CoreCube: Core Decomposition in Multilayer Graphs

Boge Liu[1], Fan Zhang[2(✉)], Chen Zhang[1], Wenjie Zhang[1], and Xuemin Lin[1]

[1] University of New South Wales, Sydney, Australia
boge.liu@unsw.edu.au, {Chenz,zhangw,lxue}@cse.unsw.edu.au
[2] Guangzhou University, Guangzhou, China
fanzhang.cs@gmail.com

Abstract. Many real-life complex networks are modelled as multilayer graphs where each layer records a certain kind of interaction among entities. Despite the powerful modelling functionality, the decomposition on multilayer graphs remains unclear and inefficient. As a well-studied graph decomposition, core decomposition is efficient on a single layer graph with a variety of applications on social networks, biology, finance and so on. Nevertheless, core decomposition on multilayer graphs is much more challenging due to the various combinations of layers. In this paper, we propose efficient algorithms to compute the CoreCube which records the core decomposition on every combination of layers. We also devise a hybrid storage method that achieves a superior trade-off between the size of CoreCube and the query time. Extensive experiments on 8 real-life datasets demonstrate our algorithms are effective and efficient.

Keywords: Core decomposition · Multilayer graph · Graph processing

1 Introduction

In real-life networks, there are usually multiple types of interactions (edges) among entities (vertices), e.g., the relationship between two users in a social network can be friends, colleagues, relatives and so on. The entities and interactions are usually modelled as a multilayer graph, where each layer records a certain type of interaction among entities [9]. Because of the strong modeling paradigm to handle various interactions among a set of entities, there are significant existing studies of multilayer graphs, e.g., [6,14]. Previous works usually focus on mining dense structures from multilayer graphs according to given parameters, e.g., [29]. Nevertheless, graph decomposition, as a fundamental graph problem [22], remains largely unexplored on multilayer graphs.

Core decomposition (or k-core decomposition), as one of the most well-studied graph decomposition, is to compute the core number for every vertex in the graph [20]. It is a powerful tool in modeling the dynamic of user engagement in social networks. In practice, a user u tends to adopt a new behavior if there are a considerable number of friends (e.g., the core number of u) in the group who also adopted the same behavior [18]. Core decomposition is also

R. Cheng et al. (Eds.): WISE 2019, LNCS 11881, pp. 694–710, 2019.
https://doi.org/10.1007/978-3-030-34223-4_44

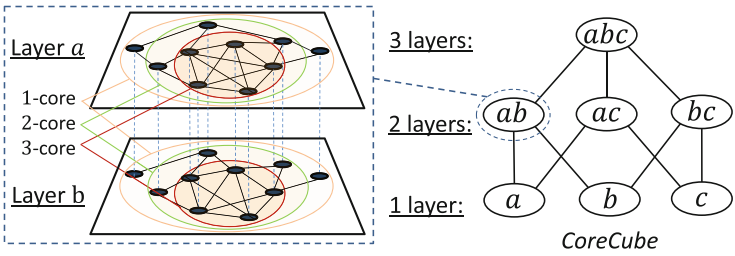


Fig. 1. Multilayer core decomposition and CoreCube of a graph

theoretically supported by Nash equilibrium in game theory [5]. It has a variety of applications, e.g., graph visualization [3], internet topology [7] and user engagement [24,26]. Extending the single-layer core decomposition to multilayer graphs is a critical task which can benefit a lot of applications considering the various real-world interactions between entities.

Given a multilayer graph, the multilayer k-core on a set of layers is defined as a set of vertices whose minimum degree in the induced subgraph of each layer is at least k. The core number of a vertex on a set of layers is the largest k such that the multilayer k-core on these layers contains the vertex. Multilayer core decomposition on a set of layers is to compute the core number for each vertex on these layers. In this paper, we propose CoreCube which records the core numbers of each vertex for every combination of layers in a multilayer graph. In the following, we show the details for some application examples.

User Engagement Evaluation. In social networks, users may participate in multiple groups with different themes, where each group forms a layer in the multilayer graph. For instance, the authors in a coauthor network have different coauthor relationship on different venues (conferences or journals). For any given user-interested combination of venues (correspond to layers), CoreCube of the coauthor network can immediately answer the engagement level for each author, i,e, the core numbers [18]. Given a degree constraint k, we can also immediately retrieve a cohesive user group from CoreCube, i.e., the multilayer k-core.

Biological Module Analysis. In biological networks, different interactions between the modules are detected with different methods due to data noise and technical limitations [11]. Analyzing module structure according to single method, i.e., on a single layer, may not be accurate. CoreCube allows us to study the connections between modules for any combination of potential methods. Thus, we can find co-expression clusters and verify the effectiveness of detection methods.

Figure 1 shows an example of CoreCube on a graph G with three layers and depicts the multilayer core decomposition on layer a and b. The 3-core on layer a and b contains 5 vertices where each vertex has a degree of at least 3 in each layer. There are 7 different combinations of layers in CoreCube of G. For each combination, we compute its multilayer core decomposition and record the core numbers in CoreCube. CoreCube can immediately answer a query for core numbers on any set of layers including the traditional single layer graph.

Table 1. Summary of notations

Notation	Definition
$G = (V, E, L)$	A multilayer graph, where V is a set of vertices, L is a set of layers, and $E \subseteq (V \times V \times L)$ is a set of edges
$V(G)$	The vertex set of G
L'; l	$L' \subseteq L$ is a subset of L; $l \in L$ is a layer in L
$E_{L'}$	The edge set in L', i.e., $E_{L'} = E \cap (V \times V \times L')$
u, v	A vertex in the graph
$\|V\|$, $\|E\|$, $\|L\|$	The number of vertices, edges, and layers in G, respectively
$N_G(v, l)$	The set of adjacent vertices of v in layer l of G
$deg_G(v, l)$	The number of adjacent vertices of v in layer l of G
d_{max}	The maximum degree, i.e., $d_{max} = \max\{deg_G(v, l) \mid v \in V \wedge l \in L\}$
$C^k_{L'}$	The multilayer k-core on a set of layers L'
$\mathcal{C}_{L'}(v)$	The core number of v on a set of layers L'
$\mathcal{C}_{L'}$	The multilayer core decomposition result on a set of layers L'
$\mathcal{C}$	The CoreCube of G, i.e., $\mathcal{C} = \{\mathcal{C}_{L'} \mid L' \subseteq L\}$

Challenges and Contributions. Although core decomposition on a single-layer graph can be computed in linear time, it becomes very challenging on a multilayer graph because the combination number of layers is exponential to the number of layers. In the general case, no polynomial-time algorithm may exist for computing the CoreCube. To the best of our knowledge, there is only one similar work [10] where the algorithms can be adapted to compute the CoreCube while it is hard to share the computation among different combination of layers. The algorithms proposed in this paper can largely speed up the computation of CoreCube. We summarize our contributions as follows:

- We propose efficient algorithms to compute the CoreCube. Several theorems reveal the inner characteristics of multilayer core decomposition. (Sect. 3)
- We devise a hybrid storage method which has a superior trade-off between query processing time and storage size. (Sect. 4)
- Extensive experiments demonstrate that our CoreCube computation and query processing are faster than baselines by more than one order of magnitude. (Sect. 5)

2 Problem Definition

In this section, we give some notations and formally define CoreCube. The notations are summarized in Table 1.

We consider an unweighted and undirected multilayer graph $G = (V, E, L)$, where V represents the set of vertices in G, L represents the set of layers, and $E \subseteq (V \times V \times L)$ represents the set of edges. We use $|V|$, $|E|$, and $|L|$ to denote the

number of vertices, edges, and layers, respectively. $N_G(v, l)$ is the set of adjacent vertices of v in layer l. We say a vertex u is incident to an edge, or an edge is incident to u, if u is one of the endpoints of the edge. We use $deg_G(v, l)$ to denote the number of adjacent vertices of u in layer l. When the context is clear, we omit the input graph in notations, such as $deg(v, l)$ for $deg_G(v, l)$.

Definition 1. ***Multilayer k-core.*** *Given a multilayer graph $G = (V, E, L)$, a set of layers $L' \subseteq L$ and an integer k, the multilayer k-core of G on L', denoted by $C^k_{L'}$, is the maximum vertex set such that every vertex v in the subgraph H induced by $C^k_{L'}$ satisfies $deg_H(v, l) \geq k$ on each $l \in L'$.*

Let k_{max} be the maximum possible k such that a multilayer k-core of G on L' exists. The multilayer k-core for all $1 \leq k < k_{max}$ has the following partial containment property:

Property 1. Given a multilayer graph $G = (V, E, L)$ and a set of layers L', $C^{k+1}_{L'} \subseteq C^k_{L'}$ for all $1 \leq k < k_{max}$.

Next, we define the core number for each $v \in V$.

Definition 2. ***Core Number.*** *Given a multilayer graph $G = (V, E, L)$ and a set of layers $L' \subseteq L$, the core number of v on L', denoted by $\mathcal{C}_{L'}(v)$, is the largest k such that v is contained in multilayer k-core on L', i.e., $\mathcal{C}_{L'}(v) = \max\{k \mid v \in C^k_{L'}\}$.*

Based on Property 1 and Definition 2, we can easily derive following lemma:

Lemma 1. *Given a multilayer graph $G = (V, E, L)$, a set of layers L', and an integer k, we have $C^k_{L'} = \{v \in V \mid \mathcal{C}_{L'}(v) \geq k\}$.*

Definition 3. ***Multilayer Core Decomposition.*** *Given a multilayer graph $G = (V, E, L)$ and a set of layers $L' \subseteq L$, the multilayer core decomposition, denoted by $\mathcal{C}_{L'}$, computes $C^k_{L'}$ for all $1 \leq k \leq k_{max}$.*

According to Lemma 1, multilayer core decomposition on L' is equivalent to computing the core number $\mathcal{C}_{L'}(v)$ for each $v \in V$. Finally, we give the formal definition of CoreCube and the problem we tackle in this paper.

Definition 4. ***CoreCube.*** *Given a multilayer graph $G = (V, E, L)$, the CoreCube of G, denoted as $\mathcal{C}$, computes multilayer core decomposition on all the subsets of L, i.e., $\mathcal{C} = \{\mathcal{C}_{L'} \mid L' \subseteq L\}$.*

Problem Statement. In this paper, we study the problem of efficiently computing and compactly storing CoreCube of multilayer graphs.

3 CoreCube Computation

In this section, we present our basic CoreCube computation algorithm and then discuss how to improve the algorithm by sharing computation among multilayer core decomposition on different sets of layers.

3.1 Basic CoreCube Algorithm

Based on Property 1, given a multilayer graph $G = (V, E, L)$ and a set of layers $L' \subseteq L$, the multilayer core decomposition on L' can be computed in a bottom up manner following the paradigm used for single layer graphs [4], which increases k step by step and iteratively removing vertices whose degree are less than k. We give this algorithm `Core-BU` in Algorithm 1. `Core-BU` computes multilayer core decomposition in increasing order of k. Each time, k is selected as the minimum degree (line 3). Whenever there exists a vertex v whose degree is no larger than k in some layer $l \in L'$ (line 4), we know that the core number of v is k (line 5) and we remove v with all its incident edges from the graph (line 6). The core numbers are returned in line 7. With the help of bin sort and the efficient data structure proposed in [13] to maintain the minimum degree, `Core-BU` can achieve a time complexity of $O(|E_{L'}| + |V|)$.

The algorithm `CoreCube-BU` which computes CoreCube with `Core-BU` is shown in Algorithm 2. In Algorithm 2, CoreCube is computed level-by-level. Each time, we generate all the subsets of L with the same size z (line 3) and compute multilayer core decomposition on each subset (line 4–5). CoreCube is returned in line 6.

Algorithm 1: **Core-BU**(G, L')

Input : $G = (V, E, L)$: a multilayer graph, L' : a subset of L
Output : $\mathcal{C}_{L'}$: the multilayer core decomposition on L'

1 $G' \leftarrow G_{L'}$;
2 **while** $G' \neq \emptyset$ **do**
3 $\quad k \leftarrow \min\{deg_{G'}(v, l) \mid v \in V(G') \wedge l \in L'\}$;
4 $\quad$ **while** $\exists v \in V(G')$ and $l \in L' : deg_{G'}(v, l) \leq k$ **do**
5 $\quad\quad \mathcal{C}_{L'}(v) \leftarrow k$;
6 $\quad\quad$ remove v and its incident edges from G';
7 **return** $\mathcal{C}_{L'}$

Algorithm 2: **CoreCube-BU**(G)

Input : G : a multilayer graph
Output : $\mathcal{C}$: the CoreCube of G

1 $\mathcal{C} \leftarrow \emptyset$;
2 **for** $z = 1$ to $|L|$ **do**
3 $\quad Z \leftarrow \{$all the subsets of L whose size are $z\}$;
4 $\quad$ **for each** $L' \in Z$ **do**
5 $\quad\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{$Core-BU$(G, L')\}$;
6 **return** $\mathcal{C}$

Complexity. Since there are $2^{|L|} - 1$ (expect $\emptyset$) subsets of L need to be processed and `Core-BU` runs in $O(|E_{L'}| + |V|)$ for any subset L', the complexity of `CoreCube-BU` is $O(2^{|L|} \cdot (|E| + |V|))$.

3.2 Computation-Sharing CoreCube Algorithm

`Core-BU` needs to remove all the edges in $E_{L'}$ when computing multilayer core decomposition on L'. This is because the core number of a vertex v is obtained only when v is removed. Therefore, `CoreCube-BU` computes each multilayer core decomposition independently. To improve the efficiency of CoreCube computation, we aim at devising an algorithm that shares computation among multilayer core decomposition on different sets of layers. We first extend the locality property of k-core in single layer graphs [19] to multilayer graphs.

Theorem 1. *Given a multilayer graph $G = (V, E, L)$ and a set of layers $L' \subseteq L$, we have the following recursive equations for core number $\mathcal{C}_{L'}(v)$ of a vertex $v \in V$:*

$$\forall l \in L' \quad M_l(v) = \max\ k\ s.t.\ |\{u \in N(v,l) \mid \mathcal{C}_{L'}(u) \geq k\}| \geq k \tag{1}$$

$$\mathcal{C}_{L'}(v) = \min\{M_l(v) \mid l \in L'\} \tag{2}$$

where $N(v,l)$ is the set of adjacent vertices of v in layer l.

Proof. (i) Let $k_c = \min\{M_l(v) \mid l \in L'\}$ and S be the multilayer k_c-core on L'. Firstly, S must be nonempty as there exists some vertex u satisfying $\mathcal{C}_{L'}(u) \geq k_c$. According to Eqs. 1 and 2, we have $\forall l \in L'$, $|\{u \in N(v,l) \mid \mathcal{C}_{L'}(u) \geq k_c\}| \geq k_c$. Therefore, in each layer $l \in L'$, v has at least k_c adjacent vertices in S, which means $v \in S$. Hence, $\mathcal{C}_{L'}(v) \geq k_c$. (ii) On the other hand, according to Eqs. 1 and 2, there must exist some $l_0 \in L'$ in which $|\{u \in N(v,l_0)|\mathcal{C}_{L'}(u) \geq k_c{+}1\}| < k_c{+}1$. Therefore, $\mathcal{C}_{L'}(v) < k_c + 1$. Combining the conclusion in (i) and (ii) together, it holds that $\mathcal{C}_{L'}(v) = \min\{M_l(v) \mid l \in L'\}$.

Following Theorem 1, we devise the algorithm `Core-TD` which computes multilayer core decomposition on L' in a top down manner. `Core-TD` iteratively reduces the upper bound of core number for each vertex. Initially, each vertex v is assigned an arbitrary upper bound of core number (e.g. the minimum degree of v in L'). Then `Core-TD` keeps updating the upper bound using Eqs. 1 and 2 until convergence. The pseudocode of `Core-TD` is given in Algorithm 3. Here, we use $\overline{\mathcal{C}}_{L'}(v)$ to denote the upper bound of $\mathcal{C}_{L'}(v)$. We also use $sup(v,l)$ (support of v) to denote the number of adjacent vertices of v in layer l whose upper bound is no less than $\overline{\mathcal{C}}_{L'}(v)$. That is

$$sup(v,l) = |\{u \in N(v,l) \mid \overline{\mathcal{C}}_{L'}(u) \geq \overline{\mathcal{C}}_{L'}(v)\}| \tag{3}$$

Note that if $sup(v,l) < \overline{\mathcal{C}}_{L'}(v)$, Eq. 1 does not hold for v in layer l. Therefore, we can determine whether $\overline{\mathcal{C}}_{L'}(v)$ needs to be updated by comparing $\overline{\mathcal{C}}_{L'}(v)$ with $sup(v,l)$ for each $l \in L'$ instead of scanning all the adjacent vertices of v.

`Core-TD` first initializes $sup(v, l)$ for every vertex based on Eq. 3 in line 1. Then it updates vertex v whose upper bound violates Eq. 1 in some layer r (line 2). c_0 records the value of $\overline{\mathcal{C}}_{L'}(v)$ before being updated (line 3). `Core-TD` updates $\overline{\mathcal{C}}_{L'}(v)$ according to Eqs. 1 and 2 (line 4–7). Then, for each layer $l \in L'$, it recomputes $sup(v, l)$ and updates $sup(u, l)$ for each adjacent vertex u of v (line 8–12). $sup(u, l)$ is decreased by 1 if v once contributed to $sup(u, l)$ but not anymore after $\overline{\mathcal{C}}_{L'}(v)$ being updated (line 11–12). Finally, after all the upper bound converges, `Core-TD` sets $\mathcal{C}_{L'}(v)$ as $\overline{\mathcal{C}}_{L'}(v)$ for each vertex $v \in V$ in line 13 and returns $\mathcal{C}_{L'}$ in line 14.

Complexity. In `Core-TD`, each time when the upper bound of some vertex v is updated, line 2–12 takes $O(\sum_{l \in L'}(deg(v, l)))$. Since $\overline{\mathcal{C}}_{L'}(v)$ is at least decreased by 1 whenever being updated, the time complexity of `Core-TD` is $O(\sum_{v \in V}(\overline{\mathcal{C}}_{L'}(v) \cdot \sum_{l \in L'} deg(v, l)))$, which is bounded by $O(d_{max} \cdot |E_{L'}|)$ as the maximum degree d_{max} can always serve as an upper bound for any vertex.

Algorithm 3: **Core-TD**(G, L', $\overline{\mathcal{C}}_{L'}$)

Input : $G = (V, E, L)$: a multilayer graph, L' : a subset of L, $\overline{\mathcal{C}}_{L'}$: upper bound of core number on L' for each vertex in V
Output : $\mathcal{C}_{L'}$: the multilayer core decomposition

1 $sup(v, l) \leftarrow |\{u \in N(v, l) \mid \overline{\mathcal{C}}_{L'}(u) \geq \overline{\mathcal{C}}_{L'}(v)\}|$ for each $v \in V$ and $l \in L'$;
2 **while** $\exists v \in V'$ and $r \in L'$: $sup(v, r) < \overline{\mathcal{C}}_{L'}(v)$ **do**
3 $c_0 \leftarrow \overline{\mathcal{C}}_{L'}(v)$;
4 **for each** $l \in L'$ **do**
5 $M_l(v) = \max\ k\ s.t.\ |\{u \in N(v, l) \mid \overline{\mathcal{C}}_{L'}(u) \geq k\}| \geq k$;
6 **if** $\overline{\mathcal{C}}_{L'}(v) > M_l(v)$ **then**
7 $\overline{\mathcal{C}}_{L'}(v) \leftarrow M_l(v)$;
8 **for each** $l \in L'$ **do**
9 $sup(v, l) \leftarrow |\{u \in N(v, l) \mid \overline{\mathcal{C}}_{L'}(u) \geq \overline{\mathcal{C}}_{L'}(v)\}|$;
10 **for each** $u \in N(v, l)$ **do**
11 **if** $\overline{\mathcal{C}}_{L'}(u) \leq c_0$ and $\overline{\mathcal{C}}_{L'}(u) > \overline{\mathcal{C}}_{L'}(v)$ **then**
12 $sup(u, l) \leftarrow sup(u, l) - 1$;
13 $\mathcal{C}_{L'}(v) \leftarrow \overline{\mathcal{C}}_{L'}(v)$ for every $v \in V$;
14 **return** $\mathcal{C}_{L'}$

Correctness. The correctness of `Core-TD` is based on Theorem 1. When `Core-TD` terminates, Eqs. 1 and 2 are satisfied for each vertex. On the other hand, the value computed for each vertex cannot be smaller than the core number because it is always an upper bound of the core number. Hence, `Core-TD` correctly computes core number for each vertex.

The key issue with `Core-TD` is how to initialize the upper bound tight enough such that it can quickly converge. To deal with this issue, we introduce the following lemma:

Algorithm 4: **CoreCube-TD**(G)

```
   Input  : G : a multilayer graph
   Output : C : the CoreCube of G
1  C ← ∅;
2  for z = 1 to |L| do
3      Z ← {all the subsets of |L| whose size are z};
4      for each L' ∈ Z do
5          for each v ∈ V do
6              if z = 1 then
7                  C̄_{L'}(v) ← deg(v, l) where l ∈ L';
8              else
9                  C̄_{L'}(v) ← min{C_D(v) | D ⊂ L' ∧ |D| = |L'| + 1};
10         C ← C ∪ {Core-TD (G, L', C̄_{L'})};
11 return C
```

Lemma 2. *Given a multilayer graph $G = (V, E, L)$ and a vertex $v \in V$, it holds that $\mathcal{C}_{L_1}(v) \geq \mathcal{C}_{L_2}(v)$ if $L_1 \subseteq L_2$.*

Proof. Let $k = \mathcal{C}_{L_2}(v)$. Based on the definition of core number, there exists a set of vertices $S \subseteq V$ such that each vertex v in the subgraph H induced by S satisfies $deg_H(v, l) \geq k$ for $l \in L_2$. Since $L_1 \subseteq L_2$, we have $\mathcal{C}_{L_1}(v) \geq k = \mathcal{C}_{L_2}(v)$.

According to Lemma 2, the core number of a vertex v on L' can serve as an upper bound of v's core number on any superset of L'. Note that if we compute CoreCube level-by-level, we will obtain core numbers on all the subsets of L' when computing multilayer core decomposition on L'. Therefore we can exploit previous computation as much as possible by initializing $\overline{\mathcal{C}}_{L'}(v)$ with the minimum core number of v on all the subsets of L', i.e., $\overline{\mathcal{C}}_L(v) = \min\{\mathcal{C}_P(v) | P \subset L'\}$. Furthermore, based on Lemma 2, we actually only need to consider the subsets whose size is only one smaller than $|L'|$ because any the subset of L' whose size is smaller than $|L'| - 1$ must be contained in some subset of L' whose size is $|L'| - 1$.

The algorithm `CoreCube-TD` which computes CoreCube with `Core-TD` is shown in Algorithm 4. Each time before it invokes `Core-TD` for a set of layers L', it sets the upper bound of core number for each vertex according to Lemma 2 (line 9). If $|L'|$ is 1, it sets the upper bound as the vertex degree (line 7). Finally, the CoreCube of G is returned in line 11.

Complexity. In `CoreCube-TD`, since the number of subsets D processed in line 9 is $|L'|$, line 5–9 takes $O(|L'| \cdot |V|)$. Considering that there are $2^{|L|} - 1$ subsets of L and `Core-TD` is invoked for each subset, the time complexity of `CoreCube-TD` is bounded by $O(2^{|L|} \cdot (|L| \cdot |V| + d_{max} \cdot |E|))$. Though the time complexity is apparently worse than that of `CoreCube-BU`, we find that much less vertices are visited in our experiments, especially when the number of layers is large. This is because the upper bound is initialized very close to the core number and converges quickly in `Core-TD`.

4 CoreCube Storage

In this section, we devise a method for compactly storing CoreCube and discuss how to process queries for core numbers on any set of layers. A straightforward method is storing core numbers on each set of layers in separate files. Given a core number query, we can directly retrieve the result from the disk. However, this method requires large disk space as we need to store each vertex in every file. To reduce space usage, we propose two optimization strategies.

Firstly, many vertices' core number on a set of layers L' can be zero when $|L'|$ is large because the core number of a vertex v is zero if $deg(v, l)$ in some layer $l \in L'$ equals to 0. Therefore, we do not record the vertex whose core number is zero. Secondly, the core number on L' can remain unchanged when a new layer l is added to L' if the core number on L' is small or the distribution of core number on l is nearly the same as that in L'. Hence, we can store the difference between core numbers on different sets of layers instead of directly storing core number for each vertex. Here, we call the file that stores nonzero core numbers as absolute storage and the file that stores the difference as relative storage. The algorithm `Hybrid-Storage` which uses both absolute storage and relative storage is given in Algorithm 5.

`Hybrid-Storage` creates a file F for each subset of L (line 3). For the subset consists of single layer, it uses absolute storage to store the nonzero core number for each vertex (line 4–6). For other subsets L', it first counts the number of

Algorithm 5: **Hybrid-Storage**(G, $\mathcal{C}$)

Input : $G = (V, E, L)$: a multilayer graph, $\mathcal{C}$: the CoreCube of G
Output : the files that stores $\mathcal{C}$

1 $Z \leftarrow$ {all the subsets of $|L|$};
2 **for each** $L' \in Z$ **do**
3 create a new file F;
4 **if** $|L'| = 1$ **then**
5 **for each** $v \in V$ **and** $\mathcal{C}_{L'}(v) \neq 0$ **do**
6 write v and $\mathcal{C}_{L'}(v)$ into F;
7 **else**
8 $n_1 \leftarrow$ the number of non zero values in $\mathcal{C}_{L'}$;
9 $P \leftarrow$ the subset of L' s.t. $|\{v \in V \mid \mathcal{C}_P(v) \neq \mathcal{C}_{L'}(v)\}|$ is minimum $\wedge |P| = |L'| - 1$;
10 $n_2 \leftarrow |\{v \in V \mid \mathcal{C}_P(v) \neq \mathcal{C}_{L'}(v)\}|$;
11 **if** $n_1 \leq n_2$ **then**
12 **for each** $v \in V$ **and** $\mathcal{C}_{L'}(v) \neq 0$ **do**
13 write v and $\mathcal{C}_{L'}(v)$ into F;
14 **else**
15 write P as the predecessor into F;
16 **for each** $v \in V$ **and** $\mathcal{C}_P(v) - \mathcal{C}_{L'}(v) \neq 0$ **do**
17 write v and $\mathcal{C}_P(v) - \mathcal{C}_{L'}(v)$ into F;

Algorithm 6: **Core-Retrieve**(G, L')

Input : $G = (V, E, L)$: a multilayer graph, L': a subset of layers
Output : $\mathcal{C}_{L'}$: the multilayer core number on L'

```
1  C_{L'}(v) ← 0 for each v ∈ V;
2  flag ← true; P ← L';
3  while flag do
4      load the file F corresponding to P from disk;
5      if F is relative storage then
6          P ← the predecessor in F;
7      else
8          flag ← false;
9      C_{L'}(v) ← C_{L'}(v) + F(v) for each v ∈ V;
10 return C_{L'}
```

nonzero core number in $\mathcal{C}_{L'}$ as n_1 (line 8). Then, it finds the subset P of L' such that the number of different values between $\mathcal{C}_{L'}$ and $\mathcal{C}_P$ is minimum (line 9) and refers this number as n_2 (line 10). If $n_1 \leq n_2$, `Hybrid-Storage` uses absolute storage (line 11–13). Otherwise, it uses relative storage that stores all the difference between $\mathcal{C}_{L'}$ and $\mathcal{C}_P$ (line 16–17). It also records P as the predecessor (line 15) so that we can know from which subset the difference is made when answering queries.

The algorithm which processes queries for core numbers on a set of layers L' is shown in Algorithm 6. `Core-Retrieve` keeps loading files from disk according to the predecessors (line 4–6) until it meets absolute storage (line 7–8). Meanwhile, `Core-Retrieve` computes core numbers by summing up the difference stored in each file (line 9). Note that we use $F(v)$ to represent the value (core number or difference) associated with node v stored in file F. Finally, core numbers are returned in line 10. Note that `Core-Retrieve` loads at most $|L'|$ files.

Table 2. Statistics of datasets

Dataset	Vertices	Edges	Layers	Domain
`Homo`	18,223	153,922	7	Genetic
`SacchCere`	6,571	247,152	7	Genetic
`Twitter`	2,281,260	3,827,964	3	Social
`Amazon`	410,237	8,132,506	4	Co-purchasing
`DBLP`	2,175,466	8,221,193	10	Co-authorship
`Flickr`	2,302,927	23,350,524	10	Social
`StackOverflow`	6,024,272	28,978,914	10	Social
`Wiki`	25,323,885	132,693,853	10	Hyperlinks

5 Experimental Evaluation

5.1 Experimental Setting

Datasets. Eight real-life networks were deployed in our experiments. Table 2 shows the statistics of the 8 datasets, listed in increasing order of their edge numbers. `Home` and `SacchCere` are networks describing different types of genetic interactions between genes. `Twitter` represents different types of social interaction among Twitter users. `Amazon` is a co-purchasing temporal network, containing four snapshots between March and June 2003. `DBLP` is a co-author network. `Flickr` is a social network represents Flickr users and their friendship connections. `StackOverflow` is a temporal network represents different types of interactions on the website Stack Overflow. `Wiki` contains users and pages from Wikipedia, connected by edit events.

Algorithms. We test 4 algorithms for CoreCube computation. `CoreCube-BU` and `CoreCube-TD` are our algorithms, e.g., Algorithms 2 and 4.

`ML-DFS` and `ML-Hybrid` are two state-of-the-art existing solutions proposed in [10]. They compute cores for all the coreness vector $\boldsymbol{k}$, where $\boldsymbol{k}$ is a $|L|$-dimension vector and the value k in each dimension represents that the degree of each vertex is no less than k in the corresponding layer. `ML-DFS` searches the space of $\boldsymbol{k}$ through depth-first search strategy. `ML-Hybrid` adopts both depth-first and breath-first search strategy. In our experiments, we compute CoreCube by using cores whose $\boldsymbol{k}$ has the same value in every nonzero dimension. For the sake of fairness, *we extract and report the time spent on computing these cores* in `ML-DFS` and `ML-Hybrid` instead of the total running time.

To the best of our knowledge, no existing work investigates the storage of CoreCube. We test three algorithms `Naive-Storage`, `Nonzero-Storage` and `Hybrid-Storage`. `Naive-Storage` stores core numbers without any optimization strategies. `Nonzero-Storage` only stores nonzero core numbers. `Hybrid-Storage` uses both absolute storage and relative storage, i.e., Algorithm 5.

`Core-Retrieve` is our algorithm for answering core number queries, i.e., Algorithm 6. `CoreScratch` computes core numbers from scratch for each query. We divide `CoreScratch` into two procedures, `CoreScratch-Load` and `CoreScratch-Comp`. `CoreScratch-Load` is the procedure that loads the graph from disk into main memory. `CoreScratch-Comp` is the procedure that computes core numbers. For `CoreScratch-Comp`, we test both `Core-BU` and `Core-TD`, and report the running time based on the faster one.

All algorithms are implemented in C++ with -O2 optimization level and tested on an server equipped with Intel Xeon CPU at 2.8 GHz and 128 GB main memory.

5.2 CoreCube Computation

In this set of experiments, we set the maximum running time for each test as 48 h. If an algorithm cannot stop within the time limit, we omit its running time.

Exp-1: CoreCube Computation Time on Different Datasets. We report the time cost for computing CoreCube on different datasets in Fig. 2. As shown

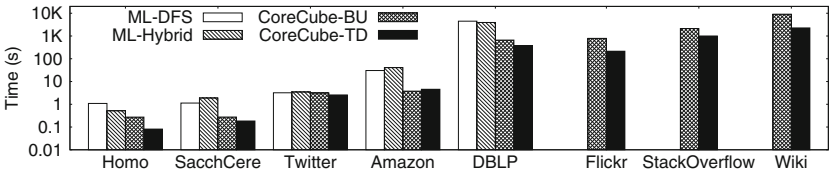


Fig. 2. CoreCube computation time in all datasets

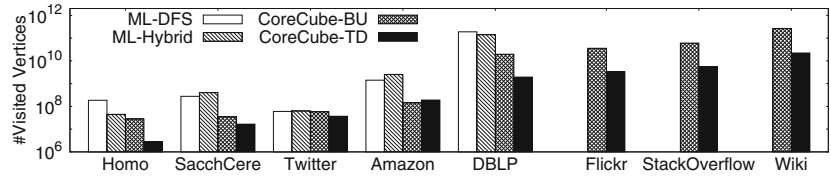


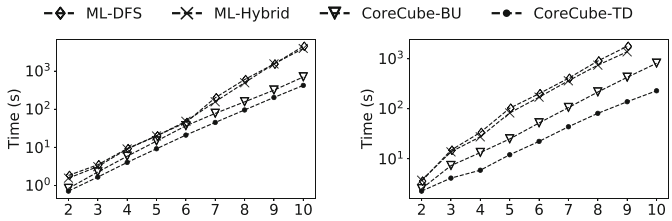
Fig. 3. The number of visited vertices in CoreCube computation in all datasets

in Fig. 2, our proposed algorithm `CoreCube-TD` is the fastest algorithm in all datasets except `Amazon` and achieves one order of magnitude improvement on average compared with existing solutions `ML-DFS` and `ML-Hybrid`. For example, in `DBLP`, `CoreCube-BU` and `CoreCube-TD` spend 662s and 375s respectively while `ML-DFS` and `ML-Hybrid` spend 4487 s and 3932 s respectively. In the three largest datasets, `ML-DFS` and `ML-Hybrid` cannot terminate within 48 h.

Exp-2: The Number of Visited Vertices in CoreCube Computation. To better demonstrate performance of the four CoreCube computation algorithms, we report the number of visited vertices in Fig. 3. The number of visited vertices represents how many times the value related to a vertex is modified or accessed, e.g., removing an edge or decreasing upper bound. For `ML-DFS` and `ML-Hybrid`, the number of visited vertices is collected during the computation of cores that are used for computing CoreCube. As shown in Fig. 3, the number of visited vertices in `CoreCube-TD` is smallest in all datasets except for `Amazon`. This is because the core numbers in `Amazon` vary a lot on different sets of layers, which leads to slow convergence in `Core-TD`. Compared with our algorithms, the number of visited vertices in `ML-DFS` and `ML-Hybrid` is much larger. The reason is that they need to generate a subgraph that contains some core before computing it.

Exp-3: Scalability of CoreCube Computation. In this experiment, we evaluate the performance of four CoreCube computation algorithms with varying the number of layers. We show results on `DBLP` and `Flickr` in Fig. 4. The trends are similar in other datasets. As shown in Fig. 4, the running time of four algorithms stably increases. The gap between existing algorithms and our proposed algorithms becomes larger as the number of layers increases. Compared with existing algorithms, our proposed algorithm `CoreCube-TD` achieves at least 1 order of

magnitude improvement when the number of layers excesses 7. Furthermore, the gap between `CoreCube-BU` and `CoreCube-TD` becomes larger with the increasing of layers, which shows that the advantages of `CoreCube-TD` is significant when the number of layers becomes large.



(a) DBLP (Vary layers) (b) Flickr (Vary layers)

Fig. 4. CoreCube computation time with varying number of layers

5.3 CoreCube Storage and Query Processing

Exp-4: Disk Usage under Different Storage Methods. In this experiment, we report the disk usage of storing CoreCube of all datasets in Fig. 5. As shown in Fig. 5, the disk usage of `Hybrid-Storage` is smallest in all datasets. For example, in `DBLP`, the disk usage of `Naive-Storage`, `Nonzero-Storage` and `Hybrid-Storage` are 21GB, 522MB and 302MB respectively. The gap between `Naive-Storage` and `Nonzero-Storage` shows that many vertices have zero core number in CoreCube. `Hybrid-Storage` further reduces disk usage by storing the difference between core numbers on different subsets of layers.

Exp-5: Core Number Query Processing Time. In this experiment, we randomly generate 100 core number queries for each dataset. Each core number query asks for core numbers on a specific set of layers. The total running time of answering the 100 queries is reported in Fig. 6. As shown in Fig. 6, `Core-Retrieve` finishes 100 queries within 10 ms in all datasets including the time spent on loading files from disk. `CoreScratch` spends more than 100 s in the largest dataset even if the graph has already been loaded into memory. In real scenarios, graphs cannot always be kept in memory. The advantage of `Core-Retrieve` is more significant when considering the graph loading time in `CoreScratch-Load`.

5.4 Case Study on DBLP

In this section, we test the effectiveness of multilayer core decomposition on DBLP. Here, the multilayer graph has two layers. One layer is the coauthor network of SIGMOD conference. Another one is the coauthor network of KDD

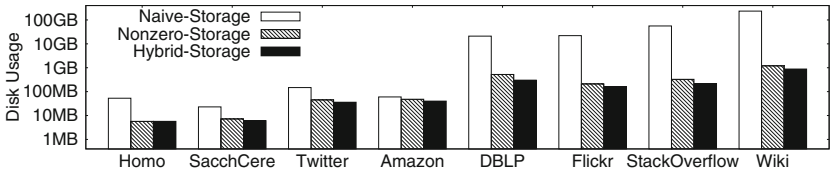


Fig. 5. CoreCube storage in all datasets

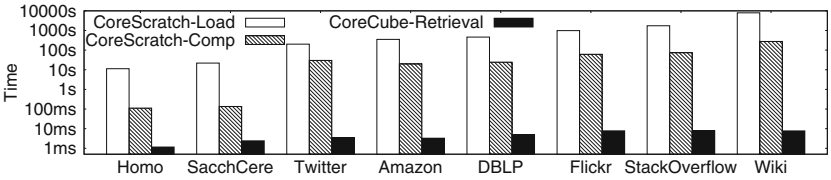


Fig. 6. Core number query processing time

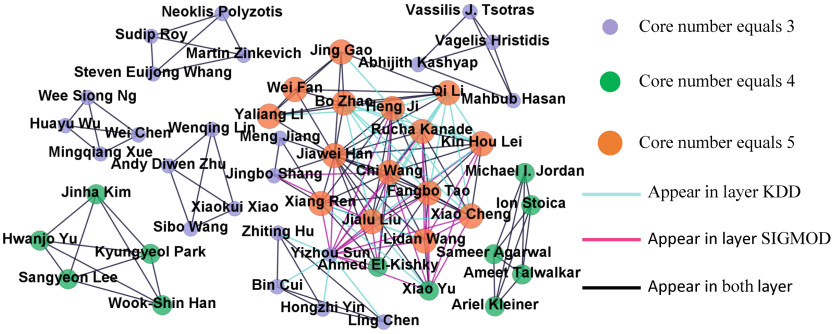


Fig. 7. Multilayer core decomposition on DBLP

conference. Two authors are connected if they collaborated on at least one paper. Both layers are extracted from data from 2013 to 2017.

Exp-6: Case Study on DBLP. We show vertices with core number no less than 3 in Fig. 7. Edges that appear exclusively in KDD and SIGMOD are colored with blue and red respectively. Edges that appear in both layers are colored with black. As shown in Fig. 7, multilayer core decomposition effectively captures authors with different engagement level in both conferences. Note that the subgraph induced by multilayer k-core are not necessarily connected.

6 Related Work

Cohesive Subgraphs. A variety of cohesive subgraph models are proposed to handle different scenarios. One of the earliest model is clique [17] where every

vertex is adjacent to every other vertex in the subgraph. The over-restrictive definition of clique leads to many relaxed models, e.g., n-clique [16], k-plex [21], and quasi-clique [1]. Cohesive subgraph models have a lot of applications on different disciplines, such as social networks [18,25,28], protein networks [2] and brain science [8]. The model of k-core [20] is well-studied on single-layer graphs for its elegant definitions and linear-time solution. Given a graph, the k-core of every input k naturally forms a hierarchical graph decomposition. Core decomposition is applied to many areas of importance, e.g., graph visualization [3], internet topology [7] and so on. A linear-time algorithm for k-core decomposition on single layer graph is proposed in [4]. Liu *et* al. [15] also studies core decomposition in bipartite graphs.

Multilayer Graphs. As a powerful paradigm to model complex networks, multilayer graphs received a lot of interests in the literature [9]. Most existing works focus on mining dense structures on multilayer networks. Zhang *et* al. [27] detect cohesive subgraphs on a 2-layer graph where one layer corresponds to user engagement and the other corresponds to user similarity. Wu *et* al. [23] find subgraphs where each subgraph is dense on one layer and connected on the other layer. Jethava and Beerenwinkel [12] study the densest common subgraph problem to find a subgraph maximizing the minimum average degree on all the layers of a graph. They propose a greedy algorithm without approximation guarantees. Zhu *et* al. [29] search diversified coherent k-cores with top sizes on multilayer graphs. Li *et* al. [14] find persistent k-cores on a temporal graph where each layer corresponds to a time span. Galimberti *et* al. [10] study core decomposition and densest subgraph extraction on multilayer graphs.

7 Conclusion

In this paper, we study core decomposition on multilayer graphs and propose the CoreCube which records the multilayer core decomposition on every combination of layers. We devise algorithms for efficiently computing and compactly storing CoreCube. The experimental results validate the efficiency of our proposed algorithms and effectiveness of multilayer core decomposition.

References

1. Abello, J., Resende, M.G.C., Sudarsky, S.: Massive quasi-clique detection. In: Rajsbaum, S. (ed.) LATIN 2002. LNCS, vol. 2286, pp. 598–612. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45995-2_51
2. Altaf-Ul-Amine, M., et al.: Prediction of protein functions based on k-cores of protein-protein interaction networks and amino acid sequences. Gen. Inf. **14**, 498–499 (2003)
3. Alvarez-Hamelin, J.I., Dall'Asta, L., Barrat, A., Vespignani, A.: Large scale networks fingerprinting and visualization using the k-core decomposition. In: NIPS, pp. 41–50 (2005)
4. Batagelj, V., Zaversnik, M.: An o(m) algorithm for cores decomposition of networks. CoRR, cs.DS/0310049 (2003)

5. Bhawalkar, K., Kleinberg, J., Lewi, K., Roughgarden, T., Sharma, A.: Preventing unraveling in social networks: the anchored k-core problem. SIAM J. Discrete Math. **29**(3), 1452–1475 (2015)
6. Boden, B., Günnemann, S., Hoffmann, H., Seidl, T.: Mining coherent subgraphs in multi-layer graphs with edge labels. In: SIGKDD, pp. 1258–1266 (2012)
7. Carmi, S., Havlin, S., Kirkpatrick, S., Shavitt, Y., Shir, E.: A model of internet topology using k-shell decomposition. PNAS **104**(27), 11150–11154 (2007)
8. Daianu, M., et al.: Breakdown of brain connectivity between normal aging and Alzheimer's disease: a structural k-core network analysis. Brain Connectivity **3**(4), 407–422 (2013)
9. Dickison, M.E., Magnani, M., Rossi, L.: Multilayer Social Networks. Cambridge University Press, New York (2016)
10. Galimberti, E., Bonchi, F., Gullo, F.: Core decomposition and densest subgraph in multilayer networks. In: CIKM, pp. 1807–1816 (2017)
11. Hu, H., Yan, X., Huang, Y., Han, J., Zhou, X.J.: Mining coherent dense subgraphs across massive biological networks for functional discovery. In: International Conference on Intelligent Systems for Molecular Biology, pp. 213–221 (2005)
12. Jethava, V., Beerenwinkel, N.: Finding dense subgraphs in relational graphs. In: Appice, A., Rodrigues, P.P., Santos Costa, V., Gama, J., Jorge, A., Soares, C. (eds.) ECML PKDD 2015. LNCS (LNAI), vol. 9285, pp. 641–654. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23525-7_39
13. Khaouid, W., Barsky, M., Srinivasan, V., Thomo, A.: K-core decomposition of large networks on a single PC. Proc. VLDB Endowment **9**(1), 13–23 (2015)
14. Li, R., Su, J., Qin, L., Yu, J.X., Dai, Q.: Persistent community search in temporal networks. In: ICDE, pp. 797–808 (2018)
15. Liu, B., Yuan, L., Lin, X., Qin, L., Zhang, W., Zhou, J.: Efficient (α, β)-core computation: an index-based approach. In: The World Wide Web Conference, WWW 2019, pp. 1130–1141. ACM, New York (2019)
16. Luce, R.D.: Connectivity and generalized cliques in sociometric group structure. Psychometrika **15**(2), 169–190 (1950)
17. Luce, R.D., Perry, A.D.: A method of matrix analysis of group structure. Psychometrika **14**(2), 95–116 (1949)
18. Malliaros, F.D., Vazirgiannis, M.: To stay or not to stay: modeling engagement dynamics in social graphs. In: CIKM, pp. 469–478 (2013)
19. Montresor, A., De Pellegrini, F., Miorandi, D.: Distributed k-core decomposition. IEEE Trans. Parallel Distrib. Syst. **24**(2), 288–300 (2013)
20. Seidman, S.B.: Network structure and minimum degree. Soc. Netw. **5**(3), 269–287 (1983)
21. Seidman, S.B., Foster, B.L.: A graph-theoretic generalization of the clique concept. J. Math. Soc. **6**(1), 139–154 (1978)
22. Wen, D., Qin, L., Zhang, Y., Lin, X., Yu, J.X.: I/O efficient core graph decomposition at web scale. In: ICDE, pp. 133–144 (2016)
23. Wu, Y., Jin, R., Zhu, X., Zhang, X.: Finding dense and connected subgraphs in dual networks. In: ICDE, pp. 915–926 (2015)
24. Zhang, F., Li, C., Zhang, Y., Qin, L., Zhang, W.: Finding critical users in social communities: the collapsed core and truss problems. In: TKDE (2018)
25. Zhang, F., Yuan, L., Zhang, Y., Qin, L., Lin, X., Zhou, A.: Discovering strong communities with user engagement and tie strength. In: Pei, J., Manolopoulos, Y., Sadiq, S., Li, J. (eds.) DASFAA 2018. LNCS, vol. 10827, pp. 425–441. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91452-7_28
26. Zhang, F., Zhang, W., Zhang, Y., Qin, L., Lin, X.: OLAK: an efficient algorithm to prevent unraveling in social networks. PVLDB **10**(6), 649–660 (2017)

27. Zhang, F., Zhang, Y., Qin, L., Zhang, W., Lin, X.: When engagement meets similarity: Efficient (k, r)-core computation on social networks. PVLDB **10**(10), 998–1009 (2017)
28. Zhang, F., Zhang, Y., Qin, L., Zhang, W., Lin, X.: Efficiently reinforcing social networks over user engagement and tie strength. In: ICDE, pp. 557–568 (2018)
29. Zhu, R., Zou, Z., Li, J.: Diversified coherent core search on multi-layer graphs. In: ICDE, pp. 701–712 (2018)