Neighborhood-based Hypergraph Core Decomposition

Naheed Anjum Arafat
NUS
Singapore
naheed_anjum@u.nus.edu

Arijit Khan
AAU
Denmark
arijitk@cs.aau.dk

Arpit Kumar Rai
IIT Kanpur
India
arpitkr20@iitk.ac.in

Bishwamittra Ghosh
NUS
Singapore
bghosh@u.nus.edu

ABSTRACT

We propose *neighborhood-based core decomposition*: a novel way of decomposing hypergraphs into hierarchical neighborhood-cohesive subhypergraphs. Alternative approaches to decomposing hypergraphs, e.g., reduction to clique or bipartite graphs, are not meaningful in certain applications, the later also results in inefficient decomposition; while existing degree-based hypergraph decomposition does not distinguish nodes with different neighborhood sizes. Our case studies show that the proposed decomposition is more effective than degree and clique graph-based decompositions in disease intervention and in extracting provably approximate and application-wise meaningful densest subhypergraphs. We propose three algorithms: **Peel**, its efficient variant **E-Peel**, and a novel local algorithm: **Local-core** with parallel implementation. Our most efficient parallel algorithm **Local-core(P)** decomposes hypergraph with 27M nodes and 17M hyperedges in-memory within 91 seconds by adopting various optimizations. Finally, we develop a new hypergraph-core model, the *(neighborhood, degree)-core* by considering both neighborhood and degree constraints, design its decomposition algorithm **Local-core+Peel**, and demonstrate its superiority in spreading diffusion.

PVLDB Reference Format:
Naheed Anjum Arafat, Arijit Khan, Arpit Kumar Rai, and Bishwamittra Ghosh. Neighborhood-based Hypergraph Core Decomposition. PVLDB, 16(9): 2061 - 2074, 2023.
doi:10.14778/3598581.3598582

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

1 INTRODUCTION

Decomposition of a graph into hierarchically cohesive subgraphs is an important tool for solving many graph data management problems, e.g., community detection [44], densest subgraph discovery [14], identifying influential nodes [43], and network visualization [1, 8]. Depending on different notions of cohesiveness, there are several decomposition approaches: core-decomposition [9], truss-decomposition [54], nucleus-decomposition [48], etc. In this work, we are interested in decomposing hypergraphs, a generalization of graphs where an edge may connect more than two entities.

Many real-world relations consist of polyadic entities, e.g., relations between individuals in co-authorships [29], legislators in

Proceedings of the VLDB Endowment, Vol. 16, No. 9 ISSN 2150-8097.
doi:10.14778/3598581.3598582

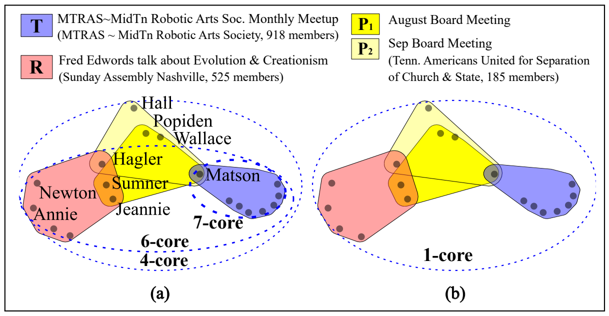

T
MTRAS~MidTn Robotic Arts Soc. Monthly Meetup
(MTRAS ~ MidTn Robotic Arts Society, 918 members)
R
Fred Edwords talk about Evolution & Creationism
(Sunday Assembly Nashville, 525 members)
P1
August Board Meeting
P2
Sep Board Meeting
(Tenn. Americans United for Separation of Church & State, 185 members)
Hall
Popiden
Wallace
Hagler
Sumner
Matson
Newton
Annie
Jeannie
7-core
6-core
4-core
1-core
(a)
(b)

Figure 1: (a) Neighborhood-based and (b) degree-based core decomposition of hypergraph H

parliamentary voting [11], items in e-shopping carts [56], proteins in protein complexes and metabolites in a metabolic process [22, 25]. For convenience, such relations are often reduced to a clique graph or a bipartite graph (§2.2). However, these reductions may not be desirable due to two reasons. First, such reductions might not be meaningful, e.g., a pair of proteins in a certain protein complex may not necessarily interact pairwise to create a new functional protein complex. Second, reducing a hypergraph to a clique graph or a bipartite graph inflates the problem-size [32]: A hypergraph in [57] with 2M nodes and 15M hyperedges is converted to a bipartite graph with 17M nodes and 1B edges. A k-uniform hypergraph with m hyperedges causes its clique graph to have $O(mk^2)$ edges. The bipartite graph representation also requires distance-2 core decomposition [13, 41], which is more expensive due to inflated problem-size (§6).

To this end, we propose a novel neighborhood-cohesion based hypergraph core decomposition that decomposes a hypergraph into nested, strongly-induced maximal subhypergraphs such that all the nodes in every subhypergraph have at least a certain number of neighbors in that subhypergraph. Being strongly-induced means that a hyperedge is only present in a subhypergraph if and only if all its constituent nodes are present in that subhypergraph.

Example 1 (Neighborhood-based core decomposition). *The hypergraph H in Figure 1(a) is constructed based on four events (T, R, P_1, P_2) from the* Nashville Meetup Network *dataset [7]. Each hyperedge denotes an event, nodes in a hyperedge are participants of that event. Two nodes are neighbors if they co-participate in an event. We notice that Hall has 4 neighbors: Popiden, Hagler, Wallace, and Matson. Similarly, Matson, Hagler, Jeannie, Sumner, Annie, Newton, Wallace, and Popiden have $13, 10, 9, 9, 6, 6, 6,$ and 6 neighbors, respectively. As every node has ≥ 4 neighbors, the neighborhood-based 4-core, denoted by $H[V_4]$, is the hypergraph H itself. The neighborhood based 6-core is the subhypergraph $H[V_6] = \{T, R\}$ because participants of T (e.g., Newton, Annie) and R (e.g., Matson)* *respectively have* 6 *and* 7 *neighbors in* $H[V_6]$. *Finally, the neighborhood based* 7*-core is the subhypergraph* $\{T\}$.

Motivation. The only hypergraph decomposition existing in the literature is that based on degree (i.e., number of hyperedges incident on a node) [46, 50]. In *degree-based core decomposition*, every node in the k-core has degree at least k in that core. It does not consider hyperedge sizes. As a result, nodes in the same core may have vastly different neighborhood sizes. For instance, *Hall* and *Matson* have 4 and 13 neighbors, respectively, yet they belong to the same 1-core in the degree-based decomposition (Figure 1(b)). There are applications, e.g., propagation of contagions in epidemiology, diffusion of information in viral-marketing, where it is desirable to capture such differences because nodes with the same number of neighbors in a subhypergraph are known to exhibit similar diffusion characteristics [36]. Indeed from a practical viewpoint, if the infectious diseases control authority is looking for some key events that cause higher infection spread (e.g., meeting with 6 neighbors per participant) and hence such events need to be intervened, they are events $\{T, R\}$ (nbr-6-core) as they cause meeting with at least 6 neighbors per participant. Such distinction across various events and participants is not possible according to degree-based core decomposition of H. Moreover, the neighborhood based decomposition is also logical: innermost core contains T, which is a tech event organized by a relatively popular group with 918 members, followed by the second innermost core containing $\{R, T\}$. R is an event organized by a secular organization with 525 members which often discusses religion matters. The outermost core contains two events held by a niche social activist group with only 185 members.

Challenges. A hyperedge can relate to more than two nodes, and a pair of nodes may be related by multiple, distinct hyperedges. Thus, a trivial adaptation of core-decomposition algorithms for graphs to hypergraphs is difficult. In the classic peeling algorithm for graph core decomposition, when a node is removed, the degree of its neighbors is decreased by 1: this allows important optimizations which makes the peeling algorithm *linear-time* and efficient [9, 15, 47]. However, in a neighborhood-based hypergraph core decomposition, deleting a node may reduce the neighborhood size of its neighboring node by more than 1. Hence, to recompute the number of neighbors of a deleted node's neighbor, one must construct the residual hypergraph after deletion, which makes the decomposition *polynomial-time* and expensive. Furthermore, the existing lower bound that makes graph core decomposition more efficient [13] does not work for neighborhood-based hypergraph core decomposition.

In the following, we also discuss the challenges associated with adopting the local approach [42], one of the most efficient methods for graph core decomposition to hypergraphs. In a local approach, a core-number estimate is updated iteratively [35, 42] or in a distributed manner [45] for every node in a graph. The initial value of a node's core-number estimate is an upper bound of its core-number. In subsequent rounds, this estimate is iteratively decreased based on estimates of neighboring nodes. [42] uses h-index [30] for such update. They have shown that the following invariant must hold: *every node with core-number k has h-index at least k*, and *the subgraph induced by nodes with h-index at least k has at least k neighbors per node in that subgraph*. The former holds but the later may not hold in a hypergraph, because the subhypergraph induced by nodes

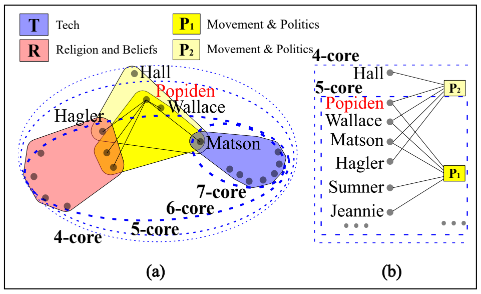


Figure 2: Alternative decompositions are sometimes undesirable: (a) Core decomposition of clique graph of H and Dist-2 core decomposition of the bipartite graph of H produce the same decomposition. Similar events (P_1 and P_2) are in the different cores. (b) Bipartite graph representation of H.

with h-index $\geq k$ may not include hyperedges that partially contain other nodes (§3.3). Due to those 'missing' hyperedges, the number of neighbors of some nodes in that subhypergraph may drop below k, violating the coreness condition. Thus a local approach used for computing the k-core [35, 42, 45] or (k, h)-core [41] in graphs results in *incorrect* neighborhood-based hypergraph cores (§6.1).

Our contributions are summarized below. **Novel problem and characterization (§2).** We define and investigate the novel problem of neighborhood-based core decomposition in hypergraphs. We prove that neighborhood-based k-cores are unique and the k-core contains the $(k + 1)$-core.

Exact algorithms (§3). We propose three exact algorithms with their correctness and time complexity analyses. Two of them, **Peel** and its enhancement **E-Peel**, adopt classic peeling [9], incurring global changes to the hypergraph. For **E-Peel**, we derive *novel lower-bound* on core-number that eliminates many redundant neighborhood recomputations. Our third algorithm, **Local-core** only makes node-level local computations. Even though the existing local method [42, 45] fails to correctly find neighborhood-based core-numbers in a hypergraph, our algorithm **Local-core** applies a *novel* **Core-correction** procedure, ensuring correct core-numbers.

Optimization and parallelization strategies (§4). We propose four optimization strategies to improve the efficiency of **Local-core**. Compressed representations for hypergraph (optimization-I) and the family of optimizations for efficient **Core-correction** (optimization-II) are novel to core-decomposition literature. The other optimizations, though inspired from graph literature, have not been adopted in earlier hypergraph-related works. We also propose parallelization of **Local-core** for the shared-memory programming paradigm.

(Neighborhood, degree)-core (§5). We define a more general hypergraph-core model, *(neighborhood, degree)-core* by considering both neighborhood and degree constraints, propose its decomposition algorithm **Local-core+Peel**, and demonstrate its superiority in diffusion spread.

Empirical evaluation (§6) on real and synthetic hypergraphs shows that the proposed algorithms are effective, efficient, and practical. Our OpenMP parallel implementation **Local-core(P)** decomposes hypergraph with 27M nodes, 17M hyperedges in 91 seconds.

Applications (§7). We show our decomposition to be more effective in disrupting diffusion than other decompositions. Our greedy algorithm proposed for volume-densest subhypergraph recovery achieves $(d_{pair}(d_{card} - 2) + 2)$-approximation, where hyperedge-cardinality (# nodes in that hyperedge) and node-pair co-occurrence (# hyperedges containing that pair) are at most d_{card} and d_{pair}, respectively. If the hypergraph is a graph ($d_{card} = 2$), our result generalizes Charikar's 2-approximation guarantee for densest subgraph discovery [14]. Our volume-densest subhypergraphs capture differently important meetup events compared to degree and clique graph decomposition-based densest subhypergraphs.

2 OUR PROBLEM AND CHARACTERIZATION

Hypergraph. A hypergraph $H = (V, E)$ consists of a set of nodes V and a set of hyperedges $E \subseteq P(V) \setminus \phi$, where $P(V)$ is the power set of V. A hyperedge is modeled as an unordered set of nodes.

Neighbors. Neighbors $N(v)$ of a node v in a hypergraph $H = (V, E)$ is the set of nodes $u \in V$ that co-occur with v in some hyperedge $e \in E$. That is, $N(v) = \{u \in V \mid u \neq v \wedge \exists\, e \in E \text{ s.t } u, v \in e\}$.

Strongly induced subhypergraph [6, 16, 28]. A strongly induced subhypergraph $H[S]$ of a hypergraph $H = (V, E)$, induced by a node set $S \subseteq V$, is a hypergraph with the node set S and the hyperedge set $E[S] \subseteq E$, consisting of all the hyperedges that are subsets of S.

$$H[S] = (S, E[S]), \text{ where } E[S] = \{e \mid e \in E \wedge e \subseteq S\} \quad (1)$$

In other words, every hyperedge in a strongly induced subhypergraph must exist in its parent hypergraph.

2.1 Problem Formulation

The *nbr-k-core* $H[V_k] = (V_k, E[V_k])$ of a hypergraph $H = (V, E)$ is the maximal (strongly) induced subhypergraph such that every node $u \in V_k$ has at least k neighbours in $H[V_k]$. For simplicity, we denote nbr-k-core, $H[V_k]$ as H_k. The *maximum core* of H is the largest k for which H_k is non-empty. The *core-number* $c(v)$ of a node $v \in V$ is the largest k such that $v \in V_k$ and $v \notin V_{k+1}$. The *core decomposition* of a hypergraph assigns to each node its core-number. Given a hypergraph, the problem studied in this paper is to correctly and efficiently compute its neighborhood-based core decomposition.

2.2 Differences with Other Core Decompositions

One can adapt broadly two kinds of approaches from the literature.

Approach-1. Transform the hypergraph into other objects (e.g., a graph), apply existing decomposition approaches [9, 13] on that object, and then project the decomposition back to the hypergraph. For instance, a hypergraph is transformed to a clique graph by replacing the hyperedges with cliques, and classical graph core decomposition [9] is applied. A hypergraph can also be transformed into a bipartite graph by representing the hyperedges as nodes in the second partition and creating an edge between two cross-partition nodes if the hyperedge in the second partition contains a node in the first partition. Finally, distance-2 core decomposition [13, 41] is applied. In distance-2 core-decomposition, nodes in k-core have at least k 2-hop neighbors in the subgraph. **First**, the decomposition they yield may be different from that of ours: $c(Popiden) = 5$ in both clique graph and dist-2 bipartite graph decompositions, whereas *Popiden* has core-number 4 in our decomposition. Clique graph and bipartite graph decompositions fail to identify that the relation $\langle Popiden\text{-}Hagler \rangle$ should not exist without the existence of event P_2, since P_2 is the only event where they co-participate. **Second**, the resulting decomposition may be unreasonable: low-importance events P_1 and P_2 by the same interest group are placed in different cores causing difficulty in separating less-important events from more-important ones. **Third**, such transformations inflate the problem size (§1). Bipartite graph representation results in inefficient decomposition (§6.2).

Approach-2. The *degree* $d(v)$ of a node v in hypergraph H is the number of hyperedges incident on v [12], i.e., $d(v) = |\{e \in E \mid v \in e\}|$. Sun et al. [50] define the *deg-k-core* H_k^{deg} of a hypergraph H as the maximal (strongly) induced subhypergraph of H such that every node u in H_k^{deg} has degree at least k in H_k^{deg}. This approach does not consider hyperedge sizes (§1). Therefore, it does not necessarily yield the same decomposition as our approach (Figure 1).

2.3 Nbr-k-Core: Properties

THEOREM 1. *The nbr-k-core H_k is unique for any $k > 0$.*

PROOF. Let, if possible, there be two distinct nbr-k-cores: $H_{k_1} = (V_{k_1}, E[k_1])$ and $H_{k_2} = (V_{k_2}, E[k_2])$ of a hypergraph $H = (V, E)$. By definition, both H_{k_1} and H_{k_2} are maximal strongly induced subhypergraphs of H. Construct the union hypergraph $H_k = (V_{k_1} \cup V_{k_2}, E[V_{k_1} \cup V_{k_2}])$. For any $u \in V_{k_1} \cup V_{k_2}$, u must be in either V_{k_1} or V_{k_2}. Thus, u must have at least k neighbours in H_{k_1} or H_{k_2}. Since $E[V_{k_1}] \cup E[V_{k_2}] \subseteq E[V_{k_1} \cup V_{k_2}]$, u must also have at least k neighbours in H_k. Since H_k is a supergraph of both H_{k_1} and H_{k_2}, H_{k_1} and H_{k_2} are not maximal, leading to a contradiction. □

THEOREM 2. *The $(k+1)$-core is contained in the k-core, $\forall\, k > 0$.*

PROOF. Let, if possible, for some node $u \in V_{k+1}$, $u \notin V_k$. Construct $S = V_k \cup V_{k+1}$. Since $u \notin V_k$, but $u \in V_{k+1} \subset S$, we get $|S| \geq |V_k| + 1$. It is easy to verify that every node $v \in S$ has at least k neighbours in $H[S]$ and $|S| > |V_k|$. Then, V_k is not maximal and thus not nbr-k-core, which is a contradiction. The theorem follows. □

3 ALGORITHMS

We propose three algorithms: Our algorithms **Peel** and its efficient variant **E-Peel** are inspired by peeling methods similar to graph core computation [9, 13]. The algorithm **Local-core** is inspired by local approaches to graph core computation [42, 45]. In the classic peeling algorithm for graph core decomposition, when a node is removed, the degree of its neighbors reduces by 1: this permits efficient optimizations and *linear-time* peeling algorithm [9, 15, 47]. However, in a neighborhood-based hypergraph core decomposition, deleting a node may reduce the neighborhood size of its neighboring node by more than 1. Hence, to recompute the number of neighbors of a deleted node's neighbor, one must construct the residual hypergraph, which makes the decomposition *polynomial-time* and costly. The algorithms **E-Peel** and **Local-core**, despite being inspired by the existing family of graph algorithms, are by no means trivial adaptations. For **E-Peel**, we devise a new local lower-bound for core-numbers as the lower-bound for graph core [13] is insufficient for our purpose. For **Local-core**, we show how a direct adaptation of local algorithm [35, 42, 45] leads to incorrect core computations (§3.3 and §6.1).

Algorithm 1 Peeling algorithm: **Peel**

```
Input: Hypergraph $H = (V, E)$
Output: Core-number $c(u)$ for each node $u \in V$
 1: for all $u \in V$ do
 2:     Compute $N_V(u)$                          ▹ set of neighbors of $u$ in $H = (V, E)$
 3:     $B[|N_V(u)|] \leftarrow B[|N_V(u)|] \cup \{u\}$
 4: for all $k = 1, 2, \ldots, |V|$ do
 5:     while $B[k] \neq \phi$ do
 6:         Remove a node $v$ from $B[k]$
 7:         $c(v) \leftarrow k$
 8:         for all $u \in N_V(v)$ do
 9:             Move $u$ to $B[\max\left(|N_{V \setminus \{v\}}(u)|, k\right)]$
10:         $V \leftarrow V \setminus \{v\}$
11: Return $c$
```

Hence, we devise *hypergraph h-index* and *local coreness constraint* and employ them to compute hypergraph cores correctly.

3.1 Peeling Algorithm

Following Theorem 2, the $(k+1)$-core can be computed from the k-core by "peeling" all nodes whose neighborhood sizes are less than $k+1$. Algorithm 1 describes our peeling algorithm: **Peel**, which processes the nodes in increasing order of their neighborhood sizes (Lines 4-10). B is a vector of lists: Each cell $B[i]$ is a list storing all nodes whose neighborhood sizes are i (Line 3). When a node v is processed at iteration k, its core-number is assigned to $c(v) = k$ (Line 7), it is deleted from the set of "remaining" nodes V (Line 10). The neighborhood sizes of the nodes in v's neighborhood are recomputed (each neighborhood size can decrease by more than 1, since when v is deleted, all hyperedges involving v are also deleted), and these nodes are moved to the appropriate cells in B (Lines 8-9). The algorithm completes when all nodes in the hypergraph are processed and have their respective core-numbers computed.

Proof of correctness. Initially $B[i]$ contains all nodes whose neighborhood sizes are i. When we delete some neighbor of a node u, the neighborhood size of u is recomputed, and u is reassigned to a new cell corresponding to its reduced neighborhood size until we find that the removal of a neighbor v of u reduces u's neighborhood size even below the current iteration number k (Line 9). When this happens, we correctly assign u's core-number $c(u) = k$. **(1)** Consider the remaining subhypergraph formed by the remaining nodes and hyperedges at the end of the $(k-1)^{\text{th}}$ iteration. Clearly, u is in the k-core since u has at least k neighbors in this remaining subhypergraph, where all nodes in the remaining subhypergraph also have neighborhood sizes $\geq k$. **(2)** The removal of v decreases u's neighborhood size smaller than the current iteration number k, thus when the current iteration number increases to $k+1$, u will not have enough neighbors to remain in the $(k+1)$-core.

Time complexity. Each node v is processed exactly once from B in Algorithm 1; when it is processed and thereby deleted from V, neighborhood sizes of the nodes in v's neighborhood are recomputed. Assume that the maximum number of neighbors and hyperedges of a node be d_{nbr} and d_{hpe}, respectively. Thus, Algorithm 1 has time complexity $O\big(|V| \cdot d_{nbr} \cdot (d_{nbr} + d_{hpe})\big)$.

3.2 Efficient Peeling with Bounding

An inefficiency in Algorithm 1 is that it updates the cell index of every node u that is a neighbor of a deleted node v. To do so, it has to compute the number of neighbors of u in the newly constructed subhypergraph. To delay this recomputation, we derive a local lower-bound for $c(u)$ via Lemma 1 and use it to eliminate many redundant neighborhood recomputations and cell updates (Algorithm 2). The intuition is that a node u will not be deleted at some iteration $k <$ the lower-bound on $c(u)$, thus we do not require computing u's neighborhood size until the value of k reaches the lower-bound on $c(u)$. Our lower-bound is local since it is specific to each node.

Algorithm 2 Efficient peeling algorithm with bounding: **E-Peel**

```
Input: Hypergraph $H = (V, E)$
Output: Core-number $c(u)$ for each node $u \in V$
 1: for all $u \in V$ do
 2:     Compute $\mathbf{LB}(u)$
 3:     $B[\mathbf{LB}(u)] \leftarrow B[\mathbf{LB}(u)] \cup \{u\}$
 4:     $setLB(u) \leftarrow True$
 5: for all $k = 1, 2, \ldots, |V|$ do
 6:     while $B[k] \neq \phi$ do
 7:         Remove a node $v$ from $B[k]$
 8:         if $setLB(v)$ then
 9:             $B[|N_V(v)|] \leftarrow B[\max(|N_V(v)|, k)] \cup \{v\}$
10:             $setLB(v) \leftarrow False$
11:         else
12:             $c(v) \leftarrow k$
13:             for all $u \in N_V(v)$ do
14:                 if $\neg setLB(u)$ then
15:                     Move $u$ to $B\left[\max\left(|N_{V \setminus \{v\}}(u)|, k\right)\right]$
16:             $V \leftarrow V \setminus \{v\}$
17: Return $c$
```

LEMMA 1 (LOCAL LOWER-BOUND). *Let $e_m(v) = \arg\max\{|e| : e \in E \wedge v \in e\}$ be the highest-cardinality hyperedge incident on $v \in V$. For all $v \in V$,*

$$c(v) \geq \max\left(|e_m(v)| - 1, \min_{u \in V} |N(u)|\right) = \mathbf{LB}(v) \tag{2}$$

PROOF. Notice that $c(v) \geq \min_{u \in V} |N(u)|$, since all nodes in the input hypergraph must be in the $(\min_{u \in V} |N(u)|)$-core. Next, we show that $c(v) \geq |e_m(v)| - 1$, by contradiction. Let, if possible, $|e_m| - 1 > c(v)$. This implies that v is not in the $(|e_m| - 1)$-core, denoted by $H[V_{|e_m|-1}]$. Consider $V' = V_{|e_m|-1} \cup \{u : u \in e_m\}$. Clearly, $|V'| \geq |V_{|e_m|-1}| + 1$, since $v \notin V_{|e_m|-1}$, but $v \in e_m$, so $v \in V'$. We next show that $H[V_{|e_m|-1}]$ is not the maximal subhypergraph where every node has at least $|e_m| - 1$ neighbors, which is a contradiction.

To prove non-maximality of $H[V_{|e_m|-1}]$, it suffices to show that for any $u \in V'$, $N_{V'}(u) \geq |e_m| - 1$. If $u \in V_{|e_m|-1} \subset V'$, $|N_{V'}(u)| \geq |N_{V_{|e_m|-1}}(u)| \geq |e_m| - 1$. If $u \in e_m$, $N_{V'}(u) \geq N_{e_m}(u) = |e_m| - 1$.

Since our premise $|e_m| - 1 > c(v)$ contradicts the fact that $H[V_{|e_m|-1}]$ is the $(|e_m| - 1)$-core, $|e_m| - 1 \leq c(v)$. □

Algorithm. Our efficient peeling approach is given in Algorithm 2: **E-Peel**. In Line 14, we do not recompute neighborhoods and update cells for those neighboring nodes u for which $setLB$ is True, thereby improving the efficiency. *$setLB$ is True for nodes for which* $\mathbf{LB}()$ *is known, but $N_V()$ at the current iteration is unknown.*

EXAMPLE 2. *Figure 3(a) illustrates the improvements made by Algorithm 2 in terms of neighborhood recomputations and cell updates. Since* $\mathbf{LB}(x) = 1$ *and every neighbor $u \in \{a, b, c, d\}$ has* $\mathbf{LB}(u) = 5$, *Algorithm 2 computes $c(x)$ before $c(u)$. Due to the local lower-bound-based initialization in Lines 1-4 and ascending iteration order of k, x is popped before u. The first time x is popped from B, x goes to $B[4]$ due to Line 9 and $setLB(x)$ is set to False*

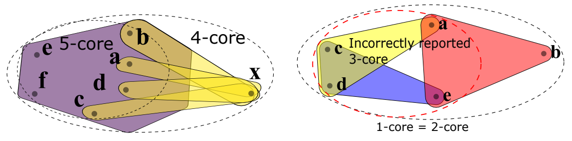


Figure 3: (a) During x's core-number computation, Algorithm 2 does not perform neighborhood recomputations and cell updates for x's neighbors $\{a, b, c, d\}$; thus saving four redundant neighborhood recomputations and cell updates. (b) For any $n > 1$, the h-index (Definition 2) of node a never reduces from $h_a^{(1)} = \mathcal{H}(2, 3, 3, 4) = 3$ to its core-number 2: $\lim_{n\to\infty} h_a^{(n)} = 3 \neq$ core-number of a. Because a will always have at least 3 neighbors (c, d, and e) whose h-indices are at least 3. As a result, the naïve approach reports an incorrect 3-core.

in Line 10. The next time x is popped (also at iteration $k = 4$), the algorithm computes $c(x)$ in Line 12. $setLB(u)$ is still True for u (Lines 13-15), as the default initialization of $setLB(u)$ has been True (Line 4). Hence, none of the computations in Line 15 is executed for u. Intuitively, since the 5-core does not contain x, deletion of x and the yellow hyperedges should be inconsequential to computing $c(u)$ correctly. $c(u)$ is computed in the next iteration ($k = 5$) after it is popped and is reassigned to $B[5]$, and $setLB(u)$ becomes False.

Proof of correctness. The proof of correctness follows that of Algorithm 1. When a node v is extracted from $B[k]$ at iteration k, we check $setLB(v)$. **(1)** Lemma 1 ensures that, if we extract a node v from $B[k]$ and $setLB(v)$ is True, then $c(v) \geq k$. In that case, we compute the current value of $N_V()$, where V denotes the set of remaining nodes, and insert v into the cell: $B[\max(|N_V(v)|, k)]$. We also set $setLB(v) =$ False, implying that $N_V(v)$ at the current iteration is known. **(2)** In contrast, if we extract a node v from $B[k]$ and $setLB(v)$ is False, this indicates that $c(v) = k$, following the same arguments as in Algorithm 1. In this case, we correctly assign v's core-number to k, and v is removed from V. Moreover, for those neighbors u of v for which $setLB(u)$ is True, implying that $c(u) \geq k$, we appropriately delay recomputing their neighborhood sizes.

Time complexity. Following similar analysis as in Algorithm 1, **E-Peel** has time complexity $O(\alpha \cdot |V| \cdot d_{nbr} \cdot (d_{nbr} + d_{hpe}))$, where $\alpha \leq 1$ is the ratio of the number of neighborhood recomputations in Algorithm 2 over that in Algorithm 1. Based on our experimental results in §6, **E-Peel** can be up to 17x faster than **Peel**.

3.3 Local Algorithm

Although **Peel** and its efficient variant **E-Peel** correctly computes core-numbers, they must modify the remaining hypergraph at every iteration by peeling nodes and hyperedges. Peeling impacts the hypergraph data structure globally and must be performed in sequence. Thus, there is little scope for making **Peel** and **E-Peel** more efficient via parallelization. Furthermore, they are not suitable in a time-constrained setting where a high-quality partial solution is sufficient. We propose a novel local algorithm that is able to provide partial solutions, amenable to optimizations, and parallelizable.

Naïve adoption of local algorithm in hypergraphs: a negative result. Eugene et al. [42] adopt Hirsch's index [30], popularly known as the *h-index* (Definition 1), to propose a local algorithm for core computation in graphs. This algorithm relies on a recurrence relation that defines higher-order h-index (Definition 2). The local algorithm for graph core computation starts by computing h-indices of order 0 for every node in the graph. At each iteration $n > 0$, it computes order-n h-indices using order-$(n-1)$ h-indices computed in the previous iteration, until there is no node whose h-index changes.

Definition 1 ($\mathcal{H}$-operator [30, 42]). *Given a finite set of positive integers $\{x_1, x_2, \ldots, x_t\}$, $\mathcal{H}(x_1, x_2, \ldots, x_t) = y > 0$, where y is the maximum integer such that there exist at least y elements in $\{x_1, x_2, \ldots, x_t\}$, each of which is at least y.*

Example 3 (H-operator). $\mathcal{H}(1, 1, 1, 1) = 1$, $\mathcal{H}(1, 1, 1, 2) = 1$, $\mathcal{H}(1, 1, 2, 2) = 2$, $\mathcal{H}(1, 2, 2, 2) = 2$, $\mathcal{H}(1, 2, 3, 3) = 2$, $\mathcal{H}(1, 3, 3, 3) = 3$

Definition 2 (h-index of order n [42]). *Let $\{u_1, u_2, \ldots, u_t\}$ be the set of neighbors of node $v \in V$ in graph $G = (V, E)$. The n-order h-index of node $v \in V$, denoted as $h_v^{(n)}$, is defined for any $n \in \mathbb{N}$ by the recurrence relation*

$$h_v^{(n)} = \begin{cases} |N(v)| & n = 0 \\ \mathcal{H}\left(h_{u_1}^{(n-1)}, h_{u_2}^{(n-1)}, \ldots, h_{u_t}^{(n-1)}\right) & n \in \mathbb{N} \setminus \{0\} \end{cases} \quad (3)$$

For neighborhood-based hypergraph core decomposition via local algorithm, we define $h_v^{(0)}$ as the number of neighbors of node v in hypergraph $H = (V, E)$ (instead of graph G). The definition of $h_v^{(n)}$ for $n > 0$ remains the same. However, this *direct adoption of local algorithm to compute hypergraph cores does not work.* Although one can prove that the sequence $(h_v^{(n)})$ adopted for hypergraph has a limit, that value in-the-limit is not necessarily the core-number $c(v)$ for every $v \in V$. For some node, the value in-the-limit of its h-indices is strictly greater than the core-number of that node. The reason is as follows. $\mathcal{H}$-operator acts as both necessary and sufficient condition for computing graph cores. It has been shown that the subgraph induced by $G[S]$, where S contains all neighbors u of a node v such that $h_u^{(\infty)} \geq c(v)$, satisfies $h_v^{(\infty)} = c(v)$ [42, p.5 Theorem 1]. However, Definition 2 is not sufficient to show $h_v^{(\infty)} = c(v)$ for a hypergraph. Because it is not guaranteed that v will have at least h_v^∞ neighbors in the subhypergraph $H[\{u : h_u^\infty \geq h_v^\infty\}]$ that is reported as the $c(v)$-core. So, the reported $c(v)$-core can be incorrect.

Example 4. *For the hypergraph in Figure 3(b), the values in-the-limit of h-indices are $h_a^{(\infty)} = h_c^{(\infty)} = h_d^{(\infty)} = h_e^{(\infty)} = 3$ and $h_b^{(\infty)} = 2$. No matter how large n is chosen, Equation (3) does not help $h_a^{(n)}$ to reach the correct core-number (=2) for a. Three neighbors of node a, namely c, d, and e have their h^∞-values at least $3 = h_v^\infty$ in Figure 3(b). But, a does not have at least 3 neighbors in the subhypergraph $H[\{a, c, d, e\}] = H[\{u : h_u^\infty \geq h_v^\infty\}]$. Thus, the 3-core $H[\{a, c, d, e\}]$ reported by the naïve h-index based local approach is incorrect. The reason is that a and e are no longer neighbors to each other in $H[\{a, c, d, e\}]$ due to the absence of b.*

Local algorithm with local coreness constraint. Motivated by the observation mentioned above, we define a constraint as a sufficient condition, upon satisfying which we can guarantee that for every node v, 1) the sequence of its h-indices converges and 2) the value in-the-limit $h_v^{(\infty)}$ is such that v has at least $h_v^{(\infty)}$ neighbors in the subhypergraph induced by $H[u : h_u^{(\infty)} \geq h_v^{(\infty)}]$. The first condition

Algorithm 3 Local algorithm with local coreness constraint: **Local-core**

```
Input: Hypergraph $H = (V, E)$
Output: Core-number $c(v)$ for each node $v \in V$
 1: for all $v \in V$ do
 2:     $\hat{h}_v^{(0)} = h_v^{(0)} \leftarrow |N(v)|$.
 3: for all $n = 1, 2, \ldots, \infty$ do
 4:     for all $v \in V$ do
 5:         $h_v^{(n)} \leftarrow \min\left(\mathcal{H}(\{\hat{h}_u^{(n-1)} : u \in N(v)\}), \hat{h}_v^{(n-1)}\right)$
 6:     for all $v \in V$ do
 7:         $c(v) \leftarrow \hat{h}_v^{(n)} \leftarrow$ Core-correction $(v, h_v^{(n)}, H)$
 8:     if $\forall v, \hat{h}_v^{(n)} == h_v^{(n)}$ then
 9:         Terminate Loop
10: Return $c$
```

is critical for algorithm termination. The second condition is critical for correct computation of core-numbers as discussed in Example 4.

DEFINITION 3 (LOCAL CORENESS CONSTRAINT (LCC)). *Given a positive integer k, for any node $v \in V$, let $H^+(v) = (N^+(v), E^+(v))$ be the subhypergraph of H such that for any $n > 0$*

$$E^+(v) = \{e \in Incident(v) : h_u^{(n)} \geq k, \forall u \in e\}$$
$$N^+(v) = \{u : u \in e, \forall e \in E^+(v)\} \setminus \{v\} \tag{4}$$

Local coreness constraint (for node v) is satisfied at k, denoted as $LCCSAT(k)$, iff $\exists H^+(v)$ with at least k nodes, i.e., $|N^+(v)| \geq k$. Here, $Incident(v)$ is the set of hyperedges incident on v.

We define *Hypergraph h-index* based on the notion of *LCCSAT* and a re-defined recurrence relation for $h_v^{(n)}$.

DEFINITION 4 (HYPERGRAPH h-INDEX OF ORDER n). *The Hypergraph h-index of order n for node v, denoted as $\hat{h}_v^{(n)}$, is defined for any natural number $n \in \mathbb{N}$ by the following recurrence relation:*

$$\hat{h}_v^{(n)} = \begin{cases} |N(v)| & n = 0 \\ h_v^{(n)} & n > 0 \wedge LCCSAT(h_v^{(n)}) \\ \max\{k \mid k < h_v^{(n)} \wedge LCCSAT(k)\} & n > 0 \wedge \neg LCCSAT(h_v^{(n)}) \end{cases} \tag{5}$$

$h_v^{(n)}$ is a newly defined recurrence relation on hypergraphs:

$$h_v^{(n)} = \begin{cases} |N(v)| & n = 0 \\ \min\left(\mathcal{H}\left(\hat{h}_{u_1}^{(n-1)}, \hat{h}_{u_2}^{(n-1)}, \ldots, \hat{h}_{u_t}^{(n-1)}\right), \hat{h}_v^{(n-1)}\right) & n \in \mathbb{N} \setminus \{0\} \end{cases} \tag{6}$$

The recurrence relations in Equations (5) and (6) are coupled: $\hat{h}_v^{(n)}$ depends on the evaluation of $h_v^{(n)}$, which in turn depends on the evaluation of $\hat{h}_v^{(n-1)}$. Such inter-dependency causes both sequences to converge, as proven in our correctness analysis.

Local-core (Algorithm 3) initializes $h_v^{(0)}$ and $\hat{h}_v^{(0)}$ to $|N(v)|$ for every node $v \in V$ (Lines 1-2) following Equation (6) and Equation (5), respectively. At every iteration $n > 0$, Algorithm 3 first computes $h_v^{(n)}$ for every node $v \in V$ (Lines 4-5) following Equation (6). In order to decide whether the algorithm should terminate at iteration n (Lines 8-9), the algorithm computes $\hat{h}_v^{(n)}$ using Algorithm 4. Algorithm 4 checks for every node $v \in V$, whether $LCCSAT(h_v^{(n)})$ is True or False. Following Equation (5), if $LCCSAT(h_v^{(n)})$ is True it returns $h_v^{(n)}$; if $LCCSAT(h_v^{(n)})$ is False, a suitable value lower than $h_v^{(n)}$ is returned. The returned value $\hat{h}_v^{(n)}$ is considered as the estimate of core-number $c(v)$ at that iteration (Line 7). To compute

Algorithm 4 **Core-correction** procedure

```
Input: Node $v$, $v$'s hypergraph $h$ index $h_v^{(n)}$, hypergraph $H$
1: while $h_v^{(n)} > 0$ do
2:     Compute $E^+(v) \leftarrow \{e \in Incident(v) : h_u^{(n)} \geq h_v^{(n)}, \forall u \in e\}$
3:     Compute $N^+(v) = \{u : u \in e, \forall e \in E^+(v)\} \setminus \{v\}$
4:     if $|N^+(v)| \geq h_v^{(n)}$ then
5:         return $h_v^{(n)}$
6:     else
7:         $h_v^{(n)} \leftarrow h_v^{(n)} - 1$
```

$LCCSAT(h_v^{(n)})$, Algorithm 4 checks in Line 4 if the subhypergraph $H^+(v) = (N^+(v), E^+(v))$ constructed in Lines 2-3 contains at least $h_v^{(n)}$ neighbors of v. If v has at least $h_v^{(n)}$ neighbors in the subhypergraph $H^+(v)$, due to Equation (5) no correction to $h_v^{(n)}$ is required. In this case, Algorithm 4 returns $h_v^{(n)}$ in Line 5. If v does not have at least $h_v^{(n)}$ neighbors in the subhypergraph $H^+(v)$ (Line 4), $LCCSAT(h_v^{(n)})$ is False by Definition 3. Following Equation (5), a correction to $h_v^{(n)}$ is required. In search for a suitable corrected value lower than $h_v^{(n)}$ and a suitable subhypergraph $H^+(v)$, Line 7 keeps reducing $h_v^{(n)}$ by 1. Reduction to $h_v^{(n)}$ causes $|N^+(v)|$ to increase, while $h_v^{(n)}$ decreases, until the condition in Line 4 is satisfied. At some point a suitable subhypergraph must be found.

Theorem 4 proves that the numbers returned by Algorithm 3 at that point indeed coincide with the true core-numbers. The termination condition $\hat{h}_v^{(n)} = h_v^{(n)}$ must be satisfied at some point because Theorem 3 proves that $\lim_{n\to\infty} \hat{h}_v^{(n)} = \lim_{n\to\infty} h_v^{(n)}$ $\forall v \in V$.

EXAMPLE 5. *Consider iteration $n = 1$ of Algorithm 3, when the input to the algorithm is the hypergraph in Figure 3(b). The algorithm corrects the core-estimate $h_a^{(1)} = 3$ to $\hat{h}_a^{(1)} = 2$ in Line 7. Because in Line 4 of **Core-correction**, the algorithm finds that for $h_a^{(1)} = 3$, a only has $|N^+(v)| = 2$ neighbors in $H^+(v) = H[\{a, c, d, e\}]$ thus violating the condition that $|N^+(v)| > h_a^{(1)}$. Hence $h_a^{(1)}$ needs to be corrected to satisfy $LCCSAT(h_a^{(1)})$. In Line 7 of **Core-correction**, it reduces $h_a^{(1)}$ by one. Subsequently for $h_a^{(1)} = 2$ the subhypergraph $H^+(a) = H$ indeed satisfies $LCCSAT(h_a^{(1)})$. This is how Algorithm 3 corrects the case of incorrect core-numbers discussed in Example 4.*

3.4 Theoretical Analysis of Local-core

Proof of Correctness. Algorithm 3 terminates after a finite number of iterations because by Theorem 3, for any $v \in V$, sequences $(h_v^{(n)})$ and $(\hat{h}_v^{(n)})$ are finite and have the same limit. At the limit, $\forall v \in V$, $\lim_{n\to\infty} h_v^{(n)} = \lim_{n\to\infty} \hat{h}_v^{(n)}$ holds and it follows from Theorem 4 that $\forall v \in V$, $\lim_{n\to\infty} h_v^{(n)} = \lim_{n\to\infty} \hat{h}_v^{(n)} = c(v)$. Due to limitation of space, proofs are given in our extended version [4].

THEOREM 3. *For any node $v \in V$ of a hypergraph $H = (V, E)$, the two sequences $(h_v^{(n)})$ defined by Equation (6) and $(\hat{h}_v^{(n)})$ defined by Equation (5) have the same limit: $\lim_{n\to\infty} h_v^{(n)} = \lim_{n\to\infty} \hat{h}_v^{(n)}$.*

THEOREM 4. *If the local coreness-constraint is satisfied for all nodes $v \in V$ at the terminal iteration, the corrected h-index at the terminal iteration $\hat{h}_v^{(\infty)}$ satisfies: $\hat{h}_v^{(\infty)} = c(v)$.*

Algorithm 5 Optimized local algorithm: **Local-core(OPT)**

```
Input: Hypergraph $H = (V, E)$
Output: Core-number $c(v)$ for each node $v \in V$
 1: Construct CSR representations /* Opt-I */
 2: for all $v \in V$ do
 3:     Compute $\mathbf{LB}(v)$ /* local lower-bounds for Opt-IV */
 4:     $c(v) \leftarrow \hat{h}_v^{(0)} \leftarrow h_v^{(0)} \leftarrow |N(v)|$ /* core-estimate c(v) initialized for Opt-III*/
 5: for all $n = 1, 2, \ldots, \infty$ do
 6:     for all $v \in V$ do
 7:         if $h_v^{(n)} > LB(v)$ then /* Opt-IV*/
 8:             $c(v) \leftarrow h_v^{(n)} \leftarrow \min(\mathcal{H}(\{c(u) : u \in N(v)\}), c(v))$ /* Opt-III*/
 9:     for all $v \in V$ do
10:         if $h_v^{(n)} > LB(v)$ then /* Opt-IV */
11:             $c(v) \leftarrow \hat{h}_v^{(n)} \leftarrow$ Core-correction $(v, h_v^{(n)}, \mathcal{H})$ /* Opt-II & Opt-III*/
12:     if $\forall v, \hat{h}_v^{(n)} == h_v^{(n)}$ then
13:         Terminate Loop
14: Return $c$
```

Time complexity. Assume that Algorithm 3 terminates at iteration τ of the for-loop at Line 3. Each iteration has time-complexity $O(\sum_v |N(v)|(d(v) + |N(v)|) + \sum_v |N(v)|)$, the first term is due to Lines 6-7 and the second term is due to Lines 4-5. Computing $\mathcal{H}$-operator requires hypergraph h-indices of v's neighbors and costs linear-time: $O(|N(v)|)$. Core-correcting v requires at most $|N(v)|$ iterations of while-loop (Line 1, Algorithm 4), at each iteration constructing each of $E^+(v)$ and $N^+(v)$ costs $O(d(v) + |N(v)|)$. Thus, **Local-core** has time complexity $O\left(\tau * \sum_v \left(d(v)|N(v)| + |N(v)|^2\right)\right)$.

Upper-bound on the number of iterations. To derive an upper-bound on the number of iterations τ required by **Local-core**, we define the notion of *neighborhood hierarchy* for nodes in a hypergraph and prove that every node converges by the time the for-loop iterator in Line 3 (Algorithm 3) reaches its neighborhood hierarchy.

Definition 5 (Neighborhood hierarchy). *Given a hypergraph H, the i-th neighborhood hierarchy, denoted by N_i, is the set of nodes that have the minimum number of neighbors in $H[V']$, where $V' = V \setminus \cup_{0 \le j < i} N_j$. Formally,*

$$N_i = \{v : \underset{v \in V'}{\arg\min} |N_{H[V']}(v)|\} \quad (7)$$

Consider the hypergraph H in Figure 1(a). Since *Hall* has 4 neighbors, $N_0 = \{Hall\}$. After deleting *Hall*, nodes *Wallace* and *Popiden* have the lowest number of neighbors (4) in the remaining sub-hypergraph. Hence, $N_1 = \{Wallace, Popiden\}$. After deleting them, nodes in R has the lowest number of neighbors (6) in the remaining sub-hypergraph. Hence, $N_2 = R$. Finally, $N_3 = T$.

Theorem 5 (individual node convergence). *Given a node $v \in N_i$ in a hypergraph H, it holds that $\forall n \ge i$, $\hat{h}_v^{(n)} = c(v)$.*

The proof by induction on i is given in our extended version [4]. Clearly, τ is at most the largest neighborhood hierarchy of the nodes.

4 OPTIMIZATION AND PARALLELIZATION OF THE LOCAL-CORE ALGORITHM

We propose four optimizations to improve the efficiency of **Local-core** (§3.3). Algorithm 5 presents the pseudocode for the optimized algorithm, **Local-core(OPT)**, where all four optimizations are indicated. Optimization-I adopts sparse representations to efficiently evaluate neighborhood queries in hypergraphs, while Optimization-II consists of three implementation-specific improvements to efficiently perform **Core-correction**. *Optimizations-I and II have not been used in earlier core-decomposition works for both graphs and hypergraphs.* Our Optimizations-III and IV are motivated by [41], where similar optimizations are proposed for graph (k, h)-core decomposition to improve convergence and eliminate redundant computations, respectively. *However such optimizations have not been adopted in earlier hypergraph-related works.*

Optimization-I (Compressed hypergraph representation). Local-core makes two primitive neighborhood queries on hypergraph structures: neighbors enumeration (for h-index computation, Lines 2 and 5, Algorithm 3) and incident-hyperedges enumeration (during **Core-correction**, Line 2, Algorithm 4). A naïve implementation keeps a $|V| \times |V|$ matrix for neighbors counting queries and a $|V| \times |E|$ matrix for incident-hyperedge queries. However, storing such matrices in memory is expensive and unnecessary for large hypergraphs since these matrices are sparse in practice. Hence, it is imperative to adopt a compressed sparse representation, e.g., *Compressed sparse row* (CSR) for these matrices. For example, in CSR representation we use two arrays $\mathcal{F}$ and $\mathcal{N}$ for storing neighbors. For all nodes $v \in V$, $\mathcal{N}[v]$ stores the starting index in $\mathcal{F}$ containing neighbors of node v. To facilitate neighborhood enumeration query, neighbors of node v are stored at contiguous indices $\mathcal{N}[v], \mathcal{N}[v]+1, \ldots, \mathcal{N}[v+1]-1$ in $\mathcal{F}$. Any neighbor u connected to v by a hyperedge has their own set of neighbors $N(u)$ stored at contiguous indices $\mathcal{N}[u]$, $\mathcal{N}[u]+1, \ldots$, $\mathcal{N}[u+1]-1$ in $\mathcal{F}$. Hence for any node $v \in V$, $\mathcal{N}[v+1] - \mathcal{N}[v]$ gives the number of neighbors $|N(v)|$ in $O(1)$ time and neighborhood enumeration takes $O(|N(v)|)$ time; in contrast to the matrix representations which take $O(|V|)$ time for both neighbors-enumeration and incident-hyperedges enumeration queries. CSR representations, due to their contiguous memory locations, can also exploit spatial memory access patterns in the **Local-core** algorithm.

Optimization-II (Efficient Core-correction and LCCSAT). We design three optimization methods for more efficient **Core-correction**.

Hyperedge-index for efficient E^+ computation: In Line 2 of **Core-correction** (Algorithm 4), we check if $h_u^{(n)} \ge h_v^{(n)}$ for every node $u \in e$ such that $e \in Incident(v)$. This computation incurs $O(d(v) + |N(v)|)$ at every while-loop (Line 1) of **Core-correction**. Since the number of while-loop iterations is $|N(v)|$ in the worst-case, an inefficient E^+ computation contributes $O(|N(v)|(d(v) + |N(v)|))$ to the total cost of **Core-correction**. We reduce this cost to $O(|N(v)|d(v))$ by maintaining an index E_e for hyperedges. $E_e^{(n)}$ records for every hyperedge $e \in E$ the minimum of h-indices of its constituent nodes ($\min_{u \in e} h_u^{(n)}$). We compute $E^+(v)$ by traversing only the incident hyperedges whose $E_e^{(n)} \ge h_v^{(n)}$. Storing E_e for all hyperedges costs $O(|E|)$ space and constructing the indices costs $O(\sum_{e \in E} |e|)$ time. However, hyperedge-indices are constructed only once before every iteration in **Local-core** (Line 3, Algorithm 3). Moreover, hyperedge-index helps efficiently compute N^+ as follows.

Dynamic programming for efficient N^+ computation: The while loop in the **Core-correction** procedure computes $k = h_v^{(n)}, h_v^{(n)} - 1, \ldots, \hat{h}_v^{(n)}$; and for every $\hat{h}_v^{(n)} \le k \le h_v^{(n)}$, it recomputes $E^+(v)$ and $N^+(v)$ until returning $\hat{h}_v^{(n)}$ as output. Without any optimization, the cost of computing $N^+(v)$ at every while-loop iteration k (Line 1) is $O(d(v) + |N(v)|)$. Since the number of while-loop iterations is $(h_v^{(n)} - \hat{h}_v^{(n)}) \le |N(v)|$, an inefficient N^+ computation contributes $O(|N(v)|(d(v) + |N(v)|))$ to the total cost of **Core-correction**.

We reduce this cost to $O(d(v) + |N(v)|)$ by constructing an index B such that $B[h_v^{(n)}]$ records the set of incident hyperedges whose hyperedge-index $E_e^{(n)} \geq h_v^{(n)}$, and for every $k < h_v^{(n)}$, $B[k]$ records the incident hyperedges whose hyperedge-index $E_e^{(n)} = k$.

Let us denote $E^+(v)$ and $N^+(v)$ at k as $E^+(v,k)$ and $N^+(v,k)$, respectively. Exploiting the index structure B, we have the following dynamic programming paradigm for efficiently computing $N^+(v,k)$.

$$N^+(v,k) = \begin{cases} \cup_e B[k] & k = h_v^{(n)} \\ N^+(v,k+1) \cup (\cup_e B[k]) & \hat{h}_v^{(n)} \leq k < h_v^{(n)} \end{cases} \tag{8}$$

Instead of traversing all incident hyperedges at every $\hat{h}_v^{(n)} \leq k \leq h_v^{(n)}$, we only traverse hyperedges at $B[k]$ to compute $N^+(v,k)$. Since the indices $B[k]$ are mutually exclusive, each incident hyperedge is traversed at most once during the entire **Core-correction** procedure. The storage cost of B is $O(h_v^{(n)} + d(v))$, as there are at most $h_v^{(n)}$ keys in B and exactly $d(v)$ hyperedges are stored in B. Due to efficient E^+ and N^+ computation, core-correction costs $O(|N(v)|d(v) + d(v) + |N(v)|)$ instead of $O(|N(v)|d(v) + |N(v)|^2)$.

Efficient LCCSAT computation: We return *True* upon finding the first hyperedge $e \in Incident(v)$ adding which to $E^+(v)$ causes $|N^+(v)| \geq h_v^{(n)}$. Adding subsequent incident hyperedges to $E^+(v)$ increases $|N^+(v)|$ more without affecting $LCCSAT(k)$=*True*.

Optimization III (Faster convergence). In Line 8 (Algorithm 5), instead of computing $h_v^{(n)}$ as per Equation (6), we compute a smaller value $g_v^{(n)}$ by replacing some operands $\hat{h}_{u_i}^{(n-1)}$ in the $\mathcal{H}$-operator with $h_{u_i}^{(n)} \leq \hat{h}_{u_i}^{(n-1)}$(due to Theorem 3). This leads to faster convergence since $g_v^{(n)} \leq h_v^{(n)}$, because lowering a few arguments may cause the output of $\mathcal{H}()$ to decrease further, e.g., $\mathcal{H}(1,1,2,2) = 2$, whereas $\mathcal{H}(1,1,1,2) = 1$. Computing g_v does not affect the correctness because the new interleave sequence $(h_v, g_v, \hat{g}_v)$ is also monotonically non-increasing and lower-bounded by 0. This is because $h_v^{(n)} \geq g_v^{(n)}$ and **Core-correction** never increases $g_v^{(n)}$, so $g_v^{(n)} \geq \hat{g}_v^{(n)}$. Thus, $(h_v^{(n)})$, $(g_v^{(n)})$, and $(\hat{g}_v^{(n)})$ have the same limit as that in Theorem 3: $\hat{h}^{(\infty)}$. Finally, from Theorem 4, it follows that $\hat{h}^{(\infty)} = c(v)$ holds despite this optimization.

Optimization-IV (Reducing redundant $\mathcal{H}$ computations and LCCSAT checks). We use local lower-bound $\mathbf{LB}(v)$ on core-numbers (Lemma 1) to reduce the number of h-index computations and core corrections. At some iteration n, if $\hat{h}_v^{(n)}$=$\mathbf{LB}(v)$, it ensures that $c(v)$=$\mathbf{LB}(v)$. Thus, h-index for v would no longer reduce in future iterations; otherwise, $c(v) = \hat{h}_v^{(\infty)} < \hat{h}_v^{(n)} = \mathbf{LB}(v)$, which is a contradiction. Hence, it must be that $c(v) = \hat{h}^{(\infty)} = \hat{h}^{(n)}$. We need not compute the h-index and core corrections for v at future iterations.

Note that Liu et al. [41] determine the redundancy of $\mathcal{H}$ computation for node v based on the convergence of both $h_v^{(n)}$ and $h_u^{(n)}$ of neighbors $u \in N(v)$. This does not work for our problem, because even if a node v and its neighbors' h-indices have converged, node v may still need core correction (e.g., Example 4).

Efficiency comparison of Local-core(Opt) and Peel. The terms d_{nbr} and d_{hpe} in the time complexity of **Peel** (§3.1) are upper bounds of $|N(v)|$ and $d(v)$, respectively, $\forall v \in V$. Hence, the complexity of **Peel** is comparable to $O(\sum_{v \in V} |N(v)|d(v) + N(v)^2)$, which is same as the complexity of one round of **Local-core** (§3.3). Thus, **Local-core** is slower than **Peel**. However, our optimizations reduce its complexity significantly. For instance, by efficiently computing E^+ and N^+, optimization-II reduces the complexity to $O(\tau \sum_v (d(v)|N(v)| + d(v) + |N(v)|)) = O(\tau \sum_v d(v)|N(v)|) = O(\tau |V| d_{hpe} d_{nbr})$. Hence, **Local-core(Opt)** is at least $\frac{d_{nbr}+d_{hpe}}{\tau d_{hpe}} = (\frac{d_{nbr}}{\tau d_{hpe}} + \frac{1}{\tau})$-times faster than **Peel** due to Optimization-II. A node usually has more neighbors than its degree in large-scale, real-world hypergraphs (e.g., *aminer*, *dblp* in Table 1), resulting in $d_{nbr} > d_{hpe}$. Optimizations III and IV further reduce τ, making the ratio > 1, and thus **Local-core(Opt)** is much faster than **Peel** in our experiments.

Parallelization of Local-core. We propose parallel **Local-core(P)** algorithm following the shared-memory, data parallel programming paradigm, which further improves efficiency. The algorithm partitions nodes into T partitions, where T is the number of threads. Each thread is responsible for computing core-numbers of nodes in its own partition. To improve load-balancing, we adopt the longest-processing-time-first scheduling approach [27] such that the aggregated number of neighbors of nodes in different threads are roughly the same. **Local-core(P)** has three primary differences compared to sequential **Local-core(OPT)** (Algorithm 5).

First, at iteration 0 (Line 3), every thread initializes hypergraph h-indices for its allocated nodes in parallel. Concurrent computation of $|N(v)|$ and $|N(u)|$ for nodes allocated to different threads requires concurrent reads to the CSR representation. Since the CSR representation does not change across queries, both queries will produce the same result as their sequential counterparts. **Second**, at subsequent iterations ($n > 0$), every thread computes $h_v^{(n)}$ and the corrected value $\hat{h}_v^{(n)}$ for its allocated nodes in parallel (Lines 6-11). Interestingly, our algorithm still terminates with the correct output because none of our previous theoretical results rely on any particular computation order of nodes in the same iteration. The computation order only affects the number of iterations required for convergence. **Finally**, all threads keep their respective core-correction counter, which counts the number of allocated nodes that have been core-corrected in a given round. Once all the counters report 0, **Local-core(P)** terminates.

5 EXTENSION TO (NEIGHBORHOOD, DEGREE)-CORE DECOMPOSITION

We propose a new hypergraph-core model, **(neighborhood, degree)-core**, or (k,d)-core in short, by considering both neighborhood and degree constraints. (k,d)-core eliminates problems such as the nbr-k-core of a node can be very large if it is involved in one large-size hyperedge. Notice that the d-value of the (k,d)-core for such a node would be small, differentiating it from other nodes having both higher degrees and a higher number of neighbors. We demonstrate the superiority of (k,d)-core in diffusion spread (§7.1).

(k,d)-core. Given integers $k,d > 0$, the (k,d)-core, denoted as $H[V_{(k,d)}] = (V_{(k,d)}, E[V_{(k,d)}])$ is the maximal subhypergraph s.t. every node $u \in V_{(k,d)}$ has at least k-neighbors and degree $\geq d$ in the strongly induced subhypergraph $H[V_{(k,d)}]$.

(k,d)-core decomposition is difficult: **(1)** Unlike neighborhood-based core decomposition that defines a total hierarchical order across different cores, (k,d)-core decomposition forms a core lattice defining only partial containments: $(k+i, d+j)$-core is contained

Table 1: Datasets: $|V|$ #nodes, $|E|$ #hyperedges, $d(v)$ avg. degree of a node, $|e|$ avg. cardinality of a hyperedge, $|N(v)|$ avg. #neighbors/node

	hypergraph	$\|V\|$	$\|E\|$	$d(v)$	$\|e\|$	$\|N(v)\|$
Syn.	*bin4U*	500	12424	99.4±8.5	4±0	225.3±15.5
	bin3U	500	16590	99.5±8	3±0	164.1±11.6
	pref3U	125329	250000	5.9±915.9	3±0	4.5±412.4
Real	*enron*	4423	5734	6.8±32	5.2±5	25.3±44
	contact	242	12704	127±55.2	2.4±0.5	68.7±26.6
	congress	1718	83105	426.2±475.8	8.8±6.8	494.7±248.6
	dblp	1836596	2170260	4±11.6	3.4±1.8	9±21.4
	aminer	27850748	17120546	2.3±5	3.7±2.6	8.4±24.1

in (k,d)-core $\forall k, d > 0; i, j \geq 0$. **(2)** Let k_{max} and d_{max} denote the maximum core-numbers via neighborhood- and degree-based core decompositions, respectively. We can have up to $O(k_{max} \cdot d_{max})$ different (k,d)-cores. Due to such challenges, we keep the problem of designing a holistic local algorithm for (k,d)-core decomposition open. Following a similar notion from multi-layer core decomposition [24], we propose a hybrid local and peeling approach.

Local-core+Peel algorithm for (k,d)-core decomposition. We initialize all (k,d)-cores as empty and compute neighborhood-based core decomposition of all nodes using **Local-core(OPT)**. Our algorithm (given in [4]) has two for-loops: At each outer for-loop iteration k, we construct the nbr-k-core ($H[V_k]$) and peel the nodes in V_k to find various (k,d)-cores with the same k but different d. To do so, we use a vector of lists B to define the initial peeling order of V_k based on their degrees in cell $H[V_k]$ similar to algorithm **Peel**. We traverse B in ascending order of values d and peel nodes from the current cell $B[d]$ until $B[d]$ is empty. Whenever a node is removed from B, it is assigned to proper (k,d)-core, and its neighbors in $H[V_k]$ are moved to appropriate cells in B. After we assign all the nodes in V_k to various (k,d)-cores for varying d, we proceed to do the same for V_{k+1} at the next iteration of the outer for-loop.

6 EMPIRICAL EVALUATION

We empirically evaluate the performance of our algorithms on four synthetic and five real-world datasets (Table 1). We implement our algorithms in GNU C++11 and OpenMP API version 3.1. All experiments are conducted on a server with 128 AMD EPYC 32-core processors and 256GB RAM. **Our code and datasets are at [3]**.

Datasets. Among synthetic hypergraphs, *bin4U* and *bin3U* are 4-uniform and 3-uniform hypergraphs, respectively, generated using state-of-the-art hypergraph configuration model [2]. *pref3U* is a 3-uniform hypergraph generated using the hypergraph preferential-attachment model [5] with parameter $p = 0.5$, where p is the probability of a new node being preferentially attached to existing nodes in the hypergraph. The node degrees in *pref3U* approximately follows a power-law distribution with exponent = 2.2. Among real-world hypergraphs, *enron* is a hypergraph of emails, where each email correspondence is a hyperedge and users are nodes [11]. We derive the *contact* (in a school) dataset from a graph where each maximal clique is viewed as a hyperedge and individuals are nodes [11]. In the *congress* dataset, nodes are congress-persons and each hyperedge comprises of sponsors and co-sponsors (supporters) of a legislative bill [11]. In *dblp*, nodes are authors and a hyperedge consists of authors in a publication recorded on DBLP [11]. Similarly, *aminer* consists of authors and publications recorded on Aminer [52].

6.1 Effectiveness of Local-core Algorithm

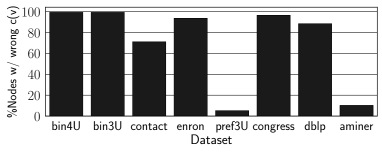
Exp-1: Novelty & importance of hypergraph h-index. We demonstrate the novelty of the proposed hypergraph h-index (Definition 4) by showing that a direct adaptation of graph h-index (Definition 2) without any core correction, that is, *running the local algorithm from [42, 45] may produce incorrect hypergraph core-numbers*. Figure 4(a) depicts that a local algorithm that only considers graph h-index without adopting our novel **Core-correction** (§ 3.3) generates incorrect core-numbers for at least 90% nodes on *bin4U*, *bin3U*, and *congress*. On *contact*, *enron*, *pref3U*, *dblp*, and *aminer*, core-numbers for at least 15%, 26%, 5%, 17%, and 10% nodes are incorrect, respectively. As nodes in *bin4U*, *bin3U*, and *congress* have relatively higher mean($|N(v)|$) (Table 1), there are more correlated neighbors in these datasets. Incorrect h-indices of correlated neighbors have a domino-effect: A few nodes with wrong h-values, unless corrected, may cause all their neighbors to have wrong h-values, which may in turn cause the neighbors' neighbors to have wrong h-values, and so on. Unless all such correlated nodes are corrected, almost all nodes eventually end up with wrong core-numbers.

We also compare the average error in the core-number estimates at each iteration by graph h-index and our hypergraph h-index. Here, avg. error at iteration n $= \sum_{u \in V}(h^{(n)}(u) - core(u))/|V|$. Figures 4(b)-(c) show avg. errors incurred at the end of each iteration on *enron* and *bin4U*. At initialization, both indices have the same error on a specific dataset. For both indices, avg. error at a given iteration is less than or equal to that in the previous iteration. However at higher iterations, hypergraph h-index incurs less avg. error compared to graph h-index. At termination, although hypergraph h-index produces correct core-numbers, graph h-index has non-zero avg. error. We notice similar trends in other datasets, however for the same iteration number, graph h-index produces higher avg. error on *bin4U* than that on *enron*. This is due to more number of correlated neighbors in *bin4U* than that in *enron* as stated earlier.

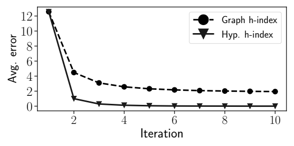
Exp-2: Convergence of Local-core. Figure 4(d) shows that as # iterations increases in our **Local-core** algorithm (§ 3.3), the percentage of nodes with correctly converged core-numbers increases. The number of iterations for convergence depend on hypergraph structure and computation ordering of nodes. As *pref3U*, *dblp*, and *enron* have more nodes with fewer correlated neighbors, more nodes achieve correct core-numbers after the first iteration. On *pref3U* 98% nodes already converge by iteration 2. This observation also suggests that one can terminate **Local-core** algorithm early at the expense of a fraction of incorrect results on *pref3U*.

6.2 Efficiency Evaluation

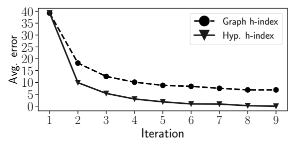
We evaluate the efficiency of our proposed algorithms: **Peel**, **E-Peel**, **Local-core(OPT)**, and **local-core(P)** vs. baselines: **Clique-Graph-Local** and **Bipartite-Graph-Local**. The baselines **Clique-Graph-Local** and **Bipartite-Graph-Local** consider clique graph and bipartite graph representations (§2.2), respectively, of the hypergraph and then apply local algorithms [41, 42] for graph core decomposition. From Figures 6(a)-(b), which report end-to-end running times, we find that **(1) E-Peel** is more efficient than **Peel** on all datasets. **(2)**



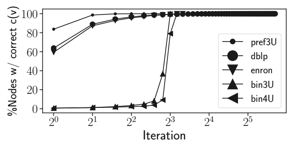
(a) Ineffectiveness of graph h-index



(b) Average error of hyp. and graph h-index on *Enron*



(c) Average error of hyp. and graph h-index on *bin4U*



(d) Convergence of Local-core

Figure 4: **Effectiveness evaluation of hypergraph h-index and Local-core**

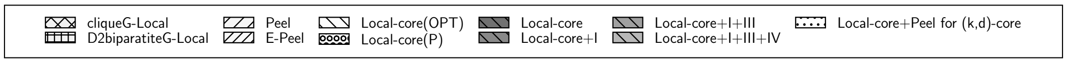
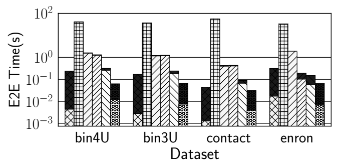
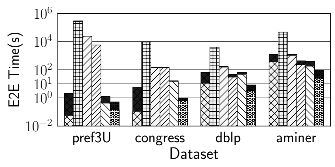


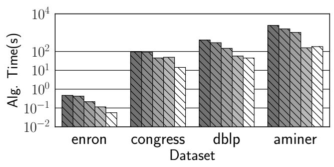
Figure 5: **Legends for Figures 6 and 7**



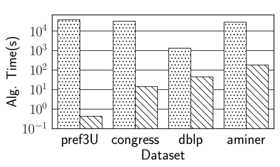
(a)



(b)

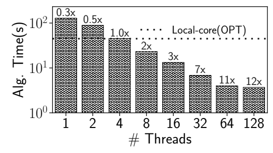


(c)

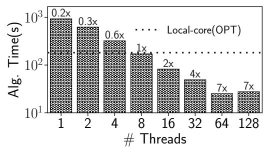


(d)

Figure 6: **(a)-(b) End-to-end (E2E) running time of our algorithms: Peel, E-Peel, Local-core(OPT), Local-core(P) with 64 Threads vs. those of baselines: Clique-Graph-Local and Distance-2 Bipartite-Graph-Local. End-to-end (E2E) running time = data structure initialization time (shaded with dark-black on top of each bar) + algorithm's execution time. (c) Impact of the four optimizations to Local-core algorithm's execution time. Here, Local-core(OPT) = Local-core + Optimizations-I + II +III + IV. (d) Local-core+Peel algorithm's execution time for (neighborhood, degree)-core decomposition vs. Local-core(OPT) for neighborhood-core decomposition.**



(a) *dblp*



(b) *aminer*

Figure 7: **Local-core(P) algorithm's execution time. Local-core(P) achieves up to 12x speedup compared to sequential Local-core(OPT).**

Local-core(OPT) is more efficient than **Peel** on all datasets. **(3) Local-core(P) is the most efficient algorithm among all the datasets. (4)** The baseline **Bipartite-Graph-Local** is the least efficient among all algorithms. This is because reducing the hypergraph to bipartite graph inflates the number of graph nodes and edges (§1), and distance-2 core decomposition [41] on this inflated graph significantly increases the running time (§2.2). **Clique-Graph-Local**'s end-to-end running time is comparable to **Local-core(OPT)** (e.g., slightly more efficient on *contact* and *congress*, while less efficient on *enron* and *aminer*). This is because reduction to clique graph also inflates the problem size (§1), but graph-based h-index computation [42] on the clique graph does not require any core-correction. However, clique graph decomposition gives a different decomposition than ours and in certain applications (as shown in §2.2, §7) our decomposition is more desirable than clique graph decomposition.

Various algorithms require different data structure initialization times, e.g., finding neighbors and incidence hyperedges for all nodes (all algorithms), creating bipartite graph (**Bipartite-Graph-Local**), clique graph (**Clique-Graph-Local**), and CSR (**Bipartite-Graph-Local**, **Clique-Graph-Local**, **Local-core(OPT)**, **Local-core(P)**). In end-to-end running times (Figures 6(a)-(b)), we included initialization times of data structures (shaded with dark-black on top of each bar). We also report initialization times on our largest *aminer* dataset separately in Table 2. We parallelize CSR construction, finding neighbors and incidence hyperedges for nodes using OpenMP with dynamic scheduling [21]. Parallel initialization time of data structures and execution time of **Local-core(P)** are 66 sec and 25 sec, respectively, on *aminer*, thus end-to-end time being 91 sec.

Exp-3: Efficiency of E-Peel. As stated in §3.2, the speedup of **E-Peel** over **Peel** is related to α, which is the ratio of the $\#|N(u)|$ queries made by **E-Peel** to that of **Peel**. In Figures 6(a)-(b), **E-Peel** achieves the highest speedup (17x compared to **Peel**) on *enron* because $\alpha = 0.35$ is the smallest on this dataset.

In remaining experiments (Figures 6(c)-(d), 7), we compare similar algorithms that have same initialization time on a dataset, thus we compare these algorithms' execution times ('Alg. Time' on y-axis).

Exp-4: Efficiency and impact of optimizations to Local-core. We next analyze the efficiency of the proposed optimizations (§4) with respect to **Local-core** without any optimization. **Local-core+I** incorporates optimization-I to **Local-core**, **Local-core+I+III** incorporates optimization-III on top of **Local-core+I**, **Local-core+I+III+IV**

Table 2: **Data structure initialization times of all algorithms on *aminer***

Local-core(P)	Local-core (OPT)	E-Peel	Peel	Clique-G-Local	Bipartite-G-Local
66 s	510 s	195 s	198 s	806 s	793 s

incorporates optimization-III on top of **Local-core+I+III**. Finally, **Local-core(OPT)** incorporates all four optimizations. Figures 6(c) shows the execution times of **Local-core**, **Local-core+I**, **Local-core+I+III**, **Local-core+I+III+IV**, and **Local-core(OPT)** on four representative datasets. On all hypergraphs, we observe that adding each optimization generally reduces the execution time.

Exp-5: Impact of parallelization. We test the parallelization performance of **Local-core(P)** by varying #threads (Figure 7). Adding 64-128 threads reduces the overall execution time up to 7-12x on larger *dblp* and *aminer* datasets. We measure execution times of **Local-core(P)** without load-balancing (§4), where load-balancing speeds up **Local-core(P)** up to 1.8x on *aminer* and *dblp*.

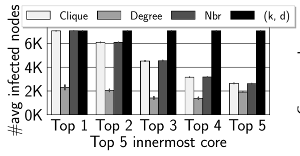
Expt-6: Efficiency of Local-core+Peel for (neighborhood, degree)-core decomposition. Figure 6(d) shows that the **Local-core+Peel** algorithm for (k, d)-core decomposition takes more time than **Local-core(OPT)** on larger datasets. This is expected since the number of possible (k, d)-cores is $O(k_{max} \cdot d_{max})$, whereas the number of neighborhood-based cores is only k_{max} (§5). As the output size increases, the algorithm **Local-core+Peel**, on top of running **Local-core(OPT)**, explicitly constructs subhypergraphs $H[V_k]$ and peels $H[V_k]$ based on degree-dimension, for all nbr core-numbers k.

7 APPLICATIONS AND CASE STUDIES

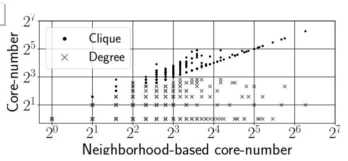
7.1 Influence Spreading and Intervention

We consider the *SIR* diffusion process [36]: Initially, all nodes except one—called a *seed*—are at the *susceptible* state. The seed node is initially at the *infectious* state. At each time step, each infected node infects its susceptible neighbors with probability β and then becomes *immunized*. Once a node is immunized, it is never re-infected.

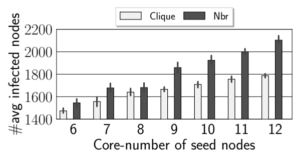
Inner-cores contain influential spreaders (Figure 8(a-c)). For each decomposition method, we select the top-5 innermost cores (on the x-axis), and for each core, we run the SIR model (β=0.3) from each of 100 uniformly selected seed nodes in that core. For (k, d)-cores, we choose inner cores with top-5 k values, where for each k, we select seed nodes with the maximum d-value. Figure 8(a) shows the #infected nodes averaged per seed node from a core. **(1)** The avg. #infected nodes generally decreases as we move from inner to outer cores, implying that inner cores contain better quality seeds. **(2)** Seeds selected according to (k, d)-core decomposition outperform seeds selected via other decomposition, indicating that our *(k, d)-core decomposition produces the best-quality seeds for maximizing diffusion*. **(3)** Seeds selected via degree-based decomposition have the lowest spread. **(4)** The quality of seeds derived from the neighborhood and clique graph decompositions have similar spread, the reason being that their inner cores have higher similarity in terms of constituent nodes, while outer cores from these two decompositions are quite different in *dblp*. Figure 8(b) highlights this difference by comparing core-numbers of 1000 randomly selected nodes according to different methods. We notice a linear correlation between the neighborhood and clique graph decomposition-based core-numbers



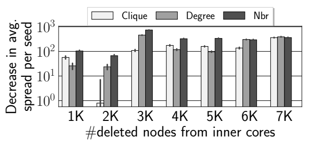
(a) *dblp*



(b) *dblp*



(c) *dblp*



(d) *dblp*

Figure 8: (a) The avg. #infected nodes decreases as seeds are selected from innermost to outer cores. (b) Neighborhood decomposition core-number vs. degree and clique graph decomposition core-numbers of nodes. (c) Neighborhood-based decomposition outperforms clique-based method when seeds are selected from outer cores. (d) The decrease in avg. expected #infected nodes per seed is the highest when the top-k nodes are deleted via neighborhood-based decomposition.

for nodes in inner cores, but such linear correlation diminishes as we consider nodes from outer cores. We do not observe any correlation between the neighborhood and degree-based core-numbers. **(5)** Neighborhood-based method outperforms clique graph when seed nodes are selected from relatively outer cores (Figure 8(c)).

Deleting inner-cores for the maximum intervention in spreading (Figure 8(d)). We devise an intervention strategy to disrupt diffusion, which is critical in mitigating the spread of contagions in epidemiology, limiting the spread of misinformation, or blocking competitive campaigns in marketing. The nodes in inner cores according to a specific decomposition method (neighborhood, degree, or clique) are considered more important than nodes in outer cores. We select the top-k most important nodes (on the x-axis) according to a specific decomposition, then we delete those nodes and incident hyperedges. For fairness across different decompositions, we consider the same set of seeds for all – seeds are selected from outside the union of deleted node sets according to three decompositions. To determine the effectiveness of such deletion strategy, on the y-axis, we report the decrease in average spread (expected number of infected nodes) per seed. We observe that when deleting up to 6K nodes, deletion of important nodes via neighborhood-based decomposition always results in a maximum decrease of spread, compared to the same done via other decompositions (degree and clique). Beyond 6K nodes, each decomposition method deletes a significant number of important nodes according to that decomposition, and the quality of seeds (which are from outside deleted regions) also degrades; thus, all approaches cause a similar decrease in spread. This result shows that our *neighborhood-based decomposition produces the best order of important nodes for deleting a limited number of them, while causing the maximum intervention in spreading*.

7.2 Densest SubHypergraph Discovery

The degree-densest subgraph is a subgraph with the maximum average node-degree among all subgraphs of a given graph [14, 19, 26], which may correspond to communities [17] and echo chambers [37] in social networks, brain regions responding to stimuli [40]. Following same principle, we define a new notion of densest subhypergraph, called the *volume-densest subhypergraph*, based on the number of neighbors of nodes in a hypergraph. The **volume-density** $\rho^N[S]$ of a subset $S \subseteq V$ of nodes in a hypergraph $H = (V, E)$ is defined as the ratio of the summation of neighborhood sizes of all nodes $u \in S$ in the induced subhypergraph $H[S]$ to the number of nodes in $H[S]$.

$$\rho^N[S] = \frac{\sum_{u \in S} |N_S(u)|}{|S|} \tag{9}$$

The **volume-densest subhypergraph** is a subhypergraph which has the largest volume-density among all subhypergraphs.

Approximation algorithm. Inspired by Charikar [14], our approach follows the peeling paradigm: In each round, we remove the node with the smallest number of neighbors in the current subhypergraph. In particular, we sort the nodes in ascending order of their neighborhood-based core-numbers, obtained from any neighborhood-based core-decomposition algorithm (**Peel**, **E-Peel**, **Local-core(P)**, or **Local-core(OPT)**). We peel nodes in that order. Among nodes with the same core-number, the one with the smallest number of neighbors in the current subhypergraph is peeled earlier. We finally return the subhypergraph that achieves the largest volume-density.

THEOREM 6. *Our volume-densest subhypergraph discovery algorithm returns* $(d_{pair}(d_{card} - 2) + 2)$*-approximate densest subhypergraph if the maximum cardinality of a hyperedge is* d_{card} *and the maximum number of hyperedges between a pair of nodes is* d_{pair} *in the input hypergraph.*

The proof is given in our extended version [4]. Notice that $d_{pair} = 2$ if the hypergraph is a graph and our result gives 2-approximation guarantee for the densest subgraph discovery [14].

Case study: Meetup dataset. We extract all events with < 100 participants from the Nashville meetup dataset [7]. The extracted hypergraph contains 24 115 nodes (participants) and 11 027 hyperedges (events) organized by various interest groups. We compute and analyze the volume-densest subhypergraph, degree-densest subhypergraph [31], and degree-densest subgraph [14] of clique graph (Table 3). **(1)** The degree-densest subhypergraph contains casual, frequent gatherings, each having less participants, from only one socializing group (*Bellevue Meetup: Meet new Friends*). **(2)** However, both the degree-densest subgraph of the clique graph and the volume-densest subhypergraph contain events involving multiple technical groups that arrange meetups about diverse technical themes. **(3)** Despite finding events of technical themes, the volume-densest subhypergraph finds a different set of events than the degree-densest subgraph of the clique graph. The reason is that the degree-density criterion used to extract these events from the clique graph is different from volume-density. The degree-densest subgraph of the clique graph, despite having a high degree-density (100.7), when projected to a subhypergraph (by projecting cliques to hyperedges), has a low volume-density (0.4). We also find these technical events to be less popular, having 5 participants on avg., compared to those returned by the volume-densest subhypergraph (avg. 78 participants).

Table 3: A summary of volume-densest subhyp., degree-densest subhyp., and degree-densest subgraph of clique graph on Meetup dataset [7]

	Vol.-den. subhyp.	**Deg.-den. subg. of clique g.**	**Deg.-den. subhyp.**
# Events	**27**	17	26
Vol.-density	**116.9**	0.4 (projected to subhyp.)	9.3
Example events	Identity and Access Controls Landscape in .NET; Web scraping in Python; Regulatory Env. Around Blockchain	Field Trip w/ Genealogical Society; Monthly Meeting: Civic Tech; Nashville (Nv) PHP User Group	Trivia Night ⓐ Plantation Pub; Trivia Night ⓐ Three Stones Pub; Dinner ⓐ Dalton
Organizing groups	Nv .NET User Group; Data Science Nv.; Nv. Blockchain User Group	State & Local Govt. Dev Network; Dev Launchpad; Nv. PHP User Group	Bellevue Meetup: Meet new Friends

8 RELATED WORK

Recently, there has been a growing interest in hypergraph data management [20, 34, 38, 49, 51, 55]. As we focus on core decomposition, we refer to recent surveys [10, 18, 39, 53] for a general exposition.

There are relatively few works on hypergraph core decomposition. [46] propose a peeling algorithm to find the maximal-degree based k-core of a hypergraph. [33, 49] discuss parallel implementations of degree-based hypergraph core computation based on peeling approach. Sun et al. [50] propose a fully dynamic approximation algorithm that maintains approximate degree-based core-numbers of an unweighted hypergraph. Gabert et al. [23] study degree-based core maintenance in dynamic hypergraphs, and propose a parallel (shared-memory) local algorithm. None of these works explore our neighborhood-based hypergraph core decomposition, which is different from degree-based hypergraph core computation (§1). Existing approaches for degree-based core decomposition cannot be easily adapted for neighborhood-based hypergraph core computation (§3).

Related work on graph cores are discussed in our full version [4].

9 CONCLUSIONS

We introduced neighborhood-cohesive core decomposition of hypergraphs, having desirable properties such as **Uniqueness** and **Core-containment**. We then proposed three algorithms: **Peel**, **E-Peel**, and novel **Local-core** for hypergraph core decomposition. Empirical evaluation on synthetic and real-world hypergraphs showed that the novel **Local-core** with optimizations and parallel implementation is the most efficient among all proposed and baseline algorithms. Our proposed decomposition is more effective than the degree and clique graph-based decompositions in intervening diffusion. Case studies illustrated that our novel volume-densest subhypergraphs capture differently important meetup events. Finally, we developed a new hypergraph-core model, *(neighborhood, degree)-core* by considering both neighborhood and degree constraints, designed decomposition algorithm **Local-core+Peel**, and depicted its superiority in diffusion spread. In future, we shall design efficient algorithms for *(neighborhood, degree)-core* decomposition.

ACKNOWLEDGMENTS

Arijit Khan acknowledges support from the Novo Nordisk Foundation grant NNF22OC0072415. Bishwamittra Ghosh acknowledges support from National Research Foundation Singapore NRF-NRFFAI1-2019-0004 , CREATE (DesCartes), MoE Singapore Tier 2 grant MOE-T2EP20121-0011 and Tier 1 grant R-252-000-B59-114.

REFERENCES

[1] J. Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat, and Alessandro Vespignani. 2005. Large scale networks fingerprinting and visualization using the k-core decomposition. In *Advances in Neural Information Processing Systems (NeurIPS)*. MIT Press, 41–50.

[2] Naheed Anjum Arafat, Debabrota Basu, Laurent Decreusefond, and Stéphane Bressan. 2020. Construction and random generation of hypergraphs with prescribed degree and dimension sequences. In *Database and Expert Systems Applications (DEXA) (Lecture Notes in Computer Science)*, Vol. 12392. Springer, 130–145.

[3] Naheed Anjum Arafat, Arijit Khan, Arpit Kumar Rai, and Bishwamittra Ghosh. 2022. Our code and datasets. https://github.com/toggled/vldbsubmission/.

[4] Naheed Anjum Arafat, Arijit Khan, Arpit Kumar Rai, and Bishwamittra Ghosh. 2023. Neighborhood-based Hypergraph Core Decomposition. *arXiv preprint arXiv:2301.06426* (2023).

[5] Chen Avin, Zvi Lotker, Yinon Nahum, and David Peleg. 2019. Random preferential attachment hypergraph. In *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. 398–405.

[6] Mohammad A. Bahmanian and Mateja Sajna. 2015. Connection and separation in hypergraphs. *Theory and Applications of Graphs* 2, 2 (2015), 5.

[7] Stephen Bailey. 2017. Nashville meetup network. https://www.kaggle.com/datasets/stkbailey/nashville-meetup.

[8] Vladimir Batagelj, Andrej Mrvar, and Matjaž Zaveršnik. 1999. Partitioning approach to visualization of large graphs. In *Graph Drawing (Lecture Notes in Computer Science)*, Vol. 1731.

[9] Vladimir Batagelj and Matjaž Zaveršnik. 2011. Fast algorithms for determining (generalized) core groups in social networks. *Advances in Data Analysis and Classification* 5, 2 (2011), 129–145.

[10] Federico Battiston, Giulia Cencetti, Iacopo Iacopini, Vito Latora, Maxime Lucas, Alice Patania, Jean-Gabriel Young, and Giovanni Petri. 2020. Networks beyond pairwise interactions: structure and dynamics. *Physics Reports* 874 (2020), 1–92.

[11] Austin R. Benson, Rediet Abebe, Michael T. Schaub, Ali Jadbabaie, and Jon Kleinberg. 2018. Simplicial closure and higher-order link prediction. *Proceedings of the National Academy of Sciences* 115, 48 (2018), E11221–E11230.

[12] Claude Berge. 1989. *Hypergraphs - combinatorics of finite sets*. North-Holland mathematical library, Vol. 45. North-Holland.

[13] Francesco Bonchi, Arijit Khan, and Lorenzo Severini. 2019. Distance-generalized core decomposition. In *The International Conference on Management of Data (SIGMOD)*. ACM, 1006–1023.

[14] Moses Charikar. 2000. Greedy approximation algorithms for finding dense components in a graph. In *Approximation Algorithms for Combinatorial Optimization, Third International Workshop (Lecture Notes in Computer Science)*, Vol. 1913. Springer, 84–95.

[15] James Cheng, Yiping Ke, Shumo Chu, and M. Tamer Özsu. 2011. Efficient core decomposition in massive networks. In *IEEE International Conference on Data Engineering (ICDE)*. 51–62.

[16] Megan Dewar, Kirill Ternovsky, Benjamin Reiniger, John Proos, Pawel Pralat, Xavier Pérez-Giménez, and John Healy. 2018. Subhypergraphs in non-uniform random hypergraphs. *Internet Math.* 2018 (2018).

[17] Yon Dourisboure, Filippo Geraci, and Marco Pellegrini. 2009. Extraction and classification of dense implicit communities in the web graph. *ACM Transactions on the Web* 3, 2 (2009), 1–36.

[18] Tina Eliassi-Rad, Vito Latora, Martin Rosvall, and Ingo Scholtes. 2021. Higher-order graph models: from theoretical foundations to machine learning (Dagstuhl Seminar 21352). *Dagstuhl Reports* 11, 7 (2021), 139–178.

[19] Yixiang Fang, Kaiqiang Yu, Reynold Cheng, Laks V. S. Lakshmanan, and Xuemin Lin. 2019. Efficient algorithms for densest subgraph discovery. *Proc. VLDB Endow.* 12, 11 (2019), 1719–1732.

[20] Pit Fender and Guido Moerkotte. 2013. Counter strike: generic top-down join enumeration for hypergraphs. *Proc. VLDB Endow.* 6, 14 (2013), 1822–1833.

[21] Andrew Finlayson. [n.d.]. OpenMP Scheduling. http://www.inf.ufsc.br/~bosco.sobral/ensino/ine5645/OpenMP_Dynamic_Scheduling.pdf.

[22] Christoph Flamm, Bärbel M.R. Stadler, and Peter F. Stadler. 2015. Generalized topologies: hypergraphs, chemical reactions, and biological evolution. In *Advances in Mathematical Chemistry and Applications*. Bentham Science, 300–328.

[23] Kasimir Gabert, Ali Pinar, and Ümit V. Çatalyürek. 2021. Shared-Memory Scalable k-Core Maintenance on Dynamic Graphs and Hypergraphs. In *IEEE International Parallel and Distributed Processing Symposium*. 998–1007.

[24] Edoardo Galimberti, Francesco Bonchi, Francesco Gullo, and Tommaso Lanciano. 2020. Core decomposition in multilayer networks: theory, algorithms, and applications. *ACM Trans. Knowl. Discov. Data* 14, 1 (2020), 11:1–11:40.

[25] Thomas Gaudelet, Noël Malod-Dognin, and Natasa Przulj. 2018. Higher-order molecular organization as a source of biological function. *Bioinformatics* 34, 17 (2018), i944–i953.

[26] Andrew V. Goldberg. 1984. *Finding a maximum density subgraph*. University of California Berkeley.

[27] Ronald L. Graham. 1969. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics* 17, 2 (1969), 416–429.

[28] Ronald L. Graham, Martin Grötschel, and László Lovász (Eds.). 1996. *Handbook of Combinatorics (Vol. 2)*. MIT Press.

[29] Yi Han, Bin Zhou, Jian Pei, and Yan Jia. 2009. Understanding importance of collaborations in co-authorship networks: a supportiveness analysis approach. In *SIAM International Conference on Data Mining (SDM)*. 1112–1123.

[30] Jorge E. Hirsch. 2005. An index to quantify an individual's scientific research output. *Proceedings of the National academy of Sciences* 102, 46 (2005), 16569–16572.

[31] Shuguang Hu, Xiaowei Wu, and T.-H. Hubert Chan. 2017. Maintaining densest subsets efficiently in evolving hypergraphs. In *ACM International Conference on Information and Knowledge Management (CIKM)*. 929–938.

[32] Jin Huang, Rui Zhang, and Jeffrey Xu Yu. 2015. Scalable hypergraph learning and processing. In *IEEE International Conference on Data Mining (ICDM)*. 775–780.

[33] Jiayang Jiang, Michael Mitzenmacher, and Justin Thaler. 2017. Parallel peeling algorithms. *ACM Transactions on Parallel Computing* 3, 1 (2017), 1–27.

[34] Igor Kabiljo, Brian Karrer, Mayank Pundir, Sergey Pupyrev, Alon Shalita, Yaroslav Akhremtsev, and Alessandro Presta. 2017. Social hash partitioner: a scalable distributed hypergraph partitioner. *Proc. VLDB Endow.* 10, 11 (2017), 1418–1429.

[35] Wissam Khaouid, Marina Barsky, Venkatesh Srinivasan, and Alex Thomo. 2015. K-core decomposition of large networks on a single PC. *Proc. VLDB Endow.* 9, 1 (2015), 13–23.

[36] Maksim Kitsak, Lazaros K. Gallos, Shlomo Havlin, Fredrik Liljeros, Lev Muchnik, H. Eugene Stanley, and Hernán A. Makse. 2010. Identification of influential spreaders in complex networks. *Nature physics* 6, 11 (2010), 888–893.

[37] Laks V.S. Lakshmanan. 2022. On a quest for combating filter bubbles and misinformation. In *The International Conference on Management of Data (SIGMOD)*. ACM, 2.

[38] Geon Lee, Jihoon Ko, and Kijung Shin. 2020. Hypergraph motifs: concepts, algorithms, and discoveries. *Proc. VLDB Endow.* 13, 11 (2020), 2256–2269.

[39] Geon Lee, Jaemin Yoo, and Kijung Shin. 2022. Mining of Real-World Hypergraphs: Patterns, Tools, and Generators. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM)*. 5144–5147.

[40] Robert Legenstein, Wolfgang Maass, Christos H. Papadimitriou, and Santosh S. Vempala. 2018. Long term memory and the densest k-subgraph problem. In *Innovations in Theoretical Computer Science Conference (ITCS) (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 94. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 57:1–57:15.

[41] Qing Liu, Xuliang Zhu, Xin Huang, and Jianliang Xu. 2021. Local algorithms for distance-generalized core decomposition over large dynamic graphs. *Proc. VLDB Endow.* 14, 9 (2021), 1531–1543.

[42] Linyuan Lü, Tao Zhou, Qian-Ming Zhang, and H. Eugene Stanley. 2016. The H-index of a network node and its relation to degree and coreness. *Nature communications* 7, 1 (2016), 1–7.

[43] Fragkiskos D. Malliaros, Maria-Evgenia G. Rossi, and Michalis Vazirgiannis. 2016. Locating influential nodes in complex networks. *Scientific Reports* 6, 19307 (2016).

[44] Irene Malvestio, Alessio Cardillo, and Naoki Masuda. 2020. Interplay between k-core and community structure in complex networks. *Scientific Reports* 10, 14702 (2020).

[45] Alberto Montresor, Francesco De Pellegrini, and Daniele Miorandi. 2012. Distributed k-core decomposition. *IEEE Transactions on parallel and distributed systems* 24, 2 (2012), 288–300.

[46] Emad Ramadan, Arijit Tarafdar, and Alex Pothen. 2004. A hypergraph model for the yeast protein complex network. In *International Parallel and Distributed Processing Symposium*.

[47] Ahmet Erdem Saríyüce, Buğra Gedik, Gabriela Jacques-Silva, Kun-Lung Wu, and Ümit V. Çatalyürek. 2013. Streaming algorithms for k-core decomposition. *Proc. VLDB Endow.* 6, 6 (2013), 433–444.

[48] Ahmet Erdem Sariyüce, C. Seshadhri, Ali Pinar, and Ümit V. Çatalyürek. 2015. Finding the hierarchy of dense subgraphs using nucleus decompositions. In *The International Conference on World Wide Web (WWW)*. ACM.

[49] Julian Shun. 2020. Practical parallel hypergraph algorithms. In *The ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 232–249.

[50] Bintao Sun, T.-H. Hubert Chan, and Mauro Sozio. 2020. Fully dynamic approximate k-core decomposition in hypergraphs. *ACM Trans. Knowl. Discov. Data* 14, 4 (2020), 39:1–39:21.

[51] Justin Sybrandt, Ruslan Shaydulin, and Ilya Safro. 2022. Hypergraph partitioning with embeddings. *IEEE Trans. Knowl. Data Eng.* 34, 6 (2022), 2771–2782.

[52] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. ArnetMiner: extraction and mining of academic social networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 990–998.

[53] Leo Torres, Ann Sizemore Blevins, Danielle S. Bassett, and Tina Eliassi-Rad. 2021. The why, how, and when of representations for complex systems. *SIAM*

Rev. 63, 3 (2021), 435–485.

[54] Jia Wang and James Cheng. 2012. Truss decomposition in massive networks. *Proc. VLDB Endow.* 5, 9 (2012), 812–823.

[55] Joyce Jiyoung Whang, Rundong Du, Sangwon Jung, Geon Lee, Barry L. Drake, Qingqing Liu, Seonggoo Kang, and Haesun Park. 2020. MEGA: multi-view semi-supervised clustering of hypergraphs. *Proc. VLDB Endow.* 13, 5 (2020), 698–711.

[56] Xin Xia, Hongzhi Yin, Junliang Yu, Qinyong Wang, Lizhen Cui, and Xiangliang Zhang. 2021. Self-supervised hypergraph convolutional networks for session-based recommendation. In *AAAI Conference on Artificial Intelligence*. 4503–4511.

[57] Jaewon Yang and Jure Leskovec. 2015. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems* 42, 1 (2015), 181–213.