Parallel Colorful h-star Core Maintenance in Dynamic Graphs

Sen Gao
National University of Singapore
Singapore
sen@u.nus.edu

Hongchao Qin
Beijing Institute of Technology
Beijing, China
qhc.neu@gmail.com

Rong-Hua Li
Beijing Institute of Technology
Beijing, China
lironghuabit@126.com

Bingsheng He
National University of Singapore
Singapore
hebs@comp.nus.edu.sg

ABSTRACT

The higher-order structure cohesive subgraph mining is an important operator in many graph analysis tasks. Recently, the colorful h-star core model has been proposed as an effective alternative to h-clique based cohesive subgraph models, in consideration of both efficiency and utilities in many practical applications. The existing peeling algorithms for colorful h-star core decomposition are to iteratively delete a node with the minimum colorful h-star degree. Hence, these methods are inherently sequential and suffer from two limitations: low parallelism and inefficiency for dynamic graphs. To enable high-performance colorful h-star core decomposition in large-scale graphs, we propose highly parallelizable local algorithms based on a novel concept of colorful h-star n-order H-index and conduct thorough analyses for its properties. Moreover, three optimizations have been developed to further improve the convergence performance. Based on our local algorithm and its optimized variants, we can efficiently maintain colorful h-star cores in dynamic graphs. Furthermore, we design lower and upper bounds for core numbers to facilitate identifying unaffected nodes in presence of graph updates. Extensive experiments conducted on 14 large real-world datasets with billions of edges demonstrate that our proposed algorithms achieve a 10 times faster convergence speed and a three orders of magnitude speedup when handling graph changes.

PVLDB Reference Format:
Sen Gao, Hongchao Qin, Rong-Hua Li, and Bingsheng He. Parallel Colorful h-star Core Maintenance in Dynamic Graphs. PVLDB, 16(10): 2538 - 2550, 2023.
doi:10.14778/3603581.3603593

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

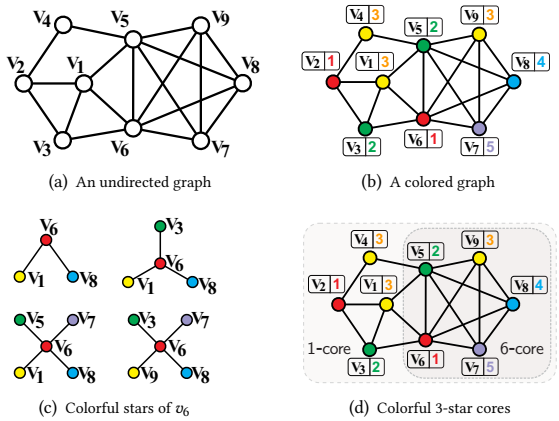
1 INTRODUCTION

The higher-order cohesive subgraph models, which exploit motif structures as basic units of the cohesive parts, have been studied in recent researches. For instance, a study in Science [6] shows that higher-order cohesive subgraph analysis can be applied to understand complex relationships and interactions within networks, such as identifying near-optimal clusters in networks, revealing the functional organization of neuronal networks, and analyzing the connectivity in transportation networks. Fang et al. [14] found the higher-order dense parts of a yeast protein-protein interaction (PPI) network [21, 33, 45] have distinct shapes and present a subnetwork with a specific function. Besides, our case studies show that analyzing the higher-order structures of NFT communities enables stakeholders to gain a deeper understanding of emerging trends and market dynamics within the ecosystem, allowing for well-informed decision-making and the recognition of opportunities and risks.

The h-clique densest subgraph [39, 42] is a maximal subgraph with the largest average number of h-cliques per node. Though this model performs well in extracting higher-order information, its computation is prohibitively expensive, especially on large-scale networks for large h values [14, 42]. This is because the number of higher-order structures, for example, h-cliques, increases exponentially as the size of the structure h increases. To avoid listing the h-cliques, the colorful h-star k-core model has been proposed in [15], which only takes $O(h \times m)$ time and $O(h \times n + m)$ space to count the motifs (colorful h-stars) for all nodes, where n and m are the numbers of nodes and edges respectively. The results in [15] show that the colorful h-star core can be a good approximation for the h-clique densest subgraph, but it can achieve more than 10× acceleration in most datasets.

However, real-world graphs, such as online social networks [10, 20, 41] and the Internet [2, 9], typically evolve over time. The existing peeling algorithm for colorful h-star core decomposition [15] is to iteratively delete a node with the minimum colorful h-star degree. But in the peeling-based algorithms, the core numbers of nodes are hard to maintain after the graph undergoes edge/node deletions/insertions frequently [1, 43]. For instance, Fig. 9(b) shows in the Q&A website Stack Overflow, this method suffers from an extremely high latency when processing answers generated sequentially. Moreover, the method is inherently sequential and hard to execute in parallel, since the method requires maintaining global information of the whole graph (the minimum colorful h-star degree after peeling) and continuously changes the structure of the graph (removing a node and its adjacent edges affects and needs to update the degrees and colorful h-star degrees of its neighbors). In conclusion, the current algorithm for colorful h-star core decomposition is non-dynamic and exhibits low parallelism.

Proceedings of the VLDB Endowment, Vol. 16, No. 10 ISSN 2150-8097.
doi:10.14778/3603581.3603593



(a) An undirected graph (b) A colored graph

(c) Colorful stars of v_6 (d) Colorful 3-star cores

Figure 1: A colored graph and its colorful 3-star cores.

To address the above-mentioned limitations, we develop a local algorithm to allow later efficient parallelization and core maintenance, with the following two observations. First, the global graph information is not indispensable because the core number of a node can be computed only based on the information of its neighbors. Specifically, we found that given the colorful h-star core number of neighbors, the computation of the core number of this node falls into two cases. By comparing the results of these two cases, the core number can be immediately obtained. Thus, we can compute the core number of each node locally by interacting only with its neighbors. Second, we observe that in dynamic graphs, only a small subset of nodes might be affected after each graph update, so it is unnecessary to compute on the entire graph.

Inspired by the above observations, our algorithm computes the core number for each node from an upper bound iteratively and independently, without undermining the global graph structure. Hence, the core decomposition can be executed in a highly parallelizable manner. To further accelerate the computation, we devise three optimizations that enhance its performance from two different angles: two of them increase the convergence rate, while the other eliminates redundant computations for each iteration. Thanks to the independence and high parallelism of our local algorithm, the cores can be efficiently maintained in dynamic graphs. To be more specific, we first identify nodes whose core numbers need to be updated. The scope of affected nodes can be narrowed by exploiting our elaborated lower and upper bounds of the original core numbers. Then, we update core numbers by leveraging original values or our designed upper bounds on new core numbers.

To summarize, the main contributions of this paper are as follows:

- We introduce a novel concept of colorful h-star n-order H-index, which has a theoretical convergence bound to be computed.
- Based on the n-order H-index, we propose a highly parallelizable local algorithm and three optimizations for colorful h-star core decomposition that achieves 10× faster than baselines.
- We extend our local algorithm to address colorful h-star core maintenance in dynamic graphs, and develop tight lower and upper bounds of core numbers which greatly facilitates affected node identification, reaching three orders of magnitude speedup.
- We conduct experiments on 14 datasets to evaluate our algorithms, and additionally apply them in real-world scenarios to study their performance in higher-order cohesive subgraph mining and real-time maintenance.

Table 1: Notations and descriptions

Notation	Description
$G = (V, E)$	A graph G with nodes set V and edge set E
n, m	$n = \|V\|, m = \|E\|$
$N_u(G)$ $d_u(G)$	$N_u(G)$ is the set of neighbors of u in G $d_u(G) = \|N_u(G)\|$
χ $\text{color}(u)$	The number of colors used in graph coloring algorithms The color value of u
$d_u(G, \mathcal{S})$	The colorful h-star degree of u in G
$C_k(G, \mathcal{S})$ or C_k $c_u(G, \mathcal{S})$ or c_u	The colorful h-star k core The colorful h-star core number of u
$\mathsf{H}_u^{(n)}(G, h)$ $\mathsf{DP}^{(n)}(i)$	n-order H-index The number of colorful h-stars that u obtains on the first i neighbors in the order of n
$p^{(n)}$	The index of a neighbor of u that satisfies specific conditions in the order of n
$w^{(n)}$	The neighbor of u with the index $p^{(n)}$ in the order of n
$\mathsf{L}_u^{(n)}$	The set of the first $p^{(n)}$ neighbors of u in the order of n
c_{lb}^-	The lower bound of core numbers of affected nodes after an edge deletion
c_{ub}^+	The upper bound of core numbers of affected nodes after an edge deletion

2 PRELIMINARIES

Let $G = (V, E)$ be an undirected and unweighted graph, where V ($|V| = n$) and E ($|E| = m$) denote the set of nodes and edges respectively. We denote the set of neighbor nodes of u in G with $N_u(G)$, and $d_u(G) = |N_u(G)|$ denotes the degree of u in G. A subgraph $H = (V_H, E_H)$ is called an induced subgraph of G if $V_H \subseteq V$ and $E_H = \{(u, v) | (u, v) \in E, u \in V_H, v \in V_H\}$. An h-star is a tree, with one internal or central node having degree $h - 1$ and the other $h - 1$ nodes having degree 1.

Graph coloring is a procedure that assigns an integer color value taken from $[1, \cdots, \chi]$ to each node u in G, denoted by $\text{color}(u)$, so that no two adjacent nodes have the same color value. Since the minimum coloring problem (χ is minimum) is NP-hard [4], we make use of linear-time greedy coloring algorithms [17, 48] to obtain a valid coloring. Example 1 illustrates the coloring procedure.

Example 1. *Consider an undirected graph in Fig. 1(a). Here we apply a widely-used graph coloring algorithm that colors all nodes following a non-increasing order of their degrees, which is $(v_6, v_5, v_1, v_9, v_7, v_3, v_2, v_4)$ in the given graph. Fig. 1(b) shows the result of this coloring procedure. The number next to a Node ID indicates a corresponding color value assigned to this node.*

Based on a valid coloring, we first introduce the concepts of the colorful h-star and the colorful h-star degree as follows.

Definition 1 (Colorful h-star and degree). *([15]) Given a colored graph $G = (V, E)$ and an integer $h \geq 2$, an h-star in G is colorful, denoted by $\mathcal{S}_h$, if any pair of nodes $u, v \in \mathcal{S}$ have different color values. A colorful h-star belongs to u if it centers on u. The colorful h-star degree of u, denoted by $d_u(G, \mathcal{S}_h)$, is the number of colorful h-stars centered on u.*

Definition 2 (Colorful h-star k core). *([15]) Given a colored graph $G = (V, E)$ and an integer h. The colorful h-star k core, or $(k, \mathcal{S}_h)$-core of G, denoted by $C_k(G, \mathcal{S}_h)$, is the maximal subgraph G' such that $\forall u \in V_{G'}$, $d_u(G', \mathcal{S}_h) \geq k$.*

The colorful h-star core number of u, denoted by $c_u(G, \mathcal{S}_h)$, is the largest k such that there exists a colorful h-star k core containing u. If the context is clear, we will use C_k and c_u instead of $C_k(G, \mathcal{S}_h)$ and $c_u(G, \mathcal{S}_h)$ for simplicity.

Problem 1 (Colorful h-star core decomposition). *Given a graph G and an integer h. The colorful h-star core decomposition problem is to compute the colorful h-star core number of each $u \in V$.*

Problem 2 (Colorful h-star core maintenance). *To maintain the colorful h-star core numbers of all nodes is to update them after deleting/inserting edges from/into G. If the two end-nodes of an inserted edge share the same color value, an efficient recoloring strategy is first applied to assign the smallest valid color to the end-node with a lower core number. This ensures that the colorful h-star core numbers remain accurate after changes to the graph's structure.*

Example 2. *Reconsider the colored graph in Fig. 1(b). Fig. 1(c) lists a colorful 3-star, two colorful 4-stars and two colorful 5-stars of u. Clearly, in this graph the colorful 3-star degree of v_3 is 2, because there are two colorful 3-stars $\{v_3, v_2, v_1\}$ and $\{v_3, v_6, v_1\}$ centering on v_3. Fig. 1(d) shows all colorful 3-star cores of G, we can see that the subgraph induced by $\{v_5, v_6, v_7, v_8, v_9\}$ is a colorful 3-star 6 core, since each node in this 5-clique has 6 colorful 3-stars. Obviously, the entire graph is a colorful 3-star 1 core.*

3 THE COLORFUL h-STAR n-ORDER H-INDEX

In this section, we first introduce the concept of colorful h-star n-order H-index which is the cornerstone of our local algorithm. Then we will show our theoretical findings on its properties.

3.1 n-order H-index

Motivations. The existing method [15] computes the colorful h-star core decomposition by iteratively deleting a node with the minimum colorful h-star degree and updating core numbers of its neighbors. This algorithm is hard to parallelize since it 1) requires global knowledge, i.e., finding a node with the minimum colorful h-star degree in the remaining graph, and 2) impacts the graph structure after removing nodes and edges. Thus, this algorithm is inherently sequential.

To develop a parallelizable algorithm, we first design the novel colorful h-star n-order H-index based on the ideas of n-order H-index for k-core decomposition proposed by Eugene et al. [27]. However, it is infeasible to apply this H-index directly to solve our problem which involves counting higher-order motifs, i.e., color h-stars for each node, not focusing on simple edges.

Observation on the colorful h-star core numbers. We here show that the core number of u can be computed only from the core numbers of its neighbors. We first prove the following theorem.

Theorem 1. *Given a graph G and a node u, let $k = c_u$ be the core number of u and let w be the neighbor of u in C_k with the minimum core number. Then we have $c_u = \min(c_w, d_u(C_k, \mathcal{S}))$.*

According to Theorem 1, c_u can be determined by $d_u(C_k, \mathcal{S})$ and c_w. Note that $d_u(C_k, \mathcal{S})$ can be obtained by counting the number of colorful h-stars on neighbors with core numbers no smaller than u. Therefore, if we find out the neighbor w, then $d_u(C_k, \mathcal{S})$ and c_w can be easily derived. The detailed computation as illustrated in Fig. 2 has the following four steps.

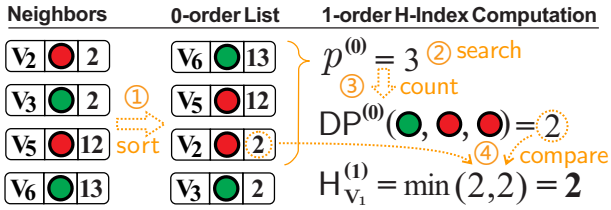


Figure 2: The computation for v_1's 1-order H-index ($h = 3$).

- **S1** Sort the neighbors in non-increasing order of their core numbers $(v_1, v_2, \ldots, v_{d_u})$, with the intuition that $d_u(C_k, \mathcal{S})$ is only related to neighbors with large core numbers.
- **S2** Search w by testing each neighbor in this order until the p-th neighbor satisfies either $\mathsf{DP}(p) \geq c_{v_p}$ (*case 1*) or $(\mathsf{DP}(p) < c_{v_p}) \wedge (\mathsf{DP}(p) \geq c_{v_{p+1}})$ (*case 2*), where $\mathsf{DP}(p)$ is the number of u's colorful h-stars computed on the first p neighbors, then the node v_p is w.
- **S3** Compute $\mathsf{DP}(p)$, which is equal to $d_u(C_k, \mathcal{S})$.
- **S4** Compare c_{v_p} and $\mathsf{DP}(p)$ and assign the minimum to c_u.

Based on these observations, we define our colorful h-star n-order H-index, also called n-order H-index for simplicity.

Definition 3 (n-order H-index). *Given a colored graph $G = (V, E)$, a node $u \in V$, and a positive integer h, the n-order H-index of u w.r.t. h on G, denoted by $\mathsf{H}_u^{(n)}(G, h)$, is defined by the following recurrence relation*

$$\mathsf{H}_u^{(n)}(G,h) = \begin{cases} d_u(G, \mathcal{S}) & n = 0 \\ \min(\mathsf{H}_{w^{(n-1)}}^{(n-1)}(G,h), \mathsf{DP}^{(n-1)}(p^{(n-1)})) & n > 0 \end{cases} \tag{1}$$

Here $\mathsf{H}_{w^{(n-1)}}^{(n-1)}(G,h)$ and $\mathsf{DP}^{(n-1)}(p^{(n-1)})$ indicate the results of the above two cases respectively. We will omit the subscript u if the context is clear.

To formally define $\mathsf{DP}^{(n)}$ and $p^{(n)}$, we first propose the n-order neighbor list of u. The u's n-order neighbor list $(v_1^{(n)}, v_2^{(n)} \cdots v_{d_u}^{(n)})$ contains neighbor nodes of u sorted in non-increasing order of their $(n-1)$-order H-index $\mathsf{H}_{v_i}^{(n-1)}(G,h)$, $v_i \in N_u(G)$.

$\mathsf{DP}^{(n)}(i)$ associated with order n, denotes the number of colorful h-stars centering on u and with the other $h-1$ leaves coming from the first i neighbors of u's n-order neighbor list. Obviously, $\mathsf{DP}^{(0)}(d_u(G))$ is equal to $d_u(G, \mathcal{S})$. Here we define $p^{(n)}$ as follows:

$$p^{(n)} = \min\{i \in \{1, 2, \cdots, d_u(G)\} : !(\mathsf{DP}^{(n)}(i) < \mathsf{H}_{v_i^{(n)}}^{(n)}(G,h) \wedge \mathsf{DP}^{(n)}(i) < \mathsf{H}_{v_{i+1}^{(n)}}^{(n)}(G,h))\}, \tag{2}$$

and $w^{(n)} = v_{p^{(n)}}^{(n)}$. We use $\mathsf{L}_u^{(n)}$ to denote the set of the first $p^{(n)}$ nodes in the n-order neighbor list of u.

Example 3. *Fig. 2 shows the computational procedure of v_1's 1-order H-index in four steps. In this example, each triad associated with a node contains three elements: Node ID, Node with color and 0-order H-index of this node. All 0-order H-indexes of v_1's neighbors are initiated with their colorful 3-star degrees, i.e., $< 2, 2, 12, 13 >$ for $< v_2, v_3, v_5, v_6 >$ respectively. We first obtain 0-order neighbor list of*

Algorithm 1: n-order H-index Based Local Algorithm

Input: A graph G and an integer h
Output: The colorful h-star core number of each node $u \in V$

1 color$[1, \cdots, n] \leftarrow$ GreedyColoring(G);
2 **for each** *node* $u \in V$ **in parallel do**
3 $\quad d_u(G, \mathcal{S}) \leftarrow$ Count$(G, h, u,$ color$)$; // proposed in [15]
4 $\quad \mathsf{H}_u^{(0)}(G, h) \leftarrow d_u(G, \mathcal{S})$;
5 updateFlag $\leftarrow$ true; $n \leftarrow 0$;
6 **while** updateFlag **do**
7 $\quad$ updateFlag $\leftarrow$ false; $n \leftarrow n + 1$;
8 $\quad$ **for each** *node* $u \in V$ **in parallel do**
9 $\quad\quad C \leftarrow \{\mathsf{H}_v^{(n-1)}(G, h) | v \in N_u(G)\}$;
10 $\quad\quad < \mathsf{H}_u^{(n)}(G, h), p^{(n-1)} > \leftarrow$ ComputeHIndex(u, C);
11 $\quad\quad$ **if** $\mathsf{H}_u^{(n)}(G, h) \neq \mathsf{H}_u^{(n-1)}(G, h)$ **then**
12 $\quad\quad\quad$ updateFlag $\leftarrow$ true;
13 $c_u(G, \mathcal{S}) \leftarrow \mathsf{H}_u^{(n)}(G, h)$ for each $u \in V$;
14 **return** $c_u(G, \mathcal{S})$ for each $u \in V$;

15 **Procedure** GreedyColoring(G)
16 Let π' be any ordering on nodes;
17 $flag(i) \leftarrow -1$ for $i = 1, \cdots, \chi$;
18 **for** *each node* $v \in \pi'$ **in order do**
19 $\quad$ **for** $u \in N_v(G)$ **do**
20 $\quad\quad flag($color$(u)) \leftarrow v$;
21 $\quad c \leftarrow \min\{i | i > 0, flag(i) \neq v\}$;
22 $\quad$ color$(v) \leftarrow c$;
23 **return** color(v) for each $v \in V$;

Algorithm 2: n-order H-index Computation

Input: A node u and a set $C = \{\mathsf{H}_v^{(n-1)}(G, h) | v \in N_u(G)\}$
Output: The n-order H-index of u

1 **Procedure** ComputeHIndex(u, C)
2 sort neighbor nodes of u such that
$\quad \mathsf{H}_{v_1}^{(n-1)}(G, h) \geq \mathsf{H}_{v_2}^{(n-1)}(G, h) \geq \cdots \geq \mathsf{H}_{v_{d_u}}^{(n-1)}(G, h)$;
3 Let χ be the number of different colors used in GreedyColoring;
4 $\mathsf{DP}^{(n-1)}(0) \leftarrow 1$;
5 **for** $i = 1$ **to** $d_u(G)$ **do**
6 $\quad \mathsf{DP}^{(n-1)}(i) \leftarrow$ Updating$(\mathsf{DP}^{(n-1)}(i-1), h, v_i,$ color$(v_i))$; // proposed in [15]
7 $\quad$ **if** $\mathsf{DP}^{(n-1)}(i) \geq \mathsf{H}_{v_i}^{(n-1)}(G, h)$ **or** $\mathsf{DP}^{(n-1)}(i) \geq \mathsf{H}_{v_{i+1}}^{(n-1)}(G, h)$ **then**
8 $\quad\quad \mathsf{H}_u^{(n)}(G, h) \leftarrow \min\{\mathsf{H}_{v_i}^{(n-1)}(G, h), \mathsf{DP}^{(n-1)}(i)\}$;
9 $\quad\quad p^{(n-1)} \leftarrow i$;
10 $\quad\quad$ **break**;
11 **return** $< \mathsf{H}_u^{(n)}(G, h), p^{(n-1)} >$;

v_1 *by sorting its neighbors in non-increasing order of their 0-order H-indexes. Then* $p^{(0)} = 3$ *can be found by Equation 2, and after that* $\mathsf{DP}^{(0)}$ *can be derived from computing the colorful 3-star degrees on the first* $p^{(0)}$ *nodes of* v_1*'s 0-order neighbor list. Finally, by Equation 1, we compare* $\mathsf{DP}^{(0)}$ *with the 0-order H-index of the third neighbor, and set* $\mathsf{H}_{v_1}^{(1)}(G, 3)$ *to the minimum value of them.*

3.2 Theoretical findings

Theorem 2 (Monotonicity). *Given a graph G, a node $u \in V$, and an integer h,* $\mathsf{H}_u^{(n)}(G, h) \leq \mathsf{H}_u^{(n-1)}(G, h)$ *for any* $n \in \mathbb{N}$.

Due to the space limit, all proofs of the following theorems and lemmas can be found in [16].

Theorem 3 (Convergence).

$$\lim_{n \to \infty} \mathsf{H}_u^{(n)}(G, h) = c_u(G, \mathcal{S}) \tag{3}$$

To bound the number of iterations that each node takes to converge, we borrow the idea of building a node hierarchy from [25].

Definition 4 (Colorful h-star degree hierarchy). *Given a colored graph $G = (V, E)$ and two integers h and i. For each $i \geq 1$, V_i is comprised of nodes with the minimum colorful h-star degrees in the induced subgraph of $V_G \setminus \bigcup_{0 \leq j \leq i-1} V_j$, where V_0 contains nodes with the minimum colorful h-star degrees in G.*

Theorem 4 (Theoretical convergence bound). *Given a graph G and a node $u \in V$, computing the n-order H-index of u takes at most i iterations to converge if $u \in V_i$.*

4 LOCAL ALGORITHM AND OPTIMIZATIONS

In this section, we first present the n-order H-index based local algorithm for colorful h-star core decomposition in Section 4.1. Then we propose three optimization strategies in Section 4.2.

4.1 Local algorithms

Our n-order H-index Based Local Algorithm is outlined in Algorithm 1. First, we apply a greedy graph coloring algorithm to color graph G (line 1). Then, Algorithm 1 computes the colorful h-star degrees of all nodes in parallel as upper bounds of colorful h-star core numbers by invoking a dynamic-programming method Count proposed in [25] (lines 2-4). After that, this algorithm iteratively recomputes $\mathsf{H}_u^{(n)}(G, h)$ for all nodes by the procedure ComputeHIndex, the results of which are used as updated upper bounds of colorful h-star core numbers (lines 6-12). Note that the updating of $\mathsf{H}_u^{(n)}(G, h)$ is independent of other nodes, thus recomputation in each iteration can be easily paralleled (line 8). The new upper bounds will only get smaller after being updated due to the monotonicity of n-order H-index. Finally, it will return the latest $\mathsf{H}_u^{(n)}(G, h)$ as colorful h-star core numbers for all nodes (lines 13-14).

The pseudocode of computing n-order H-index of a specific node u is shown in Algorithm 2. Given a node u, ComputeHIndex first sorts its neighbor nodes in a non-increasing order of their $(n-1)$-order H-index (line 2). Then the algorithm aims to find $p^{(n-1)}$ such that the n-order H-index of u can be computed with the first $p^{(n-1)}$ neighbors (lines 5-9). For the first i neighbors, the procedure starts by counting the number of colorful h-stars formed by u and nodes within the first i neighbors using an Updating technique proposed in [25] (line 6). It is worth mentioning that invoking the Updating procedure requires very small computational effort as it incrementally computes $\mathsf{DP}^{(n)}(i)$ after the i-th neighbor is involved. Then, ComputeHIndex will compare the current number of colorful h-stars with the $(n-1)$-order H-index of u's i-th and $(i+1)$-th neighbor nodes (line 6). If conditions are met, this algorithm will stop expanding the search and assign the minimum value of the $(n-1)$-order H-index of u's i-th neighbor and the current number of colorful h-stars (lines 7-9).

Remark. The performance of the Count procedure in Algorithm 1 and the Updating procedure in Algorithm 2 is affected by the adopted graph coloring, which has been thoroughly studied in [15]. Among all popular graph coloring techniques, the degree-based greedy graph coloring, i.e., coloring nodes following a non-increasing order of degrees (line 16 of Algorithm 1), is the most efficient method. Thus we choose this coloring as a default setting.

Example 4. *The execution of Local Algorithm on the example graph Fig. 1(b) to compute its colorful 3-star core decomposition is illustrated in Table 2. Here we can first see that* $\mathsf{H}_{v_i}^{(0)}(G,3)$ *represents the colorful 3-star degree of* v_i*. After 4 rounds of iterations, for each node* v_i*,* $\mathsf{H}_{v_i}^{(4)}(G,3) = \mathsf{H}_{v_i}^{(3)}(G,3)$*, which indicates the n-order H-indexes of all nodes have converged to their core numbers. The Local Algorithm invokes* ComputeHIndex *procedure 36 times during all updating steps, as each node runs the procedure in every iteration.*

The following theorem details the time and space complexity of Local Algorithm.

Theorem 5. *Given a colored graph* G *and an integer* h*, Algorithm 1 computes the colorful* h*-star core decomposition of* G *in* $O(tnd_{\max}(h+\log d_{\max}))$ *time using* $O(hn+m)$ *space, where* t *is the number of iterations,* $d_{\max}$ *is the maximum degree of nodes in* G*.*

4.2 Optimizations

In this section, we develop three optimization strategies to accelerate Local Algorithm. Based on time complexity analysis of Local Algorithm, we classify these three strategies into two categories: inter- and intra-iteration optimizations. For inter-iteration processing, the number of iterations is quite essential to the total computational effort. For intra-iteration processing, we observe that there exist some redundant computations in each iteration that could cause severe time overhead and can be safely pruned.

4.2.1 Inter-Iteration Processing. We propose OPT-1 and OPT-2 to accelerate the convergence of n-order H-index.

OPT-1: Asynchronous computing In the i-th iteration, the basic Local Algorithm computes $\mathsf{H}_u^{(i)}(G,h)$ based on its neighbors' $(i-1)$-order H-indexes. However, some neighbors' i-order H-indexes have been computed in the i-th iteration before processing node u. Our intuition is to leverage neighbors' newer values $\mathsf{H}_v^{(i)}(G,h)$ instead of the stale ones $\mathsf{H}_v^{(i-1)}(G,h)$ to update u for all $v \in N_u$. A faster convergence can be achieved by computing $\mathsf{H}_u^{(n)}(G,h)$ asynchronously, hence reducing the total number of iterations. In this case, the only difference is that the i-order neighbor list will be obtained by sorting neighbors based on their latest H-Indexes. Therefore, similar analyses as those in Theorem 2 and Theorem 3 will guarantee the efficiency and correctness of asynchrony, respectively.

Example 5. *The detailed updating steps of Local Algorithm equipped with* OPT-1 *on the toy graph in Fig. 1(b) are depicted in Table 2. As can be seen, in the first iteration,* $\mathsf{H}_{v_3}^{(1)}(G,3)$ *converges to the core number of* v_3 *since its two neighbors* v_1 *and* v_2 *have been processed before* v_3*.* OPT-1 *takes 3 iterations to converge, faster than* Local *using a synchronous processing strategy.*

OPT-2: Processing ordering heuristic Revisit the graph G in Fig. 1(b) and updating steps of Local Algorithm and OPT-1 in Table 2. It is clear that v_1 and v_4 have the same core values, but the numbers of iterations that the n-order H-indexes of them take to converge show a big difference, 3 of v_1 compared to 1 of v_4. One non-trivial reason is that v_1 has a larger initial upper bound of its core value, i.e., $\mathsf{H}_{v_1}^{(0)}(G,h) > \mathsf{H}_{v_4}^{(0)}(G,h)$. Here we propose OPT-2 to significantly accelerate the convergence of nodes with large upper bounds. We observe that to compute i-order H-index of u, OPT-1 always uses the latest H-indexes of u's neighbors to update u. If more neighbors have been processed in the i-th iteration, then according to the monotonicity of n-order H-index, u will get a smaller $\mathsf{H}_u^{(i)}(G,h)$. Based on our observation, if in each iteration we process nodes following an increasing order of their $(i-1)$-order H-indexes instead of the random processing order, i.e., in the order of node IDs, nodes with large upper bounds will converge faster.

Table 2: Illustration of Local Algorithm and Optimizations Applied to the Example Graph($h=3$, ⊘: computation pruned)

Methods	$\mathsf{H}_v^{(n)}(G,3)$	n-order H-index								
		v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
Local	$\mathsf{H}^{(0)}$	4	2	2	1	12	13	6	6	6
	$\mathsf{H}^{(1)}$	2	1	2	1	6	6	6	6	6
	$\mathsf{H}^{(2)}$	2	1	1	1	6	6	6	6	6
	$\mathsf{H}^{(3)}$	1	1	1	1	6	6	6	6	6
	$\mathsf{H}^{(4)}$	1	1	1	1	6	6	6	6	6
OPT-1	$\mathsf{H}^{(1)}$	2	1	1	1	6	6	6	6	6
	$\mathsf{H}^{(2)}$	1	1	1	1	6	6	6	6	6
	$\mathsf{H}^{(3)}$	1	1	1	1	6	6	6	6	6
OPT-2	$\mathsf{H}^{(1)}$	1	1	1	1	6	6	6	6	6
	$\mathsf{H}^{(2)}$	1	1	1	1	6	6	6	6	6
OPT-3	$\mathsf{H}^{(1)}$	2	1	2	1	6	6	6	6	6
	$\mathsf{H}^{(2)}$	⊘	⊘	1	⊘	⊘	⊘	⊘	⊘	⊘
	$\mathsf{H}^{(3)}$	1	⊘	⊘	⊘	⊘	⊘	⊘	⊘	⊘
	$\mathsf{H}^{(4)}$	⊘	⊘	⊘	⊘	⊘	⊘	⊘	⊘	⊘
Local + OPT-1-2-3	$\mathsf{H}^{(1)}$	1	1	1	1	6	6	6	6	6
	$\mathsf{H}^{(2)}$	⊘	⊘	⊘	⊘	⊘	⊘	⊘	⊘	⊘

However, in different iterations, the node ordering can also be different, so we have to dynamically pick nodes to process, which adds extra maintenance overhead. Therefore, we replace it with the degree-based ordering, i.e., computing n-order H-indexes in a non-decreasing order of degrees since we found nodes with large initial upper bounds, which are equal to their colorful h-star degrees, usually have large node degrees.

Example 6. *Table 2 illustrates the* OPT-2 *strategy on graph* G *in Fig. 1(b). In each iteration, n-order H-indexes can be computed in the non-decreasing order of node degrees* $(v_4, v_2, v_3, v_1, v_7, v_8, v_9, v_5, v_6)$*. Notice that the n-order H-index of* v_1 *converges to its core number 1 after only 1 iteration since* v_1 *is processed later than its two neighbor nodes* v_2 *and* v_3 *with smaller degrees. Finally,* OPT-2 *reduces the total number of iterations to 2.*

4.2.2 Intra-Iteration Processing. We develop OPT-3 to skip some redundant computation within one iteration.

OPT-3: Pruning technique As can be seen from Algorithm 1, the Local Algorithm calls ComputeHIndex to compute n-order H-indexes for all nodes in each iteration, though n-order H-indexes of some nodes do not change after processing. Take Local Algorithm in Table 2 for example. In the second iteration, $\mathsf{H}_{v_1}^{(2)}(G,h)$ remains the same after computed, which implies that the computation on v_2 is redundant and unnecessary, and should be avoided for efficiency purposes. To this end, we propose OPT-3, a pruning technique that can detect redundant computation and skip nodes whose n-order H-indexes will not be changed in each iteration. The theorem guarantees the safe pruning of unnecessary computations.

Algorithm 3: The pruning algorithm

Input: A graph G and an integer h
Output: The colorful h-star core number of each node $u \in V$

```
1  color[1, ··· , n] ← GreedyColoring(G);
2  for each node u ∈ V in parallel do
3      d_u(G, S) ← Count(G, h, u, color); // proposed in [15]
4      H_u^(0)(G, h) ← d_u(G, S);
5  updateFlag ← true; n ← 0;
6  while updateFlag do
7      updateFlag ← false; n ← n + 1;
8      for each node u ∈ V in parallel do
9          skip ← true;
10         for i = 1 to p^(n−2) do
11             v ← the i-th node of the (n − 2)-order neighbor list of u.;
12             if H_v^(n−1)(G, h) < H_u^(n−1)(G, h) then
13                 skip ← false;
14                 break;
15         if skip then
16             < H_u^(n)(G, h), p^(n−1) > ← < H_u^(n−1)(G, h), p^(n−2) >;
17             continue;
18         C ← {H_v^(n−1)(G, h) | v ∈ N_u(G)};
19         < H_u^(n)(G, h), p^(n−1) > ← ComputeHIndex(u, C);
20         if H_u^(n)(G, h) ≠ H_u^(n−1)(G, h) then
21             updateFlag ← true;
22 c_u(G, S) ← H_u^(n)(G, h) for each u ∈ V;
23 return c_u(G, S) for each u ∈ V;
```

Theorem 6. *Given a colored graph G, a node u and two integers h, n, $\mathsf{H}_u^{(n)}(G,h) = \mathsf{H}_u^{(n-1)}(G,h)$ holds if $\mathsf{H}_v^{(n-1)}(G,h) \geq \mathsf{H}_u^{(n-1)}(G,h)$ for any node $v \in L_u^{(n-2)}$, where $L_u^{(n-2)}$ is the set of the first $p^{(n-2)}$ nodes in the $(n-2)$-order neighbor list of u.*

To prove this theorem, we first prove the following lemma.

Lemma 1. *Given a node u, if $\mathsf{H}_v^{(n-1)}(G,h) \geq \mathsf{H}_u^{(n-1)}(G,h)$ holds for any node $v \in L_u^{(n-2)}$, then $L_u^{(n-1)} = L_u^{(n-2)}$.*

According Lemma 1, $L_u^{(n-1)}$ is equal to $L_u^{(n-2)}$ which suggests $\mathsf{DP}^{(n-1)}(p^{(n-1)}) = \mathsf{DP}^{(n-2)}(p^{(n-2)})$. By Equation 1, $\mathsf{H}_u^{(n)}(G,h) = \mathsf{H}_u^{(n-1)}(G,h)$. Thus, Theorem 6 holds.

The pruning algorithm is shown in Algorithm 3. When processing a node u, Algorithm 3 first tests the first $p^{(n-2)}$ nodes of the $(n-2)$-order neighbor list of u. If all these $p^{(n-2)}$ nodes have $(n-2)$-order H-indexes larger than that of u (lines 10-14), we do not invoke ComputeHIndex to compute u's n-order H-index. Instead, we assign $\mathsf{H}_u^{(n-1)}(G,h)$ directly to $\mathsf{H}_u^{(n)}(G,h)$ (lines 15-17), because in this case the n-order H-index never changes.

It is worth mentioning that our pruning technique can not only avoid redundant computation after a node has already converged to its core number, but also skip some unnecessary computation before it converges. Besides, this method can be slightly modified to be compatible with other optimizations, like asynchronous computing.

Example 7. *The effect of applying* OPT-3 *to Local Algorithm running on the example graph in Fig. 1(b) is illustrated in Table 2. We can clearly see that* OPT-3 *shares the same number of iterations with* Local*, since it focuses only on handling intra-iteration computation. Here ⊘ denotes the skipped computation. Computations on almost all nodes are pruned three times among 4 iterations. The total number of invocations is significantly reduced by 69.45% after applying* OPT-3.

Finally, we show the power of integrating our Local Algorithm with three optimizations in Table 2. This combination uses the least iterations and invocations, indicating that our proposed optimizations take effect independently and cumulatively.

Distributed colorful h-star core decomposition. It is worth noting that our Local Algorithm can be easily extended to distributed settings. The distributed algorithms under popular distributed graph processing frameworks, such as vertex-centric [26, 28, 32] and block-centric [13, 40, 46], are expected to exhibit excellent scalability with minimal communication overhead. This is because, in each iteration, only the new n-order H-Index of a node will be broadcasted to its neighbors.

5 COLORFUL h-STAR CORE MAINTENANCE IN DYNAMIC GRAPHS

In this section, we leverage the proposed Local Algorithm and optimizations to study the problem of efficiently maintaining the colorful h-star core decomposition in dynamic graphs.

5.1 Colorful h-star core maintenance problem

Many real-world graphs are highly dynamic and rapidly changing with edges being frequently inserted into or removed from graphs over time. The colorful h-star core maintenance problem is to efficiently update core numbers of affected nodes after one or a batch of edges are inserted or deleted from G. We focus on handling changes of edges since the node additions and removals can be considered as its trivial extensions. However, maintaining colorful h-star core decomposition in a dynamic setting is never an easy task. We state its challenges as follows.

a) Applying algorithms for static graphs to recompute core numbers of all nodes from scratch is more than prohibitive in terms of performance when handling large graphs and lots of edge changes.

b) Identifying nodes whose core numbers remain unchanged as a result of an insertion or deletion beforehand is quite hard. This is because after an edge is inserted or deleted, the core numbers of two end-nodes of that edge will get updated. This update will bring about updates on core numbers of the neighbors of these two end-nodes, which implies that the update may spread across the entire network, incurring predicting affected nodes intractable.

c) When $h = 2$, a colorful 2-star is exactly an edge. In this case, the colorful 2-star core decomposition turns into the classical core decomposition, which has been widely studied in [5, 11, 29]. Existing works show a very tight upper/lower bound for core numbers that after an edge is inserted into/deleted from G, the core number of each node may increase/decrease at most 1. However, this rule is no longer applicable for $h > 2$ because the changes of core numbers in this situation may exceed that bound, i.e., much larger than 1.

To tackle this problem, we develop efficient *two-stage* algorithms to update colorful h-star core numbers. We first identify a small subset of nodes that have to be visited, since not all nodes' core numbers will change. Filtering out affected nodes and limiting updates within these nodes will significantly reduce computational effort. Then in the updating

Algorithm 4: The EdgeDel algorithm

Input: A graph $G = (V, E)$, an integer h, a removed edge $(v, w) \in E$ and $c_u(G, \mathcal{S})$ for each $u \in V$
Output: The new colorful h-star core number of each node $u \in V$

1 $G^- \leftarrow (V, E \backslash \{(v, w)\})$;
2 $C_v^* \leftarrow \{c_u(G, \mathcal{S}) | u \in N_v(G)\}, C_w^* \leftarrow \{c_u(G, \mathcal{S}) | u \in N_w(G)\}$;
3 $\mathsf{H}_v^{(*)}(G, G^-) \leftarrow \mathsf{ComputeHIndex}(v, C_v^*)$;
4 $\mathsf{H}_w^{(*)}(G, G^-) \leftarrow \mathsf{ComputeHIndex}(w, C_w^*)$;
5 $c_{\mathrm{lb}}^- \leftarrow \min(\mathsf{H}_v^{(*)}(G, G^-), \mathsf{H}_w^{(*)}(G, G^-))$;
6 $c_{\mathrm{ub}}^- \leftarrow \min(c_v(G, \mathcal{S}), c_w(G, \mathcal{S}))$;
7 res $\leftarrow \mathsf{BFSwithBounds}(G^-, (v, w), c_{\mathrm{lb}}^-, c_{\mathrm{ub}}^-)$;
8 **for each** *node* $u \in$ res **in parallel do**
9 $\quad \mathsf{H}_u^{(0)}(G^-, h) \leftarrow c_u(G, \mathcal{S})$;
10 invoke Local Algorithm to compute new n-order H-indexes of nodes in res over G^- until all of them converge;
11 $c_u(G^-, \mathcal{S}) \leftarrow \mathsf{H}_u^{(n)}(G^-, h)$ for each $u \in$ res;
12 $c_u(G^-, \mathcal{S}) \leftarrow c_u(G, \mathcal{S})$ for each $u \notin$ res;
13 **return** $c_u(G^-, \mathcal{S})$ for each $u \in V$;

14 **Procedure** $\mathsf{BFSwithBounds}(G^-, (v, w), c_{\mathrm{lb}}^-, c_{\mathrm{ub}}^-)$
15 Q $\leftarrow \emptyset$, res $\leftarrow \emptyset$;
16 **if** $c_v(G, \mathcal{S}) > c_w(G, \mathcal{S})$ **then**
17 $\quad$ swap(v, w);
18 $\quad$ Q.push(v), res $\leftarrow$ res $\cup \{v\}$;
19 **if** $c_v(G, \mathcal{S}) = c_w(G, \mathcal{S})$ **then**
20 $\quad$ Q.push(w), res $\leftarrow$ res $\cup \{w\}$;
21 **while** Q $\neq \emptyset$ **do**
22 $\quad v' \leftarrow$ Q.pop();
23 $\quad$ **for each** *node* $u \in N_{v'}(G^-)$ **do**
24 $\quad\quad$ **if** $u \notin$ res *and* $c_{\mathrm{lb}}^- < c_u(G, \mathcal{S}) \leq c_{\mathrm{ub}}^-$ **then**
25 $\quad\quad\quad$ res $\leftarrow$ res $\cup \{u\}$;
26 $\quad\quad\quad$ Q.push(u);
27 **return** a collection of nodes res;

Algorithm 5: The EdgeIns algorithm

Input: A graph $G = (V, E)$, an integer h, an inserted edge (v, w) and $c_u(G, \mathcal{S})$ for each $u \in V$
Output: The new colorful h-star core number of each node $u \in V$

1 $G^+ \leftarrow (V, E \cup \{(v, w)\})$;
2 $c_{\mathrm{lb}}^+ \leftarrow \min(c_v(G, \mathcal{S}), c_w(G, \mathcal{S}))$;
3 Let H be the colorful h-star c_{lb}^+ core;
4 $H^+ \leftarrow (V_H, E_H \cup \{(v, w)\})$;
5 $c_{\mathrm{ub}}^+ = \min(d_v(H^+, \mathcal{S}), d_w(H^+, \mathcal{S}))$;
6 res $\leftarrow \mathsf{BFSwithBounds}(G^+, (v, w), c_{\mathrm{lb}}^+, c_{\mathrm{ub}}^+)$
7 **for each** *node* $u \in$ res **in parallel do**
8 $\quad \mathsf{H}_u^{(0)}(G^+, h) \leftarrow \min(d_u(H^+, \mathcal{S}), d_v(H^+, \mathcal{S}), d_w(H^+, \mathcal{S}))$;
9 invoke Local Algorithm to compute new n-order H-indexes of nodes in res over G^+ until all of them converge;
10 $c_u(G^+, \mathcal{S}) \leftarrow \mathsf{H}_u^{(n)}(G^+, h)$ for each $u \in$ res;
11 $c_u(G^+, \mathcal{S}) \leftarrow c_u(G, \mathcal{S})$ for each $u \notin$ res;
12 **return** $c_u(G^+, \mathcal{S})$ for each $u \in V$;

stage, we further accelerate the convergence by applying our Local Algorithm to compute from the original core numbers which can be utilized as a tight new bound.

In the remainder of the paper, we use $G^- = (V, E \backslash \{(v, w)\})$ and $G^+ = (V, E \cup \{(v, w)\})$ to denote the graph after a deletion/insertion of an edge (v, w) . Given two nodes u and v, we say u is reachable from v and vice versa if there exists a path $(v_s, \cdots, v_t)$ such that $u = v_s$ and $v = v_t$.

5.2 Edge Deletion

Before introducing the updating algorithm, we first show our theoretical findings. These findings could facilitate identifying a subset of nodes whose core numbers may change after removing an edge.

Theorem 7 (Nodes Exclusion Theorem). *Given a removed edge $e = (v, w)$, for any node $u \in V_G$ such that $c_u(G, \mathcal{S})$ is larger than $\min(c_v(G, \mathcal{S}), c_w(G, \mathcal{S}))$, then $c_u(G, \mathcal{S}) = c_u(G^-, \mathcal{S})$, where $\mathcal{S}$ represents a colorful h star.*

Theorem 7 indicates that core numbers of nodes will not change in the updated graph if the original core numbers of these nodes are larger than that of u or v.

In order to provide a lower bound for the changed core numbers, we first propose the instant H-index as follows.

Definition 5 (Instant H-index). *Given a node u and colorful h-star core numbers of all nodes in G, the instant H-index of u in a subgraph g, denoted by $\mathsf{H}_u^{(*)}(G, g)$, is equal to* ComputeHIndex *(u, C^*), where $C^* = \{c_v(G, \mathcal{S}) | v \in N_u(g)\}$.*

Intuitively, the instant H-index of u is to compute an H-index based on original core numbers of its neighbors in g. Obviously, if $g = G$, then $\mathsf{H}_u^{(*)}(G, G) = c_u(G, \mathcal{S})$.

Theorem 8 (Lower Bound). *Given a removed edge $e = (v, w)$, let $c_{\mathrm{lb}}^- = \min(\mathsf{H}_v^{(*)}(G, G^-), \mathsf{H}_w^{(*)}(G, G^-))$. For any node $u \in V$, if $c_u(G, \mathcal{S}) > c_{\mathrm{lb}}^-$, then the core number of u may change and the new core number $c_u(G^-, \mathcal{S}) \geq c_{\mathrm{lb}}^-$.*

By combining Theorem 7 and Theorem 8, we have the following corollary.

Corollary 1. *Given a removed edge $e = (v, w)$, for any node $u \in V$, if $\min(\mathsf{H}_v^{(*)}(G, G^-), \mathsf{H}_w^{(*)}(G, G^-)) \leq c_u(G, \mathcal{S}) \leq \min(c_v(G, \mathcal{S}), c_w(G, \mathcal{S}))$ and u is reachable from v or w, then the colorful h-star core number of u may change and the new core number is no smaller than $\min(\mathsf{H}_v^{(*)}(G, G^-), \mathsf{H}_w^{(*)}(G, G^-))$.*

Initiation of n-order H-index. According to the monotonicity of n-order H-index, the original core numbers are larger than the new core numbers in the updated graph. Hence, we can use $c_u(G, \mathcal{S})$, instead of $d_u(G, \mathcal{S})$, as an upper bound of $c_u(G^-, \mathcal{S})$ to initiate 0-order H-indexes.

Based on the above theorems, we develop an efficient algorithm EdgeDel to update core numbers after an edge deletion, the pseudocode of which is outlined in Algorithm 4. The EdgeDel algorithm first computes the lower bound and the upper bound of original core numbers of nodes that may have their core numbers changed (lines 2-6). Then this algorithm calls BFSwithBounds procedure to select out nodes according to the two bounds (line 7). After that EdgeDel initiates 0-order H-indexes of selected nodes with their core numbers in G , and iteratively computes the new n-order H-indexes of those nodes by invoking Local Algorithm (lines 8-10).

5.3 Edge Insertion

After inserting an edge, updating core numbers of affected nodes can be more challenging compared to a deletion of an edge. The reasons are twofold. 1) Our cohesive subgraph model is based on the graph coloring, which ensures any two adjacent nodes are colored with different color values. However, if two nodes v, w have the same color in the original

graph G, inserting the edge (v, w) will cause a conflict with graph coloring properties. 2) Since our n-order H-index is a monotonically decreasing function, we can use original core numbers as upper bounds of new core numbers when handling edge deletions. Nevertheless, there are no existing tight upper bounds for edge insertions. It is quite hard to utilize original core numbers to facilitate updating.

To address the first issue, we develop a *recoloring* strategy. If the two end-nodes v, w of the inserted edge have the same colors in G, we pick the node with a smaller core number and assign the minimum valid color value to this node following the greedy coloring technique such that the new color is different from colors of its neighbors in the updated graph G^+. The intuition is that since recoloring a node may change core numbers of its neighbors and the node with a smaller core number usually has a smaller degree, thus, fewer neighbors, adjusting neighbors' core numbers after recoloring will incur less computational overhead if fewer nodes get involved. Besides, the core number updating caused by recoloring can be executed along with the updating caused by the edge insertion. Thus, this strategy takes little effect on the performance of our algorithm. In the remainder of this section, we assume that the two end-nodes of the inserted edge have different colors in G for simplicity. Next, we first introduce our theoretical analyses on the identification of affected nodes.

Theorem 9 (Nodes Exclusion Theorem). *Given an inserted edge $e = (v, w)$, for any node $u \in V_G$, if $c_u(G, \mathcal{S})$ is smaller than $\min(c_v(G, \mathcal{S}), c_w(G, \mathcal{S}))$, then $c_u(G, \mathcal{S}) = c_u(G^+, \mathcal{S})$.*

Theorem 9 suggests that the nodes not contained in the colorful h-star $\min(c_v(G, \mathcal{S}), c_w(G, \mathcal{S}))$ core will not change their core numbers.

Theorem 10 (Upper Bound). *Given an inserted edge $e = (v, w)$, let $c^+_{\text{ub}} = \min(d_v(H^+, \mathcal{S}), d_w(H^+, \mathcal{S}))$. For any node $u \in V$, if $c_u(G, \mathcal{S}) < c^+_{\text{ub}}$, the core number of u may change and the new core number $c_u(G^+, \mathcal{S}) \leq c^+_{\text{ub}}$. Here H^+ is a subgraph that the colorful h-star $\min(c_v(G, \mathcal{S}), c_w(G, \mathcal{S}))$ core is inserted the edge $e = (v, w)$ into.*

Similarly, by combining Theorem 9 and Theorem 10, we can obtain the following corollary.

Corollary 2. *Given an inserted edge $e = (v, w)$, for any node $u \in V$, if $\min(c_v(G, \mathcal{S}), c_w(G, \mathcal{S})) \leq c_u(G, \mathcal{S}) \leq \min(d_v(H^+, \mathcal{S}), d_w(H^+, \mathcal{S}))$ and u is reachable from v or w, then the colorful h-star core number of u may change and the new core number is no smaller than $\min(d_v(H^+, \mathcal{S}), d_w(H^+, \mathcal{S}))$.*

Initiation of n-order H-index. Theorem 10 guarantees a good upper bound for affected nodes since the new color numbers must be smaller than c^+_{ub}. In addition, we can further accelerate the convergence of the new n-order H-index of each identified node u by providing a tighter upper bound $\min(d_u(H^+, \mathcal{S}), d_v(H^+, \mathcal{S}), d_w(H^+, \mathcal{S}))$ for the initiation of 0-order H-index. This is because by the definition of n-order H-index, in any subgraph H^+, a node's colorful h-star core number cannot exceed its colorful h-star degree.

Based on the above theoretical analyses, we develop the EdgeIns algorithm shown in Algorithm 5. Specifically, EdgeIns takes the original core numbers in G and the inserted edge (v, w) as an input. It first computes the range of core numbers of affected nodes, i.e., the lower bound and upper bound of core numbers (lines 2-5). Then EdgeIns searches nodes which are reachable from v or w and have core numbers located in that range by invoking the BFSwithBounds procedure (line 6). After identifying a collection of nodes, this algorithm initiates each node's 0-order H-Index with its distinctive upper bound (lines 7-8). Next, for nodes in res, EdgeIns calls Local Algorithm to compute their new n-order H-indexes (line 9).

Table 3: Datasets (1K=10^3, 1M=10^6, 1B=10^9)

Datasets	$n = \|V\|$	$m = \|E\|$	χ	$d_{\max}$	d_{avg}	Description
Buzznet	101.2K	2.8M	62	64.3K	55	Online social activities
Flickr	514K	3.2M	107	4.4K	12	
Digg	770.8K	5.9M	66	17.6K	15	
Orkut	3M	106.3M	79	27.5K	71	
LiveJournal	4.8M	42.9M	324	20.3K	18	
Twitter	41.6M	1.2B	1084	3.0M	57.7	
Nasasrb	54.9K	1.3M	38	275	48	Scientific computing
Pkustk	87.8K	2.6M	54	131	58	
Pwtk	217.9K	5.7M	42	179	52	
MsDoor	404.8K	9.4M	42	76	46	
LDoor	909.5K	20.8M	42	76	46	
DBLP	317.1K	1M	114	343	7	Collaboration
Skitter	1.7M	11.1M	71	35.5K	13	Internet topology
Patent	3.8M	16.5M	14	793	9	Citation

Theorem 11. *Assume that for each edge update, $\bar{n}$ nodes are affected and* EdgeDel/ EdgeIns *takes t iterations to converge. Then, the time complexity is given by $O(\bar{n} \times t \times h)$.*

6 EXPERIMENTS

In the following, we first describe the experimental setup, and then report the results of Local Algorithm and its optimizations. After that we investigate the performance variation of our updating algorithms over dynamic graphs.

6.1 Experimental Setup

Datasets. We collect 14 large real-world graphs from two different sources (1) Network Repository[1] and (2) Stanford Network Analysis Project[2] (SNAP). These datasets cover various domains, such as online social networks, scientific computing networks, collaboration networks, Internet topology graphs and citation networks. Table 3 summarizes the detailed statistics of each dataset. In Table 3, χ denotes the total number of different colors used by the greedy graph coloring algorithm. $d_{\max}$ and d_{avg} represent the maximum degree and the average degree respectively.

Experimental environment. All our proposed algorithms are implemented in GNU C++ and parallelized using OpenMP API. We conduct all experiments on a Linux server equipped with a 2.9GHz AMD Ryzen 3990X CPU with 64 cores and 256GB RAM running Ubuntu 22.04 (64-bit).

Parameters. In experiments, by default we will use a single thread for the sequential versions of algorithms, and use up

[1]https://networkrepository.com/
[2]http://snap.stanford.edu/data/

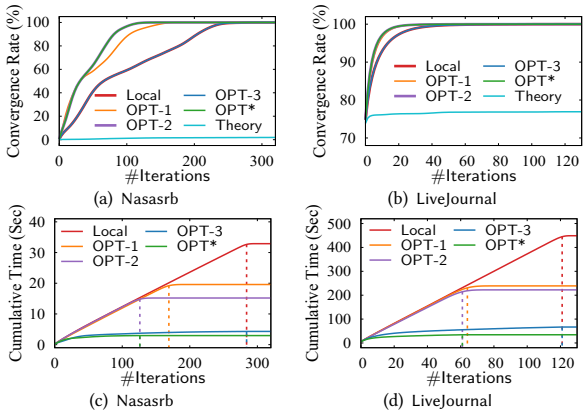


Figure 3: Convergence rate and cumulative time

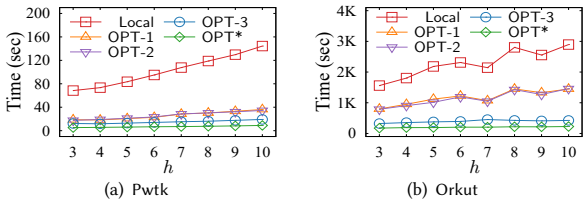


Figure 4: Running time with varying h from 3 to 10

to 64 threads when evaluating the degree of parallelism. Unless otherwise specified, we evaluate all algorithms for $h = 10$. Similar results can be observed for other values of h.

6.2 Effectiveness Evaluation

In this experiment, we evaluate the convergence of proposed algorithms in different datasets.

Algorithms. We implement the Local Algorithm Local in Section 4.1 and its three optimizations OPT-1,OPT-2 and OPT-3 in Section 4.2. We also develop OPT∗ by integrating these three optimizations with Local Algorithm. Theory represents the theoretical upper bound of the number of iterations.

Overview: Convergence evaluation. We report the convergence results of different algorithms in 14 graphs for $h = 10$ in Table 4. Table 4 considers three metrics: the number of iterations, average invocations and running time, where Average Invocations is defined as $\frac{\sigma}{|V|}$, and σ is the total number of n-order H-index calls, i.e., invoking ComputeHIndex procedure. It is obvious that our proposed algorithms go through far fewer iterations compared to the theoretical bound. The two *inter*-iteration optimizations OPT-1 and OPT-2 further reduce almost half of iterations of Local in all datasets. Note that OPT-1 and OPT-2 have the comparative reduction ability, and in most graphs our degree-based heuristic approach OPT-2 performs much better than OPT-1. In terms of pruning ability, our *intra*-iteration optimization OPT-3 avoids at least 90% unnecessary n-order H-index calls of Local in all networks except Patent. By combining OPT-1 and OPT-2, the reduction of OPT∗ can even reach 95% in most datasets. It is worth mentioning that in some graphs (e.g. Flickr, Digg, DBLP, Skitter and Patent), the invocation ratios are less than 100%, suggesting that the average number of procedure calls per node is less than 1. Additionally, our best method OPT∗ outperforms Local by at least one order of magnitude in all 14 graphs. The results are consistent with invocation ratios since each procedure call takes similar time.

Into Iterations: Convergence rate and cumulative time. Fig. 3 presents the detailed convergence statistics to demonstrate the effectiveness of our two types of optimizations. Convergence rate and cumulative time of iteration i indicates the percentage of converged nodes and total running time after the first i iterations respectively. These two metrics reflect how fast our proposed algorithms converge and time distribution along with iterations. From Fig. 3(a) and Fig. 3(b) we can observe that as the number of iterations increases, the percentage of converged nodes raises as well. Moreover, our Local Algorithm and its optimized variants show sharper increases compared to the steady but slow Theory. Besides, OPT-2 and OPT∗ benefiting from asynchronous and order-based processing strategies, achieve a faster convergence than others. Fig. 3(c) and Fig. 3(d) show the time consumption per iteration of our five algorithms where dashed lines mark the iteration that all nodes converge. It is clear that the cumulative time of Local, OPT-1 and OPT-2 increases almost linearly with the number of iterations. However, the pruning-equipped methods OPT-3 and OPT∗ show a totally different trend where time consumption per iteration drops rapidly after the first few iterations. This is because computations on the converged nodes can be identified and then avoided.

6.3 Efficiency Evaluation

The effect of h. We first study how the performance of our five algorithms is affected by different h values in Fig. 4. As expected, though all algorithms have larger time costs with an increasing h on two graphs, Local is consistently the slowest for all h values. The reason could be that the ComputeHIndex procedure will take more time for a larger h. After applying our two optimizations to reduce the number of iterations, OPT-1 and OPT-2 achieve much better performance. In addition, the two algorithms can be further improved by using our pruning technique, as shown in the results of OPT-3 and OPT∗ where the time consumption is more stable with the increase of h. It is clear that for $h = 10$, our best performing method OPT∗ is even 16× and 13× faster than the basic Local on Pwtk and Orkut respectively. Note that following the trend of running time, our algorithms are highly competent to compute the core decomposition even for $h > 10$.

Degree of parallelism. We evaluate the degree of parallelism of our proposed algorithms and Peel by varying the number of threads in Fig. 5. Here Peel is a parallel version of the peeling algorithm by simply updating colorful h-star degrees of neighbors in parallel after deleting a node. As can be seen, the performance of Peel gets slightly worse when using even more threads. The reasons are twofold: 1) After

Table 4: Convergence Evaluation w.r.t Iterations, n-order H-index calls and Running Time ($h = 10$)

Datasets	#Iterations						Average Invocations					Time (sec)				
	Theory	Local	OPT-1	OPT-2	OPT-3	OPT∗	Local	OPT-1	OPT-2	OPT-3	OPT∗	Local	OPT-1	OPT-2	OPT-3	OPT∗
Buzznet @	30234	66	35	33	66	33	33.76	17.90	16.88	1.81	1.19	16	8.6	7.9	4.5	2.4
Flickr	24686	85	45	45	85	45	8.20	4.34	4.34	0.59	0.35	19.6	10.6	10	4.9	2.6
Digg	39341	83	44	40	83	40	8.21	4.35	3.96	0.55	0.34	36.7	20.2	17.6	11.7	6.2
Orkut	1245333	237	117	119	237	119	197.53	97.52	99.19	9.58	5.33	2860.5	1422.1	1456.8	425.4	227
LiveJournal	397783	122	65	62	122	62	29.21	15.57	14.86	1.72	1.02	455	244.2	229.1	72.5	38.9
Twitter	4768506	141	73	71	141	71	56.21	29.1	28.31	0.89	0.68	31540.1	15280.1	14170.2	2477.5	1274.7
Nasasrb	16797	285	170	127	285	127	284.52	169.72	126.79	26.94	17.49	32.8	19.5	15.2	4.3	3
Pkustk	5032	91	43	39	91	39	91.00	43.00	39.00	7.15	5.48	20.5	10	9.1	2.6	2
Pwtk	15421	289	74	69	289	69	288.39	73.84	68.86	28.94	14.50	144.3	36	34.5	19.1	8.9
MsDoor	27107	166	88	93	166	93	166.00	88.00	93.00	6.14	4.67	137.9	74.1	79.1	8.5	6.5
LDoor	77066	235	120	122	235	122	235.00	120.00	122.00	7.19	5.54	434.9	224.7	231.9	23.5	18.4
DBLP	880	30	17	19	30	19	1.89	1.07	1.20	0.11	0.09	1	0.6	0.6	0.2	0.2
Skitter	46157	79	43	43	79	43	6.36	3.46	3.46	0.30	0.21	33.6	18.9	18.5	6.8	4.1
Patent	1492	69	37	37	69	37	0.06	0.04	0.04	0.02	0.02	3.6	3.2	2.2	3.2	2.1

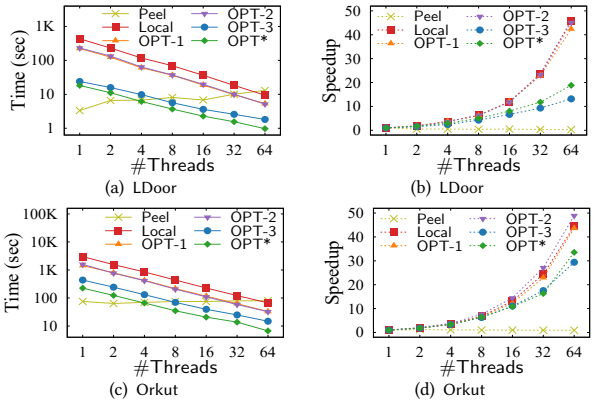


Figure 5: Parallelization Performance Evaluation

deleting a node, parallelizing the update of Peel could introduce extra time overhead, such as cache misses. 2) The sequential version of Peel, i.e., using only 1 thread, can well leverage the space locality since neighbors are stored in an adjacent array in the CSR format. In contrast, our algorithms all achieve a significantly high degree of parallelism. For example, OPT-2 with 64 threads is 45× and 49× faster than its sequential version. Moreover, the speedup of our two pruning-equipped algorithms OPT-3 and OPT∗ is not always as high as other methods. It is because that in the last few iterations, most nodes have converged and computations on them are pruned, which incurs that many threads keep idle and therefore limit the degree of parallelism.

6.4 Update Evaluation

Algorithms and settings. Since the previous experiments have demonstrated the superiority of our best algorithm OPT∗, we omit the evaluation of other algorithms in this experiment. Here we consider two types of algorithms: 1) ReComp. For each edge update, this algorithm calls OPT∗ to recompute core numbers of all nodes. 2) EdgeDel and EdgeIns. Two algorithms only compute core numbers on identified nodes. For edge deletions, ReComp and EdgeDel initiate the new 0-order H-indexes both with original core numbers. For edge insertions, ReComp takes the new colorful h-star degrees

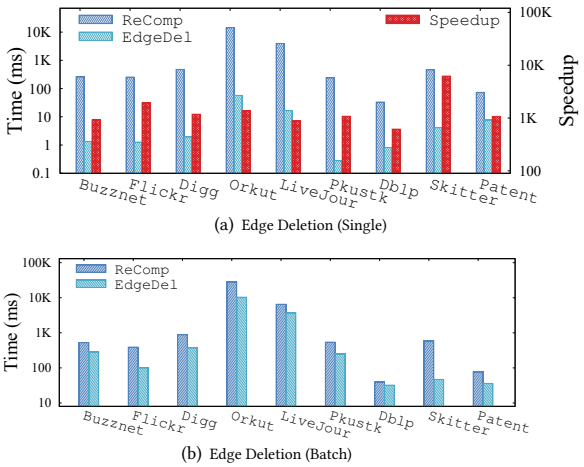
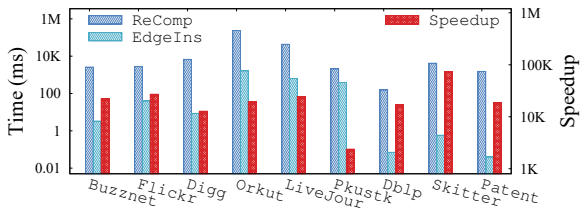


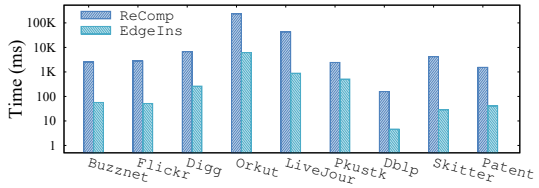
Figure 6: Update Time and Speedup for Edge Deletions

in the updated graph as the initiation, but EdgeIns will use our proposed upper bounds. Here we consider two kinds of updates: a single edge update and a batch of edge updates. For the batch updates, we sample 100 edges uniformly at random from the original graph as edge updates. We invoke ReComp for the entire batch of updates, while in contrast, we run EdgeDel and EdgeIns to handle each update one by one.

Results for the edge deletion. We compare two edge deletion algorithms in terms of updating time for computing new core numbers on 9 datasets in Fig. 6. We can observe that EdgeDel is significantly faster than ReComp, reaching 2-3 orders of magnitude speedup on all graphs. For instance, on Skitter and Flickr, ReComp takes 6224× and 1927× updating time than EdgeDel. It is worth mentioning that the speedup can be used as an important indicator for choosing an appropriate algorithm when handling different number of edge deletions. To be more specific, the speedup on Orkut is 1369, which means if there is less than 1369 edges to be sequentially deleted from the original graph at a time interval, we recommend invoking EdgeDel multiple times to handle each edge update independently. If the number of edge deletions



(a) Edge Insertion (Single)



(b) Edge Insertion (Batch)

Figure 7: Update Time and Speedup for Edge Insertions

is larger than that speedup, then ReComp is preferable since it will be more efficient. Fig. 6(b) shows the competence of EdgeDel when dealing with a batch of edge updates. Note that the performance gap between ReComp and EdgeDel narrows in this case because after 100 edges are deleted, it only needs to call OPT∗ once to update core numbers, rather than summing up the running time of each edge deletion as EdgeDel does. Even so, EdgeDel still achieves two to three times improvement on most graphs.

Results for the edge insertion. Fig. 7(a) and Fig. 7(b) report the results for a single edge insertion and a batch of edge insertions respectively. As Fig. 7(a) shows, EdgeIns is at least four orders of magnitude faster than ReComp on all datasets except Pkustk. Moreover, the average updating time on DBLP, Skitter and Patent is all less than 1 millisecond, which does not surprise us since as we all know, real-world graphs are usually sparse and most nodes of them have smaller colorful h-star degrees. Thus, in most cases inserting an edge will affect a small number of nodes that can be identified by EdgeIns. EdgeIns remains its high efficiency in the test of batch insertions in Fig. 7(b), outperforming ReComp on all networks. Another interesting observation is that EdgeIns can achieve a higher speedup compared to EdgeDel over ReComp when handling a batch of edge updates. This is because EdgeIns benefits from our well-developed upper bounds that are much tighter than the bounds ReComp uses.

Skewed updates on graphs with skewed structures. Here we use skewed updates to evaluate the worse-case performance of our algorithms. Instead of sampling edge updates uniformly at random, we collect the 100 most skewed edge updates from the graph, where deleting or inserting each of them will result in the maximum number of affected nodes. Additionally, we use two graphs with skewed structures to demonstrate the efficiency of our algorithms on real-world graphs (i.e. following power-low degree distribution). The results are presented in Table 5. As observed, the skewed deletions have a slight impact on the number of affected nodes, compared to the significant increase caused by skewed insertions on real-world graphs (e.g., 1.2% increase for deletions

Table 5: Average updating time (millisecond) per random and skewed edge deletion/insertion for graphs with power-law and skewed degree distributions using 32 threads (%: proportion of affected nodes)

Dataset	Type	Delete					Insert				
		ReComp	Random		Skew		ReComp	Random		Skew	
			%	EdgeDel	%	EdgeDel		%	EdgeIns	%	EdgeIns
Skitter	Power law	183.2	*0.1*	0.11	*1.2*	11.6	686.9	*1.9*	12	*94.0*	559.3
Digg		101.5	*1.8e-5*	0.12	*0.5*	1.7	447.8	*0.4*	11.6	*39.2*	70.1
Twitter		44K	*2.4e-7*	14.7	*1.9e-6*	265.3	107K	*1.2e-5*	21.5	*31.5*	11K
Pwtk	Skew	122.2	*1.2*	0.8	*6.2*	3.8	806.6	*84.4*	665.7	*87.9*	837.2
MsDoor		267.2	*0.7*	1.1	*6.4*	10.1	508.5	*35.8*	182.2	*58.9*	243.6

and 92.1% increase for insertions on Skitter). The reason is that the upper bound for the new core number is still much larger than the old one. Hopefully, our experiment indicates that this type of skewed edges only accounts for a minor portion. On the skewed graphs, the results show a big difference. The skewed deletions and insertion both affect a slightly larger number of nodes (e.g., 5% and 3.5% increase in deletions and insertions respectively on Pwtk). This occurs because, in the skewed structure, most nodes have large and similar core numbers. Therefore, though the bounds we obtain for new core numbers are tight, there are still a large number of nodes that fall within the range of bounds and old core numbers.

6.5 Applications

We empirically study the significance of the colorful h-star core model and its local algorithm in higher-order cohesive subgraph mining. Other applications can be found in [16].

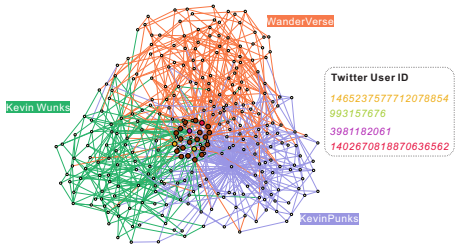
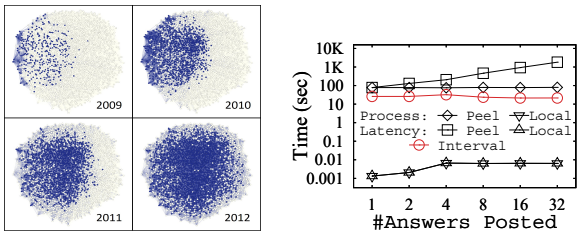


Figure 8: Higher-order structure in NFT comunities

Application 1. Identification of higher-order structures in NFT communities. NFT communities are groups of people who share an interest in Non-Fungible Tokens (NFTs) and engage in activities related to their creation, collection, and trading. Identifying higher-order structures within NFT communities allows stakeholders to better comprehend the intricate relationships and dynamics of the ecosystem, make informed decisions and recognize opportunities and risks. We analyzed an NFT transaction network from the first week of March 2022 on the Ethereum blockchain. Fig. 8 shows the maximal colorful 3-star core of the network, revealing three active communities with the most frequent transactions: "WanderVerse" for Play-to-Earn gaming, and "Kevin Wunks" and "KevinPunks" for trading social media profile pictures (PFPs). Our model further identified a higher-order group, where each member participates in all three communities. By



(a) Active user distribution in four years (b) Average time for temporal edge insertions

Figure 9: The evolution of Stack Overflow and performance comparison for maintaining colorful h-star cores

combining public information from online social media platforms, such as Twitter, and Ethereum Name, we infer certain personal details. Upon analyzing their tweets, we discovered that the higher-order group members are genuinely interested in PFPs and PC gaming, and maintain strong social relationships, which highlights potential investment opportunities and mitigates concerns about money laundering.

Application 2. Revelation of active users in social networks and real-time capture of graph changes. Here we discuss the application of our model in dynamic graphs. We construct an undirected temporal graph of a public Q&A platform Stack Overflow from the Stack Exchange Data Dump[3]. In this temporal graph, each edge (u, v) associated with a timestamp t represents user u answered a question asked by user v at time t. Fig. 9(a) shows the distribution of users over four years (only displays the dense part). Users marked in blue lie in the colorful h-star maximum core of each year, and they are usually the potential influential ones who actively give answers or post questions related to hot topics. We can clearly see that our model is excellent at detecting newly joined active users in different epochs. The efficiency of this detection is demonstrated in Fig. 9(b). In this figure, we observe that an answer is generated (an edge is inserted) at intervals of about 20 seconds, but the classical Peel approach takes approximately 80 seconds to handle this graph change, which incurs a very high average latency for each update. By contrast, the Local algorithm takes at most 0.01 second for any insertion, significantly faster than the answer generation. Due to this feature, the local algorithm enables a real-time detection of influential users in dynamic social networks.

7 RELATED WORK

Higher-Order Core Decomposition. Unlike the classical k-core [22, 38] which only considers the simple structure of nodes and adjacent edges, recently a large number of higher-order core models have been proposed to identify the cohesive subgraph with higher-order Information. Notable examples include k-truss [7, 12, 19, 39] which considers the involvement of edges and triangles, triangle k-core [30, 31, 49] which focuses on the structure of nodes and triangles they participate in, and their non-trivial generalization k-(r, s)-nucleus [36, 37]. k-(r, s)-nucleus treats every simple structure as a clique, i.e., 1-clique for a node and 2-clique for an edge ,and stipulates that in its k-core, each r-clique is contained in at least k s-cliques [35]. Among all combinations of r, s values, h-clique core with $r = 1, s = h$ received more attention. It was first proposed by Fang et al. [14] to provide an approximation for k-clique densest subgraph problem [42]. The generalization for higher-order structures is not limited within a node and its neighbors. Francesco et al. [8] proposed the distance generalized core, also called (k, h)-core, to consider the structural information of h-hop neighborhood. Liao et al. [23] has conducted research on the problem of D-core decomposition in distributed settings by introducing the concept D-index. Even almost all higher-order core models have a peeling approach to compute their core decomposition, they are not appealing since they either require too much computational effort, or have limited applications.

Core Maintenance. Core maintenance is to update core numbers after the graph structure is changed, and has usually been done locally, i.e., without recomputing new core numbers for all nodes. Li et al. developed pruning strategies to maintain the k-core numbers in a dynamic graph [22]. Similarly, Ahmet et al. proposed the first incremental k-core decomposition algorithms for streaming graph data [34]. Recently, Yu et al. studied the problem of processing multiple edges concurrently [47]. Lin et al. [24] maintained k-cores by taking the connections among different k-cores into consideration. Besides, an I/O efficient algorithm following a semi-external model, which only allows node information to be loaded in memory, has been devised by Wen et al. to study the out-memory core maintenance [44]. Parallel algorithms for dynamic graphs have been developed by Hua et al. [18]. For higher-order core maintenance, Liu et al. [25] adopted the n-order H-index to design an algorithm for (k, h)-core maintenance. Besides, Bai et al. [3] presented a bottom-up algorithm on dynamic bipartite graphs.

8 CONCLUSION

In this paper, we revisit the colorful h-star core decomposition problem. We first introduce a new concept of colorful h-star n-order H-index and study its properties through thorough theoretical analyses. Based on the n-order H-index, we propose a local algorithm and three optimizations to address the low degree of parallelism of the existing methods. The decomposition in dynamic graphs can be efficiently maintained by extending our local algorithm. The experimental results on 14 large real-world graphs demonstrate the efficiency and effectiveness of our proposed algorithms.

ACKNOWLEDGMENTS

This work was partially supported by (i) the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG2-TC-2021-002), (ii) National Key Research and Development Program of China 2020AAA0108503, (iii) NSFC Grants U2241211, 62072034, U1809206 and (iv) CCF-Huawei Populus Grove Fund. Rong-Hua Li is the corresponding author of this paper.

[3]https://archive.org/details/stackexchange

REFERENCES

[1] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani. Large scale networks fingerprinting and visualization using the k-core decomposition. In *NIPS*, pages 41–50, 2005.

[2] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani. K-core decomposition of internet graphs: hierarchies, self-similarity and measurement biases. *NHM*, 3(2):371–393, 2008.

[3] W. Bai, Y. Chen, D. Wu, Z. Huang, Y. Zhou, and C. Xu. Generalized core maintenance of dynamic bipartite graphs. *Data Min. Knowl. Discov.*, 36(1):209–239, 2022.

[4] B. Balasundaram and S. Butenko. Graph domination, coloring and cliques in telecommunications. In *Handbook of Optimization in Telecommunications*, pages 865–890. Springer, 2006.

[5] V. Batagelj and M. Zaversnik. An O(m) algorithm for cores decomposition of networks. *CoRR*, cs.DS/0310049, 2003.

[6] A. R. Benson, D. F. Gleich, and J. Leskovec. Higher-order organization of complex networks. *Science*, 353(6295), 2016.

[7] F. Bonchi, F. Gullo, A. Kaltenbrunner, and Y. Volkovich. Core decomposition of uncertain graphs. In *KDD*, 2014.

[8] F. Bonchi, A. Khan, and L. Severini. Distance-generalized core decomposition. In P. A. Boncz, S. Manegold, A. Ailamaki, A. Deshpande, and T. Kraska, editors, *SIGMOD*, 2019.

[9] S. Carmi, S. Havlin, S. Kirkpatrick, Y. Shavitt, and E. Shir. A model of internet topology using k-shell decomposition. *PNAS*, 104(27):11150–11154, 2007.

[10] J. Chen and Y. Saad. Dense subgraph extraction with application to community detection. *IEEE Transactions on knowledge and data engineering*, 24(7):1216–1230, 2010.

[11] J. Cheng, Y. Ke, S. Chu, and M. T. Özsu. Efficient core decomposition in massive networks. In *ICDE*, 2011.

[12] J. Cohen. Trusses: Cohesive subgraphs for social network analysis. *Technical report, National Security Agency*, 2005.

[13] W. Fan, J. Xu, Y. Wu, W. Yu, and J. Jiang. GRAPE: parallelizing sequential graph computations. *Proc. VLDB Endow.*, 10(12):1889–1892, 2017.

[14] Y. Fang, K. Yu, R. Cheng, L. V. Lakshmanan, and X. Lin. Efficient algorithms for densest subgraph discovery. *Proc. VLDB Endow.*, 12(11):1719–1732, 2019.

[15] S. Gao, R. Li, H. Qin, H. Chen, Y. Yuan, and G. Wang. Colorful h-star core decomposition. In *ICDE*, pages 2588–2601. IEEE, 2022.

[16] S. Gao, H. Qin, R.-H. Li, and B. He. Parallel colorful h-star core maintenance in dynamic graphs. 2023. available at: https://github.com/Gawssin/ColorfulStarLocal/blob/main/FullVersion.pdf.

[17] W. Hasenplaugh, T. Kaler, T. B. Schardl, and C. E. Leiserson. Ordering heuristics for parallel graph coloring. In *SPAA*, 2014.

[18] Q. Hua, Y. Shi, D. Yu, H. Jin, J. Yu, Z. Cai, X. Cheng, and H. Chen. Faster parallel core maintenance algorithms in dynamic graphs. *IEEE Trans. Parallel Distributed Syst.*, 31(6):1287–1300, 2020.

[19] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. *SIGMOD*, 2014.

[20] X. Huang, L. V. Lakshmanan, and J. Xu. Community search over big graphs: Models, algorithms, and opportunities. In *ICDE*, 2017.

[21] M. Kitsak, L. K. Gallos, S. Havlin, F. Liljeros, L. Muchnik, H. E. Stanley, and H. A. Makse. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, 4:2, 2003.

[22] R.-H. Li, J. Yu, and R. Mao. Efficient core maintenance in large dynamic graphs. *IEEE Trans. Knowl. Data Eng.*, 26(10):2453–2465, 2014.

[23] X. Liao, Q. Liu, J. Jiang, X. Huang, J. Xu, and B. Choi. Distributed d-core decomposition over large directed graphs. *Proc. VLDB Endow.*, 15(8):1546–1558, 2022.

[24] Z. Lin, F. Zhang, X. Lin, W. Zhang, and Z. Tian. Hierarchical core maintenance on large dynamic graphs. *Proc. VLDB Endow.*, 14(5):757–770, 2021.

[25] Q. Liu, X. Zhu, X. Huang, and J. Xu. Local algorithms for distance-generalized core decomposition over large dynamic graphs. *Proc. VLDB Endow.*, 14(9):1531–1543, 2021.

[26] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. Distributed graphlab: A framework for machine learning in the cloud. *Proc. VLDB Endow.*, 5(8):716–727, 2012.

[27] L. Lü, T. Zhou, Q.-M. Zhang, and H. E. Stanley. The h-index of a network node and its relation to degree and coreness. *Nature communications*, 7(1):1–7, 2016.

[28] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD Conference*, pages 135–146. ACM, 2010.

[29] A. Montresor, F. D. Pellegrini, and D. Miorandi. Distributed k-core decomposition. *IEEE Trans. Parallel Distrib. Syst.*, 24(2):288–300, 2013.

[30] R. Rossi, D. Gleich, and A. Gebremedhin. Triangle core decomposition and maximum cliques. *SIAM Workshop on Network Science*, 01 2013.

[31] R. A. Rossi. Fast triangle core decomposition for mining large graphs. volume 8443 of *Lecture Notes in Computer Science*, pages 310–322. Springer, 2014.

[32] S. Salihoglu and J. Widom. GPS: a graph processing system. In *SSDBM*, pages 22:1–22:12. ACM, 2013.

[33] R. Samudrala and J. Moult. A graph-theoretic algorithm for comparative modeling of protein structure. *Journal of molecular biology*, 279(1):287–302, 1998.

[34] A. E. Sariyüce, B. Gedik, G. Jacques-Silva, K.-L. Wu, and Ü. V. Çatalyürek. Streaming algorithms for k-core decomposition. *PVLDB*, 6(6):433–444, 2013.

[35] A. E. Sariyuce and A. Pinar. Fast hierarchy construction for dense subgraphs. *Proc. VLDB Endow.*, 10(3):97–108, 2016.

[36] A. E. Sariyüce, C. Seshadhri, A. Pinar, and Ü. V. Çatalyürek. Finding the hierarchy of dense subgraphs using nucleus decompositions. In *WWW*, 2015.

[37] A. E. Sariyüce, C. Seshadhri, A. Pinar, and Ü. V. Çatalyürek. Nucleus decompositions for identifying hierarchy of dense subgraphs. *TWEB*, 11(3):16:1–16:27, 2017.

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

[39] B. Sun, M. Danisch, T. Chan, and M. Sozio. Kclist++: A simple algorithm for finding k-clique densest subgraphs in large graphs. *Proceedings of the VLDB Endowment (PVLDB)*, 2020.

[40] Y. Tian, A. Balmin, S. A. Corsten, S. Tatikonda, and J. McPherson. From "think like a vertex" to "think like a graph". *Proc. VLDB Endow.*, 7(3):193–204, 2013.

[41] C. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. Tsiarli. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In *KDD*, 2013.

[42] C. E. Tsourakakis. The k-clique densest subgraph problem. In *WWW*, 2015.

[43] T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. J. van Wijk, J.-D. Fekete, and D. W. Fellner. Visual analysis of large graphs: State-of-the-art and future research challenges. *Comput. Graph. Forum*, 30(6):1719–1749, 2011.

[44] D. Wen, L. Qin, Y. Zhang, X. Lin, and J. X. Yu. I/O efficient core graph decomposition at web scale. In *ICDE*, pages 133–144, 2016.

[45] S. Wuchty and E. Almaas. Peeling the yeast protein network. *Proteomics*, 5, 2005.

[46] D. Yan, J. Cheng, Y. Lu, and W. Ng. Blogel: A block-centric framework for distributed computation on real-world graphs. *Proc. VLDB Endow.*, 7(14):1981–1992, 2014.

[47] D. Yu, N. Wang, Q. Luo, F. Li, J. Yu, X. Cheng, and Z. Cai. Fast core maintenance in dynamic graphs. *IEEE Transactions on Computational Social Systems*, 9(3):710–723, 2022.

[48] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang. Effective and efficient dynamic graph coloring. *PVLDB*, 11(3):338–351, 2017.

[49] Y. Zhang and S. Parthasarathy. Extracting, analyzing and visualizing triangle k-core motifs within networks. In *ICDE*, 2012.