s = 2$, and $Q = \{v_5\}$, since v_5 is in the dominant vertex set of node "2", the possible layer combination is $\{1, 2\}$. Then, DSA starts from the root, node "1" and node "2" are matched sequentially, and $\{v_3, v_4, v_5, v_6, v_7\}$ are added to the result. Node "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.

Theorem 5.5. *The time complexity of Algorithm 5 is* $O(\lambda \cdot (\mathcal{H} + |L| \cdot \log(|L|)))$.

Proof. 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 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

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

Dataset	Abbr.	$\|V\|$	$\|E\|$	$\|L\|$	$\overline{\|DL(v)\|}$
Slashdot [20]	SD	51**K**	139**M**	6	1.3
Homo [49]	HM	18**K**	153**K**	7	6.1
SacchCere [49]	SC	6.5**K**	247**K**	7	29.0
DBLP [19]	DB	512**K**	1.0**M**	10	1.4
Obama [49]	OB	2.2**M**	3.8**M**	3	0.5
YEAST [49]	YE	4.5**K**	8.5**M**	4	1635.2
Higgs [49]	HI	456**K**	13**M**	4	27.5
StackOverflow [42]	SO	2.6**M**	64.5**M**	9	8.5
Wiki [29]	WK	6.9**M**	129**M**	10	5.9

Table 2: Community quality comparison for different models

Datasets	Models	*D-Avg*	*D-Min*	*D-P*	*GCC-Avg*	*GCC-Min*	*GCC-P*
HM	SynCore	**0.4684**	**0.4541**	**0.6387**	**0.8337**	**0.8037**	**0.8512**
	d-CC	0.3105	0.3000	0.4579	0.6717	0.6525	0.6769
	FirmCore	0.0012	0.0007	0.0072	0.0732	0.0126	0.0562
	FocusCore	0.3917	0.3417	0.5333	0.7906	0.7069	0.7801
	k-core	0.3874	0.3251	0.5419	0.6844	0.4132	0.7685
	(k+1)-core	0.0004	0.0003	0.0027	0.0472	0.0067	0.0309
	FirmTruss	0.0630	0.0012	0.2683	0.3425	0.1250	0.4567
SO	SynCore	**0.7420**	**0.6812**	**0.9217**	**0.7520**	**0.7372**	**0.9311**
	d-CC	0.4800	0.4358	0.7222	0.6049	0.5208	0.8115
	FirmCore	0.0004	0.0002	0.0031	0.0441	0.0281	0.2264
	FocusCore	0.5234	0.4804	0.8198	0.6449	0.6219	0.8511
	k-core	0.5208	0.4906	0.7943	0.6557	0.6277	0.8406
	(k+1)-core	0.0006	0.0001	0.0021	0.0152	0.0123	0.2945
	FirmTruss	0.1899	0.0083	0.6897	0.5627	0.2126	0.8122

vertices whose SynCoreness has changed. Let $C^*_{L'}(v)$ denote the new SynCoreness of v on layers $L' \subseteq L$.

For each affected vertex, we first determine its new sets of dominant layers when k is specified, denoted as $DL^*_k(v)$. Specifically, $DL^*_k(v) = \{L' \subseteq L \mid C^*_{L'}(v) \geq k \wedge \nexists L'' \subseteq L$ such that $L' \subset L''$ and $C^*_{L''}(v) \geq k\}$. 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 set of newly added dominant layers L^+, let $V(L^+)$ denote the vertices whose newly added dominant layers set contains L^+, i.e., $V(L^+) = \{v \in V \mid L^+ \in DL^+_k(v)\}$. Each set of newly added dominant layers L^+ is then visited starting from the root of the DLT. If a corresponding node does not exist, a new DLT node is created. For the DLT node with the last number in L^+, we add $V(L^+)$ to its dominant vertex set. For each set of deleted dominant layers L^-, let $V(L^-)$ denote the vertices whose deleted dominant layers set contains L^-, i.e., $V(L^-) = \{v \in V \mid L^- \in DL^-_k(v)\}$. Each deleted set of dominant layers L^- is then visited starting from the root of the DLT. For the node with the last number in L^-, $V(L^-)$ is deleted from its dominant vertex set. If the subtree rooted at that node has no other non-empty dominant vertex sets, the subtree is deleted.

6 EXPERIMENTS

This section evaluates the performance of our proposed algorithms and index. In experiments, we employ nine real-world multilayer graphs, which are summarized in Table 1. All algorithms are implemented in C++ and compiled by GCC 9.3.0 with -O3 optimization. All experiments are 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 128G memory. In each experiment, we run 1000 queries and report the average results.

6.1 Effectiveness Evaluation

Firstly, we study the effectiveness of our proposed model SynCore.

Exp-1: Community quality comparison for different models. We assess the quality of communities retrieved by different models.

Competitors. We compare SynCore with five cohesive subgraph models for multilayer graphs, i.e., d-CC [35, 50], FirmCore [20], FocusCore [47], **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 use two types of metrics: the *density* ($\frac{2\cdot|E|}{|V|\cdot(|V|-1)}$) [8] and the *global clustering coefficient* ($\frac{3\cdot|\Delta|}{|triplet|}$) [39]. Let H be the returned community; $D_l[H]$ and $GCC_l[H]$ (rep. $D_P[H]$ and $GCC_P[H]$) be the density and global clustering coefficient of H on layer l (rep. projected graph), respectively; L_c be the set of layers that satisfies the degree constraints of cohesive subgraph models for multilayer graphs.

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

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

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

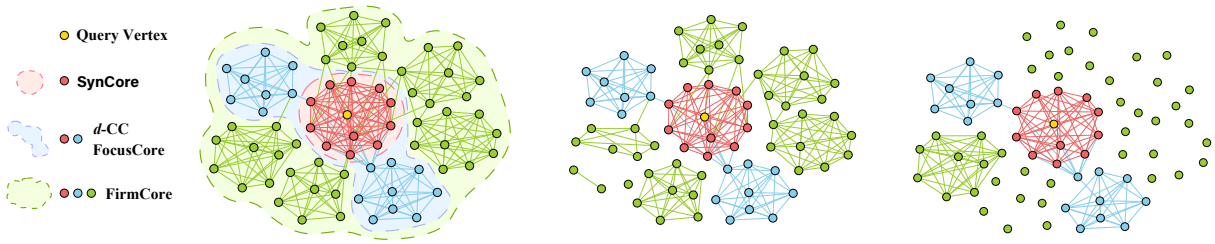
$$GCC\text{-}Avg = \frac{\sum_{l\in L_c} GCC_l[H]}{|L_c|},$$

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

For fair comparisons, we set the same values for the same parameters under different models.

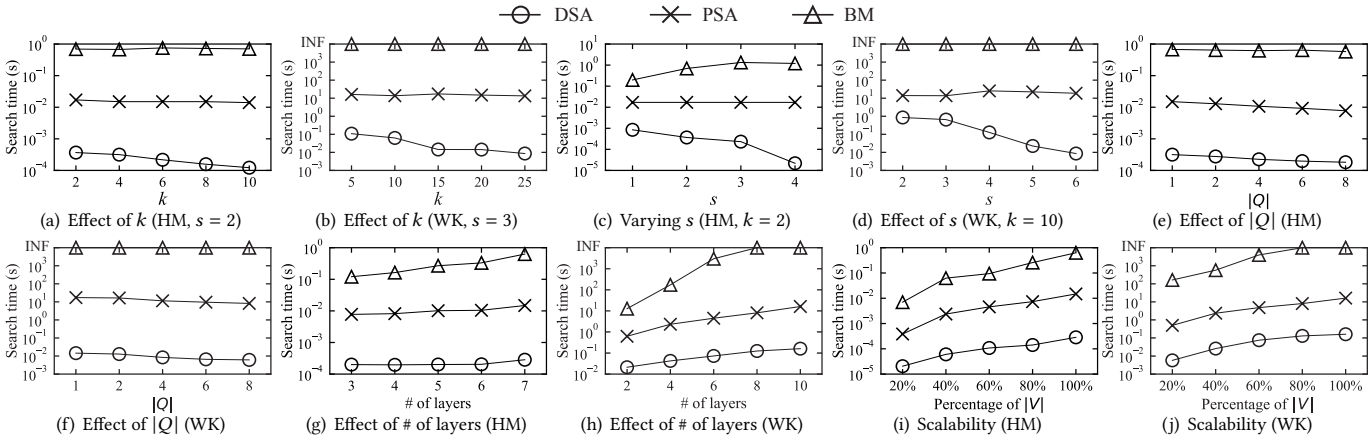
Results. Table 2 shows the results on datasets HM and SO. We can observe that the SynCore has the best performance under all metrics. The reason behind this is that SynCore requires the community should be a cohesive structure on both some layers and the projected graph, which enables SynCore to identify more cohesive communities. Other models, such as d-CC and **k**-core, only consider cohesiveness within partial layers, resulting in inferior densities and global clustering coefficients compared with SynCore. The FirmCore and FirmTruss do not require vertices to form a dense structure on layers, resulting in communities of low quality. Although FocusCore takes into account the density of the background layer, it is merely a simple combination of d-CC and FirmCore, which does not significantly improve the quality of the community. All the above models only require density for certain layers. In addition, the $(k+1)$-core in the projected graph has the worst performance, demonstrating that the high quality of SynCore does not rely solely on the constraints of the projected graph. Therefore, the results justify the design of SynCore.

Exp-2: Case Study on DBLP. In this experiment, we conduct a case study on DBLP [12]. We collect papers from VLDB, SIGMOD, ICDE, and KDD, and construct a four-layer graph. Each vertex in the graph represents an author, and each layer represents a conference. In a



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

Figure 4: Case study on DBLP



(a) Effect of k (HM, $s = 2$) (b) Effect of k (WK, $s = 3$) (c) Varying s (HM, $k = 2$) (d) Effect of s (WK, $k = 10$) (e) Effect of $|Q|$ (HM)

(f) Effect of $|Q|$ (WK) (g) Effect of # of layers (HM) (h) Effect of # of layers (WK) (i) Scalability (HM) (j) Scalability (WK)

Figure 5: Efficiency evaluation

layer, an edge exists between two authors if the paper coauthored by them has been published in the corresponding conference. We set the query vertex $Q = \{\text{Samuel Madden}\}$; $k = k_{max} = 7$ to ensure the returned community is non-empty; and $s = 2$. For the d-CC and FocusCore, we set SIGMOD and ICDE as the specified layers. For the FirmCore, we set $\lambda = 2$. The returned communities are visulized in Figure 4, in which red vertices represent authors returned by `SynCore`; blue vertices represent authors identified by d-CC and FocusCore; and green vertices represent authors returned by FirmCore. The community identified by SynCore stands out as the most cohesive, clearly forming a closely-knit group of academic partners. They exhibit strong cohesiveness not only within the SIGMOD and ICDE, but also form a tighter unit in 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, the group of blue vertices in the upper left has only one edge connecting to the red vertices. Clearly, it should not be included in the result.

6.2 Efficiency Evaluation

In this section, we study the efficiency of our proposed algorithms, including the *DLT-based Search Algorithm* (DSA), the *Progressive Search Algorithm* (PSA), and the *Basic Method* (BM). The tested parameters include k, s, $|Q|$, $|L|$, and $|V|$. Note that in each experiment, we vary only one parameter while keeping the others at their default values. Specifically, the default values of k and s are determined by specific datasets; the default value $|Q|$ is 1; and the default values of $|L|$ and V are set to the original datasets. We report the running time for each algorithm. If the algorithm does not complete within 10^4 seconds, we denote it by INF.

Exp-3: Effect of k. First, we explore the effect of parameter k. Figures 5(a) and 5(b) show the results of Homo and Wiki, respectively. We have the following observations. (1) DSA demonstrates the best performance among the three algorithms, being up to six orders of magnitude faster than BM. Additionally, PSA is up to three orders of magnitude faster than BM. DSA achieves the best performance because it uses the DLT index to store the precomputed SynCore decomposition results, avoiding multiple traversals of multilayer graphs. In contrast, BM must enumerate all layer combinations and use the peeing technique to find the `SynCore` for each layer combination, resulting in the worst performance. Although PSA improves upon BM by pruning many unqualified layer combinations, it still requires graph traversal to compute the `SynCore`. Consequently, PSA's performance is better than BM but inferior to DSA. (2) As k increases, the search time of DSA decreases, while the search times of PAS and BM remain almost unchanged. It is because the larger k, the smaller `SynCore`. Thus, DSA traverses fewer DLT nodes to find `SynCore`, reducing the search time. For PSA and BM, the parameter k does not influence the number of layer combinations. Hence, their search times almost keep the same.

Exp-4: Effect of s. Next, we study the effect of s. The results for the datasets Homo and Wiki are shown in Figures 5(c) and 5(d), respectively. First, the search time of the DSA decreases as s increases. This is because a larger s results in a smaller `SynCore`, reducing the search space in the DLT index. Second, as s increases, the number of layer combinations also increases, causing BM to take more time. Third, although a larger s leads to more layer combinations, PSA can prune these layer combinations to a small number, resulting in stable performance. Both DAS and PSA consistently outperform BM across different values of s.

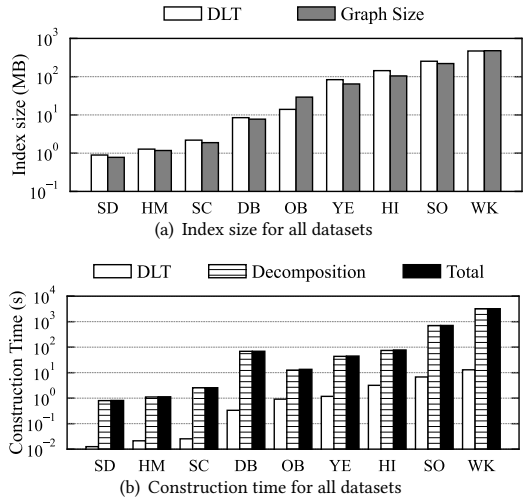


Figure 6: DLT construction evaluation

Exp-5: Effect of $|Q|$. Then, we test the effect of the number of query vertices by varying $|Q|$ from 1 to 8. The results are shown in Figures 5(e) and 5(f). As the number of query vertices increases, the search times for DSA and PSA slightly decrease, while the search time for BM shows no obvious change. For DSA and PSA, more query vertices lead to fewer layer combinations, accelerating the search process. In contrast, BM needs to compute SynCore for all layer combinations, whose number does not change for different $|Q|$ values. Hence, the number of query vertices does not significantly affect BM's search time.

Exp-6: Effect of $|L|$. In this experiment, we vary the number of layers in the multilayer graphs and evaluate the performance of three algorithms. The results are shown in Figures 5(g) and 5(h). We observe that as $|L|$ increases, the performance of all algorithms deteriorates. This is because more layers in the multilayer graphs result in more layer combinations, thereby expanding the search space and increasing the search time. Notably, despite the varying number of layers, PSA remains 1 to 2 orders of magnitude faster than BM, while DSA maintains approximately 2 orders of magnitude faster performance than PSA.

Exp-7: Scalability. Finally, we evaluate the scalability of three algorithms under different graph sizes. In this experiment, we randomly sample a certain number of vertices from the original graph to generate a set of graphs with different cardinalities. Results are shown in Figures 5(i) and 5(j). As expected, the performance of all algorithms degenerates when the graph cardinality increases. The reason is that the larger the graph, the larger the community. However, DSA and PSA consistently outperform BM by several orders of magnitude.

6.3 Index Evaluation

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

Exp-8: DLT Construction. In this experiment, we test the DLT size and construction time on all datasets. The results are shown in Figure 6. In Figure 6(a), we can see that the index size of DLT is comparable to the graph size across all datasets. For example, for the WK dataset, the sizes of DLT and the original are 469.3MB and 477.1MB, respectively. The compactness of DLT benefits from the design of dominant layers, which allows DLT to store only a limited subset of layer combinations rather than the entirety of possible combinations. In addition, the construction time of DLT is displayed in Figure 6(b), in which *Decomposition* denotes the SynCore decomposition time, *DLT* denotes the time of dominant layers computation and DLTs 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. Take the WK dataset for instance, *Decomposition* and *DLT* take 3209.4s and 12.9s, respectively, with *DLT* accounting for 0.4% 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 for large datasets, such as WK, *Decomposition* still shows good performance. This is because *Decomposition* employs the reuse technique, eliminating the need to compute SynCore for each layer combination from scratch.

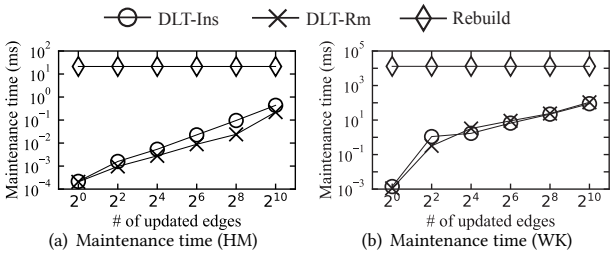


Figure 7: DLT maintenance evaluation

Exp-9: 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}. The maintenance times on HM and WK are shown in Figure 7. *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 graphs update. 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* degenerate 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 synergeticc 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 significantly prunes the search space. To further improve search efficiency, we design a novel index called the dominant layers-based trie (DLT). We propose a DLT-based search algorithm by traversing DLT to return the community. Extensive experimental results on large real-world networks confirm the effectiveness and efficiency of our proposed model, algorithms, and index. In the future, we would like to explore the truss-based cohesive subgraph model for multilayer graphs to handle the synergetic community search.

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] 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.

[23] 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.

[24] Xin Huang, Laks VS Lakshmanan, Jianliang Xu, and HV Jagadish. 2019. *Community search over big graphs*. Vol. 14. Springer.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

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

[30] 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)*. IEEE, 2483–2495.

[31] 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)*. IEEE, 797–808.

[32] 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.

[33] 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.

[34] 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.

[35] Boge Liu, Fan Zhang, Chen Zhang, Wenjie Zhang, and Xuemin Lin. 2019. Core-cube: 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.

[36] 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.

[37] 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.

[38] 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)*. IEEE, 937–948.

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

[40] 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.

[41] 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.

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

[43] 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.

[44] 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.

[45] 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.

[46] 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)*. IEEE, 85–96.

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

[48] 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.

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

[50] 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.

APPENDIX

A SUPPLEMENTARY EXPERIMENTAL RESULTS

This section presents additional experimental results that are not included in the main text.

Exp-1: Community quality comparison for different models. We assess the quality of communities retrieved by different models. We show the results on SC and HI in Table 3.

Table 3: Community quality comparison for different models

	Models	*D-Avg*	*D-Min*	*D-P*	*GCC-Avg*	*GCC-Min*	*GCC-P*
SC	SynCore	**0.9286**	**0.8571**	**0.9321**	**0.9423**	**0.8846**	**0.9888**
	d-CC	0.4818	0.4364	0.5273	0.8912	0.8824	0.8824
	FirmCore	0.0028	0.0001	0.0179	0.0917	0.0540	0.1121
	FocusCore	0.4818	0.4364	0.5273	0.8912	0.8824	0.8824
	k-core	0.4721	0.4524	0.5470	0.9086	0.8599	0.9090
	(k+1)-core	0.0022	0.0001	0.0140	0.0885	0.0518	0.1067
	FirmTruss	0.1006	0.0207	0.4591	0.4229	0.2870	0.6078
HI	SynCore	**0.2718**	**0.2213**	**0.3940**	**0.5058**	**0.4287**	**0.5940**
	d-CC	0.1399	0.0989	0.2217	0.3737	0.3037	0.4578
	FirmCore	0.0003	0.0001	0.0010	0.0083	0.0016	0.0223
	FocusCore	0.1644	0.1162	0.2539	0.3945	0.3341	0.4797
	k-core	0.1377	0.0983	0.2237	0.3966	0.3272	0.4549
	(k+1)-core	0.0004	0.0002	0.0002	0.0030	0.0004	0.0100
	FirmTruss	0.0464	0.0023	0.1385	0.1642	0.0840	0.3427

Exp-3: Effect of k. First, we explore the effect of parameter k. Figure 8 show the results. We have the following observations. (1) DSA demonstrates the best performance among the three algorithms, being up to six orders of magnitude faster than BM. Additionally, PSA is up to three orders of magnitude faster than BM. DSA achieves the best performance because it uses the DLT index to store the precomputed SynCore decomposition results, avoiding multiple traversals of multilayer graphs. In contrast, BM must enumerate all layer combinations and use the peeing technique to find the SynCore for each layer combination, resulting in the worst performance. (2) As k increases, the search time of DSA decreases, while the search times of PAS and BM remain almost unchanged. It is because the larger k, the smaller SynCore. Thus, DSA traverses fewer DLT nodes to find SynCore, reducing the search time. For PSA and BM, the parameter k does not influence the number of layer combinations. Hence, their search times almost keep the same.

Exp-4: Effect of s. Next, we study the effect of s. The results are shown in Figure 9. First, the search time of the DSA decreases as s increases. This is because a larger s results in a smaller SynCore, reducing the search space in the DLT index. Second, as s increases, the number of layer combinations also increases, causing BM to take more time. Third, although a larger s leads to more layer combinations, PSA can prune these layer combinations to a small number, resulting in stable performance. Both DAS and PSA consistently outperform BM across different values of s.

Exp-5: Effect of $|Q|$. Then, we test the effect of the number of query vertices by varying $|Q|$ from 1 to 8. The results are shown in Figure 10. As the number of query vertices increases, the search times for DSA and PSA slightly decrease, while the search time for BM shows no obvious change. For DSA and PSA, more query vertices lead to fewer layer combinations, accelerating the search process. In contrast, BM needs to compute SynCore for all layer combinations, whose number does not change for different $|Q|$ values. Hence, the number of query vertices does not significantly affect BM's search time.

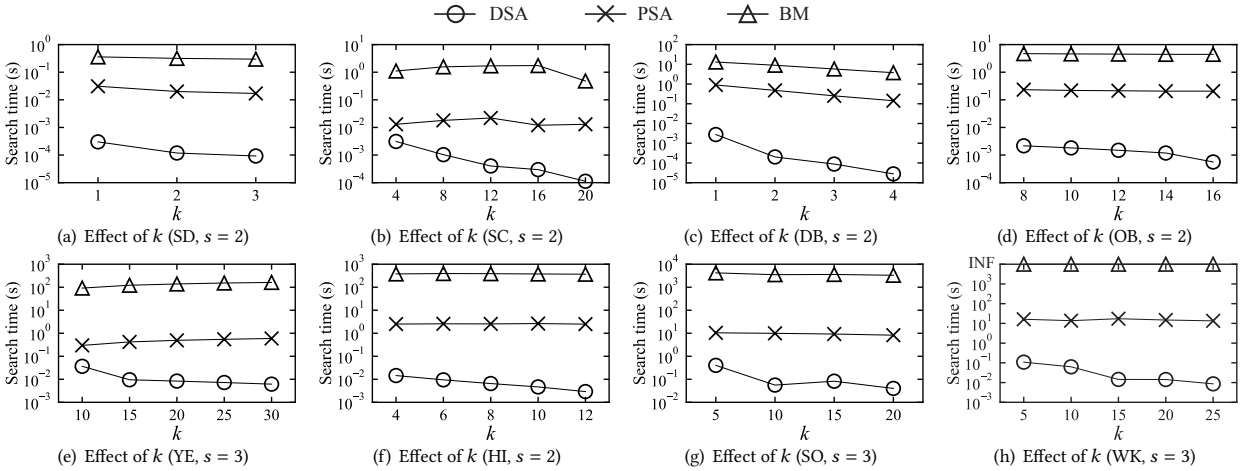
Exp-6: Effect of $|L|$. In this experiment, we vary the number of layers in the multilayer graphs and evaluate the performance of three algorithms. The results are shown in Figure 11. We observe that as $|L|$ increases, the performance of all algorithms deteriorates. This is because more layers in the multilayer graphs result in more layer combinations, thereby expanding the search space and increasing the search time. Notably, despite the varying number of layers, PSA remains 1 to 2 orders of magnitude faster than BM, while DSA maintains approximately 2 orders of magnitude faster performance than PSA.

Exp-7: Scalability. Finally, we evaluate the scalability of three algorithms under different graph sizes. In this experiment, we randomly sample a certain number of vertices from the original graph to generate a set of graphs with different cardinalities. Results are shown in Figure 12. As expected, the performance of all algorithms degenerates when the graph cardinality increases. The reason is that the larger the graph, the larger the community. However, DSA and PSA consistently outperform BM by several orders of magnitude.

Exp-9: 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}. The maintenance times 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 graphs update. 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* degenerate 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.

Exp-10: Scalability of DLT. Then, we evaluate the scalability of DLT by varying the graph sizes from 20%$|V|$ to 100%$|V|$. The index sizes and construction times of DLT are plotted in Figure 13. As expected, the index size and construction time of DLT increase as the dataset size grows. The reason behind this is that larger datasets store more dominant layers and vertices in the DLT. Thus, the DLT becomes larger and requires more construction time. However, regardless of the dataset size, the DLT remains consistently comparable in size to the original graph.

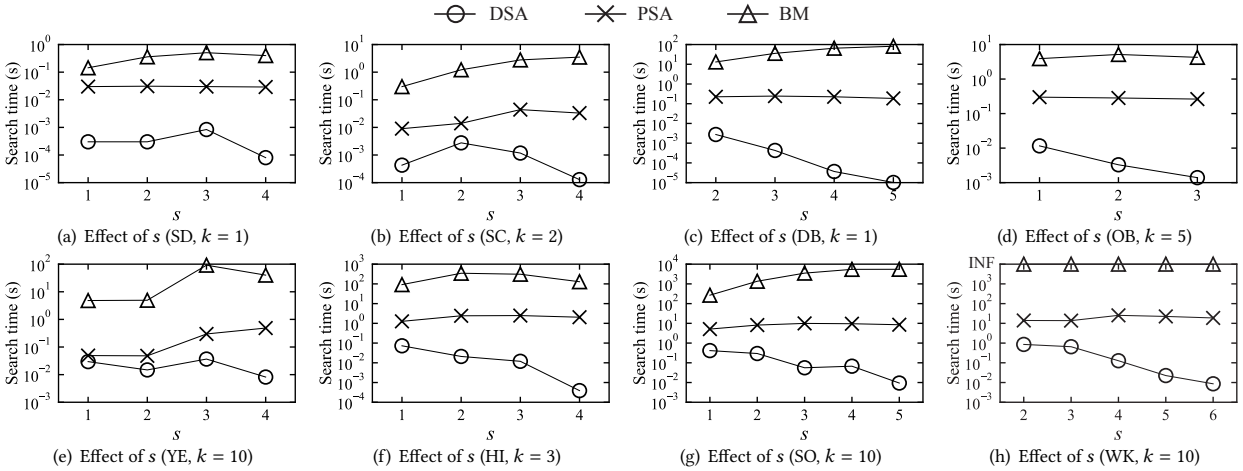
Exp-11: Effect of $|L|$ on DLT. Next, we vary the number of layers in the datasets to test the performance of DLT. To this end, we randomly select a certain number of layers to generate multilayer graphs with different layer counts. The index size and construction time are shown in Figure 15. As the number of layers increases, both the index size and construction time increase. This is because as the number of layers increases, the number of non-empty SynCores will also increase, leading to more dominant layers. Essentially, the index stores all dominant layers, so DLT becomes larger and takes more time to construct. However, the index size of DLT is consistently comparable to the graph size. Additionally, since DLT builds on the results of the SynCore decomposition, its construction time remains a small fraction of the total build time.



(a) Effect of k (SD, $s = 2$) (b) Effect of k (SC, $s = 2$) (c) Effect of k (DB, $s = 2$) (d) Effect of k (OB, $s = 2$)

(e) Effect of k (YE, $s = 3$) (f) Effect of k (HI, $s = 2$) (g) Effect of k (SO, $s = 3$) (h) Effect of k (WK, $s = 3$)

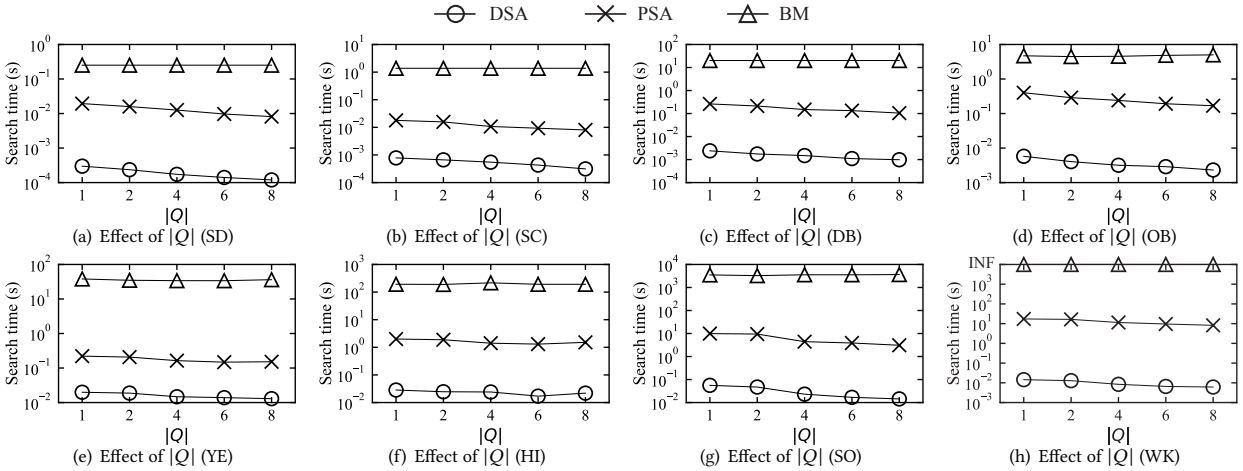
Figure 8: Effect of k on search algorithms



(a) Effect of s (SD, $k = 1$) (b) Effect of s (SC, $k = 2$) (c) Effect of s (DB, $k = 1$) (d) Effect of s (OB, $k = 5$)

(e) Effect of s (YE, $k = 10$) (f) Effect of s (HI, $k = 3$) (g) Effect of s (SO, $k = 10$) (h) Effect of s (WK, $k = 10$)

Figure 9: Effect of s on search algorithms



(a) Effect of $|Q|$ (SD) (b) Effect of $|Q|$ (SC) (c) Effect of $|Q|$ (DB) (d) Effect of $|Q|$ (OB)

(e) Effect of $|Q|$ (YE) (f) Effect of $|Q|$ (HI) (g) Effect of $|Q|$ (SO) (h) Effect of $|Q|$ (WK)

Figure 10: Effect of $|Q|$ on search algorithms

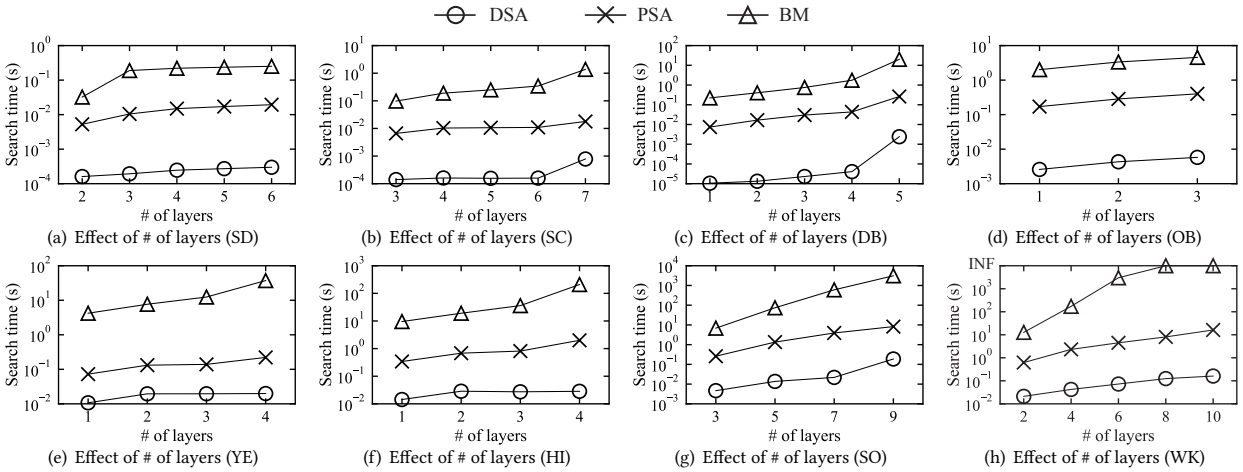


Figure 11: Effect of # of layers on search algorithms

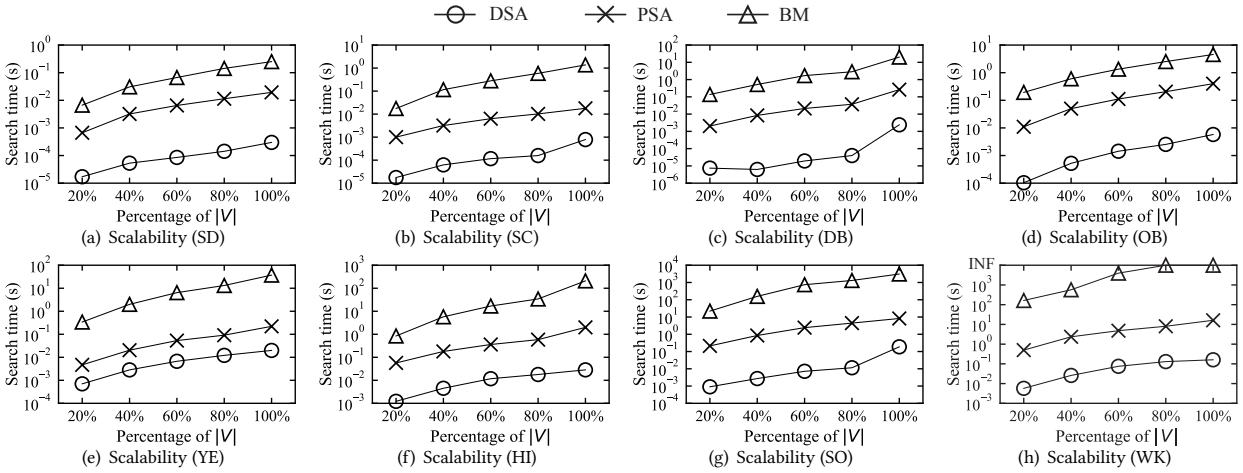


Figure 12: Scalability on search algorithms

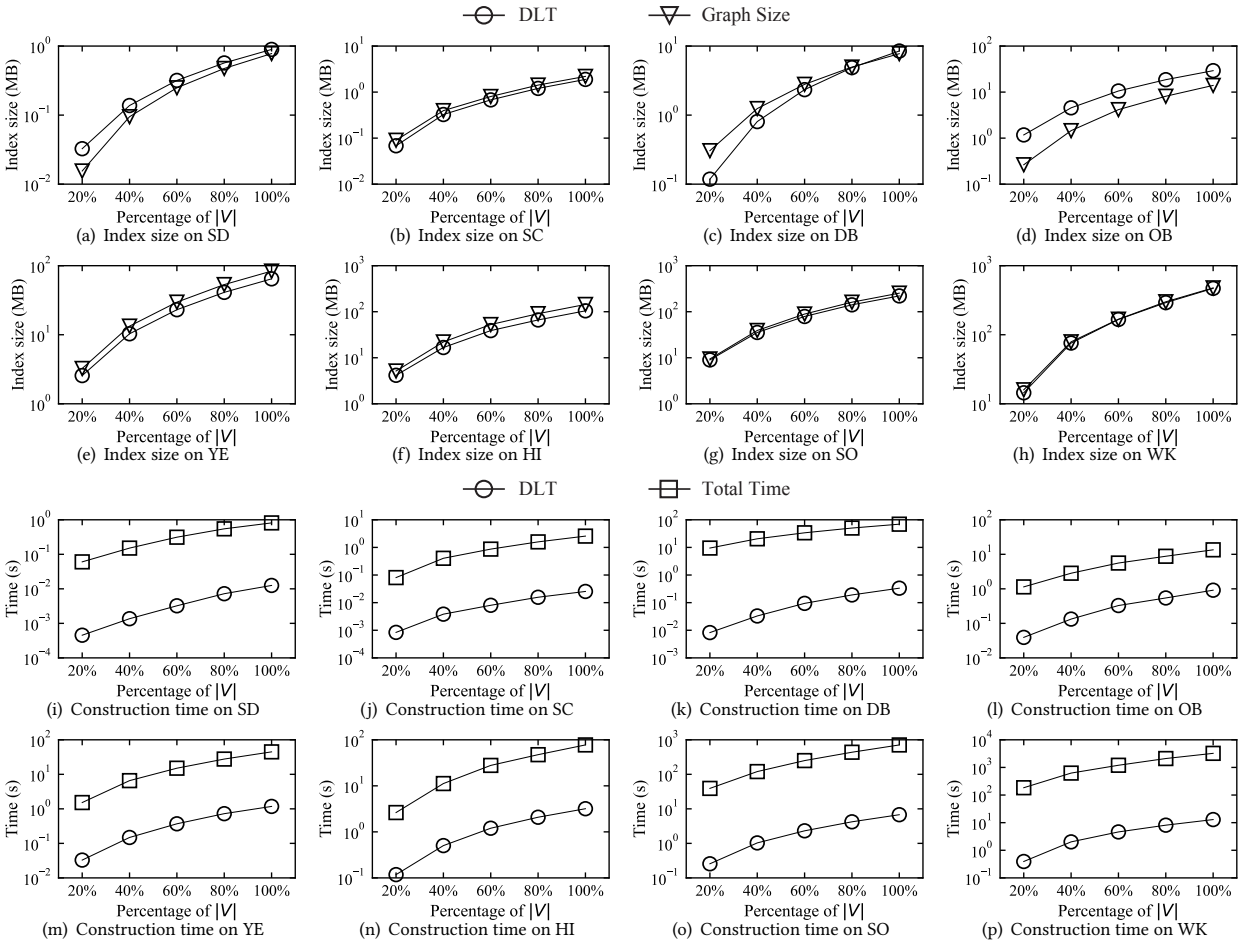


Figure 13: Scalability on DLT

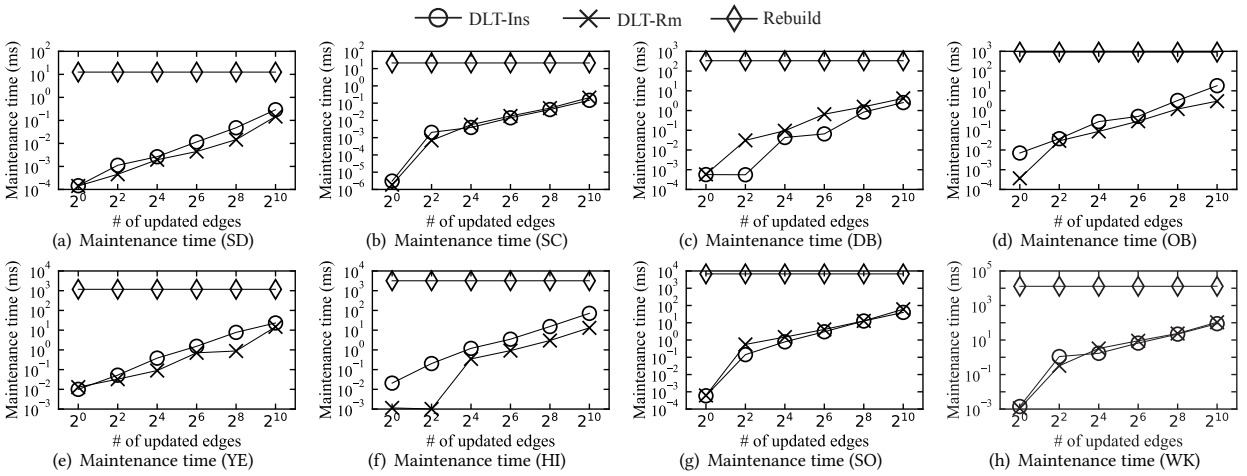


Figure 14: DLT maintenance

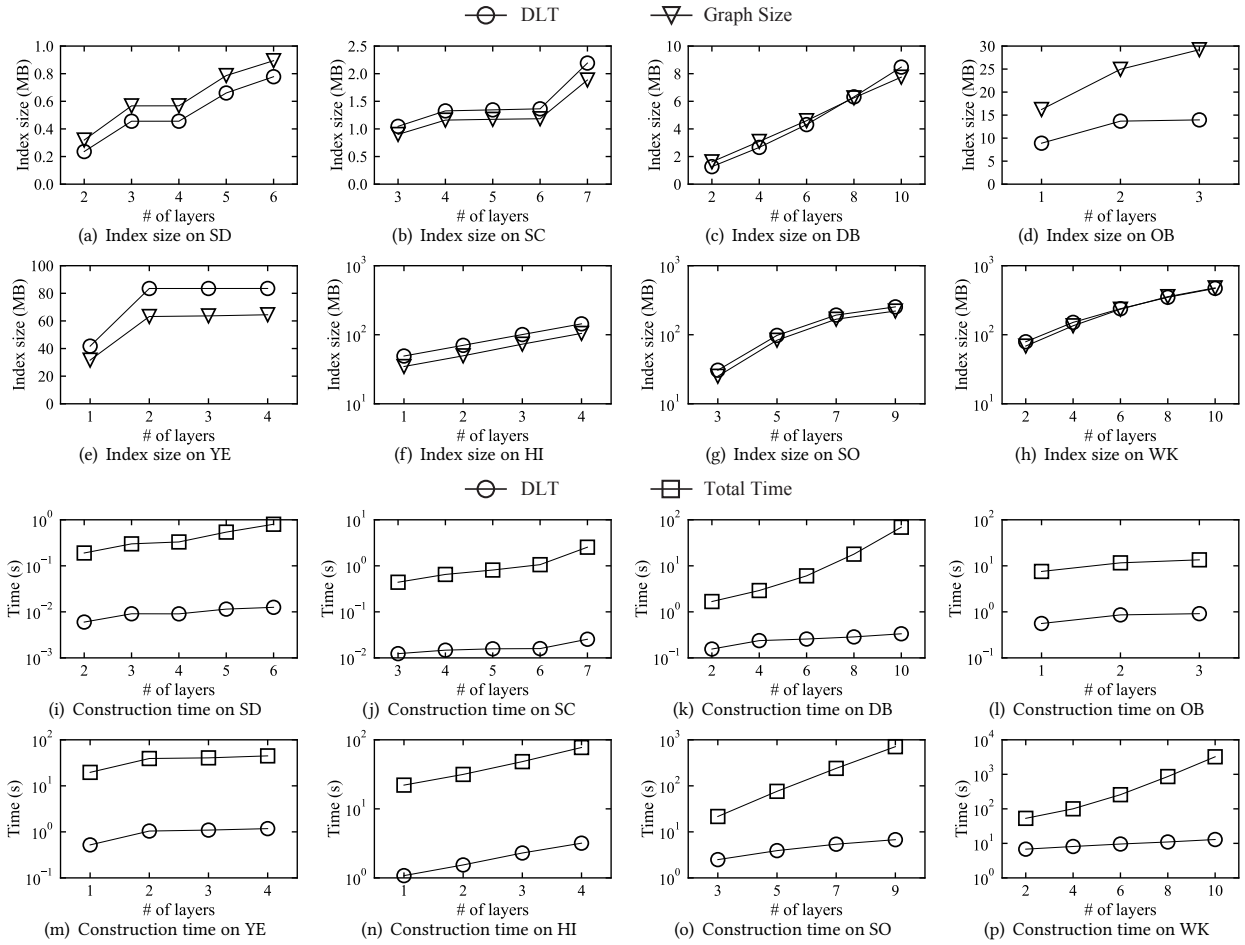


Figure 15: Effect of # of layers on DLT