Supplementary Material –
Fast Multilayer Core Decomposition and Indexing

A. Supplementary Explanation to MLCS (Algorithm 4).

We here present a more detailed explanation to the MLC-tree-based ML core search algorithm MLCS in Algorithm 4.

As stated in Section IV-B1, when provided with a vector $\mathbf{k}$, MLCS first calls Procedure SEARCH to locate the $\mathbf{k}$-node N in the MLC-tree and then invokes Procedure RECOVER to restore the ML core represented by N.

Procedure SEARCH (lines 3–8) works as follows: it repeatedly calls Procedure FORWARD to find the path from the root node of the MLC-tree to the $\mathbf{k}$-node. During this process, we maintain a node N, along with a counter i, initialized to the root of the MLC-tree and 1, respectively. Each call to Procedure FORWARD examines the coreness vector represented by N, denoted as $\mathbf{k}_N$. If $\mathbf{k}_N[i] < \mathbf{k}[i]$, the procedure forwards to search the child node N' of N with vectors different from $\mathbf{k}_N$ in their i-th components and thereby returns N' with i (lines 10–13). Otherwise, the i-th component of $\mathbf{k}_N$ already matches that of $\mathbf{k}$. It then returns N and $i+1$ to move on to the matching of their $(i+1)$-th components (lines 14–15). Procedure FORWARD establishes the following invariant, which ensures the correctness of Procedure SEARCH:

Lemma 1. *Given N and i as input, let N' and i' be the output of Procedure* FORWARD. *N' must be on the path from the root to the $\mathbf{k}$-node in the MLC-tree. And if $i' = i+1$, we have that $\mathbf{k}_{N'}[j] = \mathbf{k}[j]$ for $j = 1, 2, \ldots, i$.*

Procedure RECOVER (lines 16–23) repeatedly collects the vertex sets associated with the nodes along the rightmost path starting from N down to a leaf node and returns the union of these vertex sets as the result. This process correctly restores the ML core represented by N, as guaranteed by Theorem 3.

B. Supplements for MLC-tree-based WDS.

The pseudocode for the MLC-tree-based WDS algorithm is given in Algorithm 5. It makes a DFS traversal on the MLC-tree, and extracts the ML core maximizing the weighted density as the result. The basic DFS traversal framework is clear from the pseudocode, and we detail the search process on a given rightmost path, which is outlined in Procedure SEARCHRMPATH.

Procedure SEARCHRMPATH (lines 12–24) takes an MLC-tree T and a node N as arguments. It returns the node N_{dense} with the highest weighted density on N's rightmost path with its density ρ, the number n of vertices in the ML core represented by N, and the per-layer edges numbers in the subgraph induced by this ML core, which is stored in an array m. The procedure starts by going straight to the leaf node of N's rightmost path (lines 13–14) and then identifying the densest ML core on the path during backtracking. Before initiating backtracking, n and all components of m are set to 0 (line 16). When backtracking to node N, letting C be the ML core represented by rightmost child of N, n and m already store the size of C and the per-layer edges in the subgraph induced by C, respectively (line 14). We then obtain the vertex set S associated with N in line 17. Theorem 3 guarantees that the ML core represented by N equals $C \cup S$. Therefore, the number of vertices in it can be restored by adding $|S|$ to n (line 18). However, restoring the per-layer edge numbers is a bit tricky (lines 19–20): it has to call Procedure GetDiffEdge to compute the number of newly introduced edges ΔE_i due to the involvement of S in each layer i, i.e., the ones with one endpoint in S while the other in $S \cup C$, and then add the value to $m[i]$. Then, the weighted density of the ML core represented by N can be computed using Procedure GetWDensity in line 21 based on n and m. If this ML core attains a higher density than any other ML cores on N's rightmost path, N_{dense}, and ρ are accordingly updated (lines 22–23).

Algorithm 5 MLWDS (Weighted densest subgraph search)

```
Input: The MLC-tree $T$ for an ML graph $\mathcal{G} = (V, E, L)$, a positive real number $\beta$,
    and $|L|$ positive real weights $w_1, w_2, \cdots, w_{|L|}$
Output: The vertex set $C$ with approximate the highest weighted density (Eq. (1)), with
    its density $\rho$
1: $N \leftarrow$ the root of the MLC-tree $T$
2: $N_{dense}, \rho \leftarrow$ SEARCHWDS($T, N$)
3: $C \leftarrow$ RECOVER($T, N_{dense}$)
4: return $C, \rho$
5: procedure SEARCHWDS($T, N$)
6:     $N_{dense}, \rho \leftarrow$ SEARCHRMPATH($T, N$)
7:     for each non-rightmost child $N'$ of $N$ do
8:         $N'_{dense}, \rho' \leftarrow$ SEARCHWDS($T, N'$)
9:         if $\rho' > \rho$ then
10:            $\rho \leftarrow \rho', N_{dense} \leftarrow N'_{dense}$
11:    return $N_{dense}, \rho$
12: procedure SEARCHRMPATH($T$, $N$)
13:    if $N$ is not a leaf node then
14:        $N_{dense}, \rho, n, m \leftarrow$ SEARCHRMPATH($T, N$)
15:    else
16:        $\rho \leftarrow 0, n \leftarrow 0, m \leftarrow \{0, 0, \cdots, 0\}$
17:    $S \leftarrow$ the vertex set associated with $N$
18:    $n \leftarrow n + |S|$
19:    for $i \leftarrow 1, 2, \cdots, |L1|$ do
20:        $m[i] \leftarrow m[i] +$ GETDIFFEDGE($S$)
21:    $\rho' \leftarrow$ GETWDENSITY($n, m$)
22:    if $\rho' > \rho$ then
23:        $\rho \leftarrow \rho', N_{dense} \leftarrow N$
24:    return $N_{dense}, \rho, n, m$
```

C. Supplements to experimental settings.

We here supplement some experimental settings not mentioned in the paper for reproducibility.

1) All ML core decomposition algorithms proposed in this paper are executed on ML graphs with layers sorted in a non-decreasing order of layer density.
2) The hashtable used for storing the ML core decomposition is implemented with separate chaining, where the number of buckets is set to 2^b, with $b = \lceil \log_2 n \rceil$ and n representing the number of nonempty ML cores in the respective graph. The hashing function employed is `hash_range` from the Boost library [1].

D. Effect of layer orders.

TABLE I
RUNNING TIME (S) OF MLCD W.R.T. LAYER ORDERS

Graph	Random	Density/Degeneracy↑	Density/Degeneracy↓
SC	16.25	**7.18**	118.32
DS	**4.44**	4.87/5.11	4.68/4.66
OI	5.67	**3.65**	5.65
A	76.02	**75.23**	130.21
H	237.66	**10.36**	240.38
Ff	44.70	**44.65**	184.85
DL	1367.25	**391.70**/434.52	10638.20/10013.60
W	354.26	333.98/**165.47**	4512.40/4570.11
FG	–	**31355.30**/38318.90	–

TABLE II
MEMORY USAGE (MB) OF THE MLC-TREE W.R.T. LAYER ORDERS

Graph	Random	Density/Degeneracy↑	Density/Degeneracy↓
SC	28.15	**28.03**	225.61
DS	10.02	9.98/**9.95**	9.97/10.01
OI	35.32	**25.12**	35.32
A	**198.84**	**198.84**	371.86
H	255.42	**14.19**	255.42
Ff	**80.36**	**80.36**	197.41
DL	622.77	617.26/**616.05**	1493.33/1496.66
W	225.62	225.62/**121.80**	681.86/681.57
FG	–	**19436.90**	–

In this supplementary experiment, we investigated the impact of different layer orders on the efficiency of ML core decomposition and the memory cost of the constructed MLC-tree index. We tested a total of five orders, including random order, non-decreasing order of layer density or degeneracy, and non-increasing order of layer density or degeneracy. As sorting layers by density and degeneracy usually yields the same order, we report a single value for running time or memory cost in the table; if the resulting orders differ, two values are shown.

The results on the decomposition efficiency are reported in Table I, which align with our expectations: ordering layers in increasing order of their density or degeneracy generally incurs minimal time overhead for most graphs. It is also notable that layer orders influence the decomposition efficiency in ML graphs with wide layer density/degeneracy ranges more significantly. For example, in graph Higgs (H), where the densest layer has a density $445\times$ larger and a degeneracy $41\times$ larger than the sparsest layer, sorting layers in non-decreasing order of layer density leads to a substantial $23.2\times$ speedup compared to sorting in the non-increasing order.

The results on memory costs of the MLC-tree index are presented in Table II. In line with the analysis in Section IV-A, the non-decreasing order of layer degeneracy always yields the smallest MLC-tree. Sorting layers in non-decreasing order of the layer degeneracy is observed to show varying levels of improvements on graphs DBLP-Large (DL) and Wiki (W) compared with sorting based on layer density. On graph DL, these two orders guarantee the same $|L|$-th layer, leading to identical costs for storing differences along the rightmost paths. However, sorting by degeneracy results in fewer nodes with small $lnz(\mathbf{k})$ values, contributing to smaller space costs for storing the tree nodes. In contrast, the significant space reduction observed in graph FG is attributed to the decreased number of rightmost paths that need to be stored.

E. Missing Proofs.

1) Supplementary explanations to the proof for Theorem 1: In the proof of Theorem 1, the time for maintaining the result set R in line 10 of Algorithm 1 is $O(\prod_{i=1}^{|L|} \kappa(G_i)|V|)$, which is not included in the analysis. In fact, if we replace line 10 with associating the set of vertices removed by **PEEL** (line 8) from the father core to the node representing the father core, an MLC-tree index (Section IV-A) will be built after the algorithm terminates. This process incurs a time cost of $O(\prod_{i=1}^{|L|-1} \kappa(G_i)|V|)$, which does not increase the time complexity of a standalone ML core decomposition process without materializing the results.

2) Proof of Theorem 3:

Proof. **(Theorem 3)** Let $N_0, N_1, \cdots, N_m$ be nodes on the rightmost path of N, where $N_0 = N$ and N_m is a leaf, C_i be the ML core represented by N_i, and S_i be the vertex set associated with N_i. We have $S_i = C_i - C_{i+1}$ for $0 \leq i < m$ and $S_m = C_m$. Therefore, $\bigcup_{i=0}^{m} V_i = \bigcup_{i=0}^{m-1}(C_i - C_{i+1}) \bigcup C_m = C_0$, establishing the theorem. □

3) Proof of Theorem 4:

Proof Sketch. **(Theorem 4)** The correctness of Algorithm 4 is ensured by Lemma 1 and Theorem 3. In terms of the time overhead, the total number of tree nodes visited during the execution of Procedure SEARCH and Procedure RESTORE is bounded by the height of the MLC-tree, i.e., $O(\sum_{i=1}^{|L|} \kappa(G_i))$. Considering the extra $|C|$ time to collect the target ML core C, Algorithm 4 runs in $O(\sum_{i=1}^{|L|} \kappa(G_i) + |C|)$ time. □

4) Proof for Theorem 5: We first present some foundations for the proof of Theorem 5. Let $l_c \in L$ be the layer with the maximum degeneracy, i.e., $l_c = \arg\max_{i=1}^{|L|} \kappa(G_i)$, and let $k^* = \kappa(G_{l_c})$. Let $C^*_{\text{SL}} \subseteq V$ be k^*-core on layer l_c. We have that C^*_{SL} maximizes the minimum vertex degree among all possible vertex subsets on all layers. Certainly, $C^*_{\text{SL}} \in \mathcal{C}$.

Lemma 2 ([2]). *$\rho(C^*) \geq \rho(C^*_{SL})$.*

Next, let $\mu(S, l)$ represent the minimum vertex degree in $G_l[S]$, the induced subgraph of S on layer l. Define S^*_{SL} as the weighted densest subgraph among all single layers, i.e., $S^*_{\text{SL}} =$

$\arg\max_{S \subseteq V} \max_{i \in L} \frac{w_i |E_i[S]|}{|S|}$, and l_s be the layer where S^*_{SL} exhibit the highest density.

Lemma 3 ([2])**.** $\mu(S^*_{SL}, l_s) \geq \frac{|E[S^*_{SL}]|}{|S^*_{SL}|}$.

Lemma 4. $\rho(S^*) \leq \frac{w_{l_s} |E[S^*_{SL}]|}{|S^*_{SL}|} |L|^{\beta}$.

We then give the proof of Theorem 5:

Proof. **(Theorem 5)** It holds that:

$$\begin{aligned}
\rho(C^*) &\geq \rho(C^*_{\text{SL}}) \geq \max_{i \in L} \frac{w_i |E_i[C^*_{\text{SL}}]|}{|C^*_{\text{SL}}|} \cdot 1^{\beta} \\
&\geq \frac{1}{2} \max_{i \in L} w_i \mu(C^*_{\text{SL}}, i) \quad \triangleright \text{ avg. degree} > \text{min. degree} \\
&\geq \frac{1}{2} w_{l_c} \mu(C^*_{\text{SL}}, l_c) \\
&\geq \frac{1}{2} w_{l_c} \mu(S^*_{\text{SL}}, l_s) \qquad\qquad \triangleright\ C^*_{\text{SL}} \text{ maximizes } \mu(\cdot) \\
&\geq \frac{1}{2} w_{l_c} \frac{E_{l_s[S^*_{\text{SL}}]}}{|S^*_{\text{SL}}|} \\
&\geq \frac{1}{2} \frac{w_{l_c}}{w_{l_s} |L|^{\beta}} \rho(S^*) \geq \frac{w^-}{2w^+ |L|^{\beta}} \rho(S^*),
\end{aligned}$$

which establishes Theorem 5. □

5) Proof for Theorem 6:

Proof sketch. **(Theorem 6)** Algorithm 5 returns an $\frac{w^-}{2w^+|L|^{\beta}}$-approximation to the optimal solution of the WDS problem. Theorem 6 establishes the correctness of Algorithm 5 and the approximation ratio. Consider the computation for a given rightmost path, the time overhead is dominated by the restoration of per-layer edge numbers. With the basic MLC-tree, the restoration against $\mathcal{G}$ requires $O(|L| \cdot |V| + |E|)$ time, while using the edge-difference-augmented MLC-tree incurs a cost of $\kappa(G_{|L|})$ time. As the number of rightmost paths is bounded by $O(\prod_{i=1}^{|L|-1} \kappa(G_i))$, the time complexity given in the theorem holds. □

REFERENCES

[1] Boost C++ Libraries. boost/container_hash/hash.hpp - 1.77.0. Accessed on: 2023-11. [Online]. Available: https://www.boost.org/doc/libs/1_77_0/boost/container_hash/hash.hpp

[2] E. Galimberti, F. Bonchi, F. Gullo, and T. Lanciano, "Core decomposition in multilayer networks: Theory, algorithms, and applications," *ACM Trans. Knowl. Discov. Data*, vol. 14, no. 1, pp. 11:1–11:40, 2020.