Neighborhood-based Hypergraph Core Decomposition

Naheed Anjum Arafat
National University of Singapore
Singapore
naheed_anjum@u.nus.edu

Arijit Khan
Aalborg University
Denmark
arijitk@cs.aau.dk

Arpit Kumar Rai
Indian Institute of Technology, Kanpur
India
arpitkr20@iitk.ac.in

Bishwamittra Ghosh
National University of Singapore
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 of decomposing hypergraphs such as reduction to clique graphs or bipartite graphs inflate the problem-size and are not meaningful in certain applications, while existing degree-based hypergraph decomposition does not distinguish nodes with different neighborhood sizes. Applications and our case-studies show that the proposed decomposition is more effective than degree-based decomposition in diffusion problems such as disease intervention and in extracting provably approximate and application-wise meaningful, non-trivial densest subhypergraphs. As technical contributions, we propose three algorithms: **Peel**, its efficient variant **E-Peel**, and a novel local algorithm: **Local-core** with parallel implementation. Our most efficient sequential algorithm **Local-core(OPT)** decomposes hypergraph with 27M nodes and 17M hyperedges in-memory within 16 minutes by adopting various optimizations that are unique to hypergraphs. Our parallel implementation **Local-core(P)** further expedites computation achieving at least 2-5x speedup over **Local-core(OPT)**.

PVLDB Reference Format:
Naheed Anjum Arafat, Arijit Khan, Arpit Kumar Rai, and Bishwamittra Ghosh. Neighborhood-based Hypergraph Core Decomposition. PVLDB, 14(1): XXX-XXX, 2023.
doi:XX.XX/XXX.XX

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 [49], densest subgraph discovery [15], identifying influential nodes [48], and network visualization [1, 8]. Depending on different notions of *cohesiveness*, there are several decomposition approaches: core-decomposition [9], truss-decomposition [62], nucleus-decomposition [53], etc. In this work, we are interested in decomposing *hypergraphs*, a generalization of graphs where an edge may connect more than two entities.

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

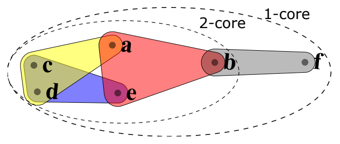


Figure 1: Neighborhood-based core decomposition of hypergraph H

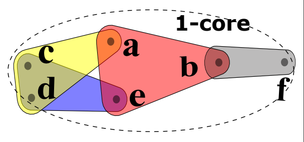


Figure 2: Degree-based hypergraph core decomposition of H

Many real-world relations consist of poly-adic entities, e.g., relations between individuals in co-authorships [33], legislators in parliamentary voting [11], items in e-shopping carts [66], proteins in protein complexes, and metabolites in a metabolic process [24, 27]. Often such relations are reduced to a clique graph or a bipartite graph for convenience (§2.2). However, these reductions may not be desirable due to two reasons. First, such reductions might not be meaningful, e.g., three authors collaborating on an article does not necessarily mean that they must have collaborated pairwise and vice versa. A pair of proteins in a certain protein-complex may not necessarily interact pairwise to create a new functional protein-complex. We show in §2.2 that the core decomposition of the clique graph or bipartite graph representation of a hypergraph may not yield the same result as the core decomposition of the hypergraph. Second, reducing a hypergraph to a clique graph or a bipartite graph *inflates* the problem-size [36]: A hypergraph in [67] with 2M nodes and 15M hyperedges is converted to a bipartite graph with 17M nodes and 1B edges. Even worse, a k-uniform hypergraph with m hyperedges causes its clique graph to have $O(mk^2)$ edges.

To this end, we propose a novel neighborhood-cohesion based approach for hypergraph core decomposition. *Neighborhood-based core decomposition* is a decomposition of 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). *In the hypergraph H in Figure 1, the node a has 4 neighbors: $\{b, c, d, e\}$. Similarly, nodes b, c, d, e, and f have 3, 3, 3, 4, and 1 neighbors, respectively. As every node has ≥ 1 neighbor, the neighborhood-based 1-core, denoted by $H[V_1]$, is the hypergraph H*

itself. The neighborhood based 2*-core is the subhypergraph* $H[V_2] = \{\{a, c, d\}, \{c, d, e\}, \{a, b, e\}\}$ *because nodes b, a, e, c, and d respectively have* 2, 4, 4, 3, *and* 3 *neighbors in* $H[V_2]$.

Motivation. The only hypergraph decomposition existing in the literature is that based on degree [51, 56]. The *degree-based core decomposition* decomposes a hypergraph into a sequence of nested maximal subhypergraphs (cores) such that every node in the k-th core has degree at least k in that core. Degree-based core decomposition does not take hyperedge sizes into consideration. As a result, nodes in the same core may have vastly different neighborhood sizes. For instance, nodes f and a have 1 and 4 neighbors, respectively, yet they belong to the same core in the degree-based decomposition, as illustrated in Figure 2. 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 [40]. Indeed from an intuitive viewpoint, node f (as a seed) propagates information (disease) to only 1 node, whereas a can do the same to 4 nodes. Thus, targeting (intervening) node a should be more important than f for devising targeted campaigns (disease intervention strategies). Clearly, we need a new measure of node-importance different from degree, and hence a new decomposition which respects that measure.

Applications. First, we demonstrate the usefulness of neighborhood-based core decomposition in diffusion-related domains [10, 55]. Specifically, we show in § 6.1 that nodes in the innermost core of our decomposition are not only the most influential in spreading information, but also the earliest adopters of diffused information. Besides, deleting an innermost core node is more likely to disrupt the spread of an infectious disease (e.g., COVID19, Ebola) compared to an outer core node. We show such an intervention to be generally effective due to decrease in connectivity and increase in the average length of shortest paths after intervention. Furthermore, an intervention based on our neighborhood-based decomposition is much more effective compared to a degree-based decomposition strategy as discussed in the *Motivation*.

Second, the proposed core-decomposition gives rise to a new type of densest subhypergraph, which we refer to as the *volume-densest subhypergraph*. Nodes in the volume-densest subhypergraph have the largest number of average neighbors (in the subhypergraph) among all subhypergraphs. Our neighborhood-based decomposition induces a node-ordering which we exploit to obtain the volume-densest subhypergraph approximately with a theoretical guarantee (§6.2). In §6.2.1, we show that the proposed volume-densest subhypergraphs capture neighborhood-cohesive regions more effectively than the existing degree-densest subhypergraphs [35]. Case-study on human protein-complexes (§6.2.2) shows that the volume-densest subhypergraph extracts complexes that participate in RNA metabolism and localization. Case-study on organizational emails (§6.2.3) shows that the volume-densest subhypergraph extracts emails about internal announcements, meetings, and employee gatherings.

Challenges. Hypergraph core decomposition is challenging because a hyperedge can relate to more than two nodes. Furthermore, a pair of nodes may be related by multiple, yet distinct hyperedges. Thus, trivial adaptation of core-decomposition algorithms for graphs to hypergraphs is difficult. For instance, 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 is expensive. In the following, we discuss the challenges associated with adopting the local approach [46], one of the most efficient methods for graph core decomposition, to hypergraphs.

Challenges in adopting a local approach [46, 50]. In this approach, a core-number estimate is updated iteratively [46] or in a distributed manner [50] for every node in a graph. The initial value of a node's core-number estimate is a known upper bound of its core-number. In subsequent rounds, this estimate is iteratively decreased based on estimates of neighboring nodes. [46] uses Hirsch's index (h-index) [34] for such an 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 with h-index at least k may not include hyperedges that partially contain other nodes. As we illustrate in §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. Therefore, while the local approach is used for computing the k-core [46] or more general (k, h)-core [45] in graphs, the same approach results in *incorrect* neighborhood-based hypergraph cores, as demonstrated empirically in §5.1.

Our contributions and roadmap. Our contributions are summarized as follows:

Novel problem and characterization (§ 2). We are the first to define and investigate the novel problem of neighborhood-based core decomposition in hypergraphs. Our core decomposition is well-defined and logical: 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 to compute neighborhood-based cores in a hypergraph, with their formal correctness and time complexity analyses. Two of them, **Peel** and its enhancement **E-Peel** adopt the classic peeling approach [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, called **Local-core** is the most efficient one, it only makes node-level local computations. Even though the existing local method [46, 50] fails to correctly find neighborhood-based core-numbers in a hypergraph, our algorithm **Local-core** applies a *novel* **Core-correction** procedure after local updates, ensuring correct core-number computations.

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.

Empirical evaluation (§ 5). Empirical evaluation on real and synthetic hypergraphs shows that the proposed algorithms are effective, efficient, and practical. **Local-core** with optimizations decomposes hypergraph with 27M nodes and 17M hyperedges within 16 minutes. Furthermore, our OpenMP parallel implementation **Local-core(P)** achieves 2-5x speedup compared to its sequential counterpart.

Applications (§ 6). In diffusion-related applications, we show our decomposition to be more effective in disrupting diffusion than other decompositions. Such diffusion-interruption is significant for intervening epidemics or selecting target-groups for marketing.

Our greedy algorithm proposed for the volume-densest subhypergraph recovery achieves a $(d_{pair}(d_{card} - 2) + 2)$-approximation guarantee, where hyperedge-cardinality and node-pair co-occurrence 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 the densest subgraph discovery [15]. The proposed volume-densest subhypergraphs capture neighborhood-cohesive regions more effectively than existing degree-densest subhypergraphs [35]. Our case-studies show that the proposed subhypergraphs captures functionally significant complexes in human cells and important emails in organizations.

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, 17, 31]. 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.

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

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

2.1 Problem Formulation

Nbr-k-core. 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 of notation, 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

There are broadly two kinds of approaches that one can adapt from the literature towards decomposing hypergraphs.

Approach-1. One may transform the hypergraph into other objects (e.g., a graph), apply existing decomposition approaches [9, 14] on that object, and then project the decomposition back to the hypergraph. For instance, a hypergraph can be transformed into a clique graph by replacing the hyperedges with cliques and then classical graph decomposition is applied (Figure 3). A hypergraph can also be

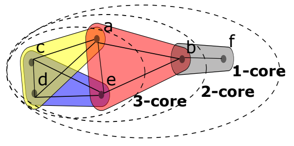


Figure 3: Core decomposition of the clique-graph of H

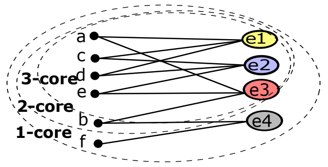


Figure 4: Distance-2 core decomposition of the bipartite graph of H

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 is applied (Figure 4). In distance-2 core-decomposition, nodes in k-core has at least k 2-hop neighbors in the subgraph. Both clique graph and bipartite graph representations inflate the problem size as discussed in the Introduction. Besides, the decomposition they yield may be different from that yielded by ours. For instance, $core(d) = 3$ in both clique graph decomposition and dist-2 bipartite graph decomposition, whereas d has core-number 2 in our decomposition.

Approach-2. Sun et al. [56] define the k-core (i.e., *deg-k-core*) of a hypergraph from the notion of node degree in a hypergraph. The *degree* $d(v)$ of a node v in hypergraph H is the number of hyperedges incident on v [12], i.e., $d(v) = |\{e \in E \mid v \in e\}|$. The *deg-k-core* H_k^{deg} of a hypergraph H is 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 take hyperedge sizes into consideration as stated in §1. Therefore, it does not necessarily yield the same decomposition as our approach, which can be seen from the difference between Figures 2 and 1. This difference originates from the fact that, unlike in a graph, the number of neighbors ($|N(v)|$) of a node v in a hypergraph is not equal to its degree.

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}. In both cases, u must have at least k neighbours in the respective subhypergraph 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}, it follows that, H_{k_1} and H_{k_2} are not maximal, and thus are not nbr-k-cores, 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 $u \in V_{k+1} \subset S$, the set S is larger than V_k, to be precise, $|S| \geq |V_k| + 1$. 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. □

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 $c$
```

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 $c$
```

3 ALGORITHMS

We propose three algorithms to exactly compute neighborhood-based hypergraph cores. The algorithms **Peel** and its efficient variant **E-Peel** are inspired by a family of peeling-based algorithms similar to graph core computations [9, 14]. The algorithm **Local-core** is inspired by a family of local approaches to graph core computation [46, 50]. 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 because hypergraphs generalize graphs and the lower-bound for graph core is naturally insufficient for our purpose. For **Local-core**, we illustrate how a direct adaptation of local algorithm [46] may lead to incorrect core computations. Hence, we devise the notions of *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. Can we delay this recomputation for some neighbors u of v? 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 in the hypergraph.

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, $|e_m| - 1 \leq c(v)$. □

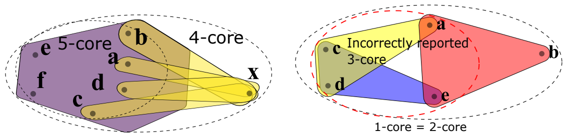


Figure 5: (a) During x's core-number computation, Algorithm 2 does not perform neighborhood recomputations and cell updates for x's neighbors $\{a, b, c, d\}$; thus saving four redundant neighborhood recomputations and cell updates. (b) For any $n > 1$, the h-index (Definition 2) of node a never reduces 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.

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* **LB**() *is known, but $N_V()$ at the current iteration is unknown.*

Example 2. *Figure 5(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 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 § 5, **E-Peel** can be up to 14x faster than **Peel**.

3.3 Local Algorithm

Although **Peel** and its more efficient variant **E-Peel** correctly computes core-numbers, they must modify the remaining hypergraph at every iteration by peeling (deleting) nodes and hyperedges. Peeling operation may impact 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 a number of optimizations, as well as parallelizable.

Naïve adoption of local algorithm in hypergraphs: a negative result. Eugene et al. [46] adopt Hirsch's index [34], 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. It is well-known that higher-order h-indices monotonically converge to the core-number of that node in the graph.

Definition 1 ($\mathcal{H}$-operator [34, 46]). *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 [46]). *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*

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

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 $h_v^{(\infty)} = c(v)$ [46, 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, thr reported $c(v)$-core can be incorrect.

Example 4. *For the hypergraph in Figure 5(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*

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 $c$
```

$3 = h_v^{\infty}$ *in Figure 5(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 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* $\exists H^+(v)$ *contains 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).

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:         $h_v^{(n)} \leftarrow h_v^{(n)} - 1$
```

To compute $LCCSAT(h_v^{(n)})$, 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 5(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 1 and 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 for any $v \in V$, both sequences $(h_v^{(n)})$ and $(\hat{h}_v^{(n)})$ are finite and have the same limit by Theorem 3. 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, we only provide proof sketches, while the formal proofs are given in our extended version [3].

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:* $\lim_{n\to\infty} h_v^{(n)} = \lim_{n\to\infty} \hat{h}_v^{(n)}$.

Proof sketch. Construct a new sequence $(h\hat{h}_v)$ by interleaving components from $(h_v^{(n)})$ and $(\hat{h}_v^{(n)})$ as the following:

$$(h\hat{h}_v) = (h_v^{(0)}, \hat{h}_v^{(0)}, h_v^{(1)}, \hat{h}_v^{(1)}, h_v^{(2)}, \hat{h}_v^{(2)} \ldots)$$

It can be verified that this *interleave sequence* [59, Defn 680], denoted as $(h\hat{h}_v)$, is monontonically non-increasing and is lower-bounded by 0. By Monotone convergence theorem [7], $(h\hat{h}_v)$ has a limit. An interleave sequence has a limit if and only if its constituent sequence pairs are convergent and have the same limit [59]. Since $(h\hat{h}_v)$ has a limit, the sequences $(h_v^{(n)})$ and $(\hat{h}_v^{(n)})$ converges to 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: $\hat{h}_v^{(\infty)} = c(v)$.*

Proof sketch. Given any $v \in V$, we first show that $c(v) \geq \hat{h}_v^{\infty}$, followed by showing $\hat{h}_v^{\infty} \geq c(v)$. The former holds since it can be proved that $v \in \hat{h}_v^{\infty}$-core. However, by definition of core-number, $c(v)$ is the largest integer for which $v \in c(v)$-core. Thus, $c(v) \geq \hat{h}_v^{\infty}$. Next, $\hat{h}_v^{\infty} \geq c(v)$ can be proved by induction on the number of iteration. Hence, the theorem.

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

4 OPTIMIZATION AND PARALLELIZATION OF THE LOCAL-CORE ALGORITHM

We propose four optimizations to further 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 from [45], where similar optimizations are proposed for graph (k, h)-core decomposition to improve convergence and eliminate redundant computations, respectively, *though such optimizations have not been adopted in earlier hypergraph-related works.*

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 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 $c$
```

Optimization-I (Compressed hypergraph representation): Local-core makes two primitive neighborhood queries on hypergraph structures: neighbors enumeration (for h-index computation) and incident-hyperedges enumeration (during **Core-correction**). A naive implementation keeps an $N \times N$ matrix for neighbors counting queries and an $N \times E$ matrix for incident-hyperedge queries. However, storing such matrices in main memory is not only expensive, but also unnecessary for large hypergraphs, since these matrices are sparse in practice. Hence, it is imperative to adopt a compressed sparse representation for these matrices. *Compressed sparse row* (CSR) is one such widely-used representation in scientific computations.

CSR representation for storing neighbors: We use two arrays F and N. $N[v]$ stores the starting index in F containing neighbors of node v . Neighbors of node v are stored in contiguous locations $(F[N[v]], F[N[v]+1], \ldots, F[N[v+1]-1])$. $N[v+1] - N[v]$ gives us the number of neighbors $|N(v)|$ of node v.

CSR representation for storing incident-hyperedges works analogously. Alternatively, one can use hash tables for storing a node's neighbors and incident hyperedges. In §5.2, we empirically show that **Local-core + Optimization-I** is more efficient than **Local-core** which uses hash tables on large datasets.

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 $h_u^{(n)} \geq h_v^{(n)}$ 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**. We reduce this cost to $O(d(v))$ 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 $E_e^{(n)} \geq h_v^{(n)}$. 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 $h_v^{(n)} \leq k \leq \hat{h}_v^{(n)}$, it recomputes $E^+(v)$ and $N^+(v)$ until returning $\hat{h}_v^{(n)}$ as output. The cost of computing $N^+(v)$ for every k, without any optimization, is $O(\sum_j |e_v^j|)$, where

e_v^j is the j^{th} hyperedge incident on v. Thus, the total cost of the **Core-correction** procedure, without any optimization, is $O((h_v^{(n)} - \hat{h}_v^{(n)}) \sum_j |e_v^j|)$. We reduce this cost to $O(\sum_j |e_v^j|)$ 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 $N^+(v,k)$.

$$N^+(v,k) = \begin{cases} \cup_e B[k] & k = h_v^{(n)} \\ N^+(v,k+1) \cup (\cup_e B[k]) & \hat{h}_v^{(n)} \leq k < h_v^{(n)} \end{cases} \quad (7)$$

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. This reduces the runtime complexity of **Core-correction** to $O(\sum_j |e_v^j|)$. 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.
Efficient LCCSAT computation: We return *True* immediately 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, we use the most-recent core-estimates $c[u_1], c[u_2], ..c[u_t]$ to update the node $v \in N(u_i)$'s core-number estimate $c[v]$ (and vice versa): $c[v] \leftarrow \min(\mathcal{H}(c[u_1], c[u_2], ..., c[u_t]), c[v])$. Notice that due to the min() operator, $c[v]$ and $c[u_i]$'s are non-increasing with more iterations. Assuming that $c[u_1]$ is updated before $c[v]$, $c[v]$ might decrease more if v uses the most-recent $c[u_1]$ instead of using $c[u_1]$ from the previous iteration. The reason for faster decrease is that lowering a few arguments might cause the output of $\mathcal{H}()$ to decrease as well, e.g. $\mathcal{H}(1,1,2,2) = 2$, whereas $\mathcal{H}(1,1,1,2) = 1$.
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 $\hat{h}_v^{(n)}$ is equal to $\mathbf{LB}(v)$, we can ensure that core-number $c(v) = \mathbf{LB}(v)$. In other words, 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 node v at future iterations.

Note that Liu et al. [45] 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 as we have shown in Example 4 that even if a node and its neighbors' h-indices have converged, node v may still need core correction.
Parallelization of Local-core. We propose a parallel implementation of the local algorithm, **Local-core(P)** following the shared-memory, data parallel programming paradigm. The algorithm partitions the nodes into T partitions, where T is the number of threads. Each thread is responsible for computing core-numbers of nodes in

Table 1: Dataset statistics: $|V|$ #nodes, $|E|$ #hyperedges, $d(v)$ (mean) degree of a node, $|e|$ (mean) cardinality of a hyperedges, $|N(v)|$ (mean) #neighbors per node

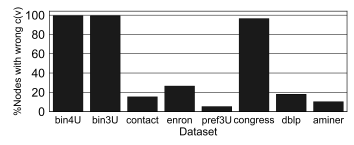
	hypergraph	$\|V\|$	$\|E\|$	$d(v)$	$\|e\|$	$\|N(v)\|$
Syn.	*bin4U*	500	12424	99.4±8.5	4±0	225.3±15.5
	bin3U	500	16590	99.5±8	3±0	164.1±11.6
	pref3U	125329	250000	5.9±915.9	3±0	4.5±412.4
Real	*enron*	4423	5734	6.8±32	5.2±5	25.3±44
	contact	242	12704	127±55.2	2.4±0.5	68.7±26.6
	congress	1718	83105	426.2±475.8	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

its own partition. To improve load-balancing, we adopt the longest-processing-time-first scheduling approach [30] such that the aggregated number of neighbors of nodes in different threads are roughly the same. **Local-core(P)** has three core differences compared to its sequential counterpart **Local-core(OPT)** (Algorithm 5).

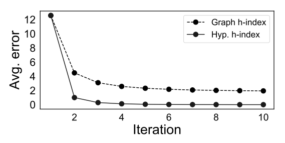
First, at iteration 0 (Line 3), every thread initializes hypergraph h-indices for its allocated nodes asynchronously in parallel. Note that concurrent computation of $|N(v)|$ and $|N(u)|$ for nodes allocated to different threads requires concurrent reads to the CSR representation. Since the CSR representation 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 corrected value $\hat{h}^{(v)}$ for its allocated nodes asynchronously in parallel (Lines 6-11). At some iteration n, the computation of $c[v]$ requires $c[u]$ of its neighbor u. The same goes for **Core-correction** procedure. One may wonder if the parallel algorithm outputs wrong core-number, since due to asynchronous processing, the computation order of $c[v]$ and $c[u]$ is not fixed any more. Interestingly, our algorithm still terminates with the correct output because none of the theorems in § 3.4 rely on any particular computation-order of nodes. The computation-order only affects the number of iterations required for convergence, not the converged value. **Finally**, **Local-core(P)** does not use optimization-IV, as we empirically find that Optimization-IV does not improve the execution time significantly.

5 EMPIRICAL EVALUATION

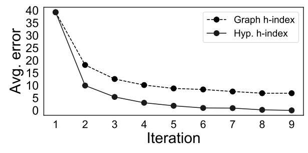
We empirically evaluate the performance of our algorithms on four synthetic and five real-world datasets (Table 1). We implement our algorithms in GNU C++11 and OpenMP API version 3.1. All experiments are conducted on a server with 80 core Intel(R) Xeon(R) 2.8GHz CPU and 128GB RAM. **Our code and datasets are at [4]**.
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 $\beta = 2.2$. 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



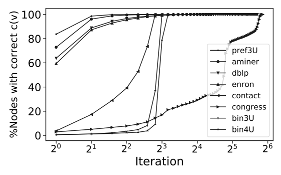
(a) Ineffectiveness of graph h-index



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

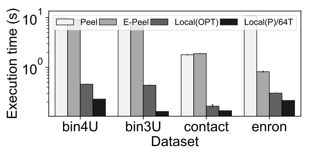


(c) Average error of hyp. and graph h-index on *bin4U*

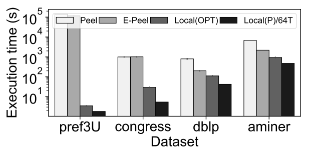


(d) Convergence of Local-core

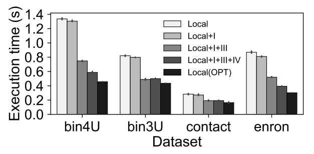
Figure 6: Effectiveness evaluation of hypergraph h-index and Local-core



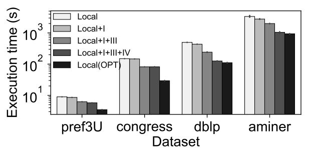
(a) Smaller datasets



(b) Larger datasets

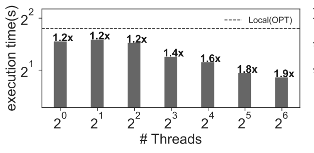


(c) Smaller datasets

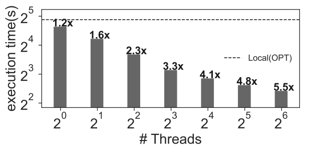


(d) Larger datasets

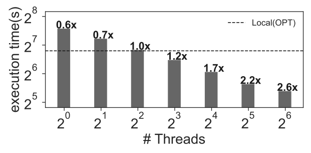
Figure 7: (a)-(b) Efficiency of Peel, E-Peel, Local-core(OPT), and local-core(P) with 64 Threads. (c)-(d) Impact of four optimizations to Local-core



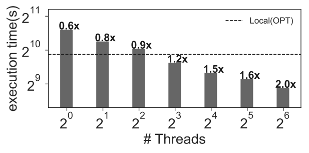
(a) *pref3U*



(b) *congress*



(c) *dblp*



(d) *aminer*

Figure 8: Efficiency of Local-core(P) on larger datasets. Local(P) achieves up to 5.5x and at least 1.9x speedup compared to sequential Local(OPT).

bill put forth in both the House of representatives and the senate [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 [58].

5.1 Effectiveness of Local-core Algorithm

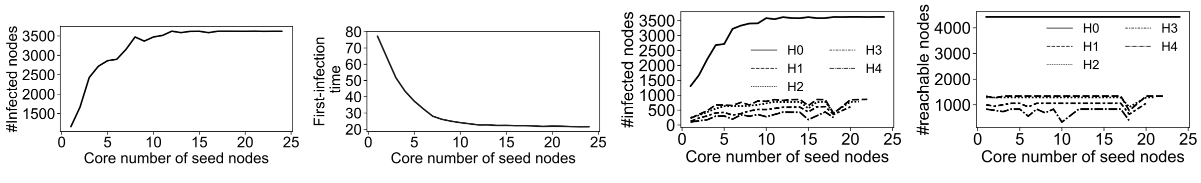
Exp-1: Novelty & importance of hypergraph h-index. We demonstrate the novelty of the proposed hypergraph h-index (Definition 4) by showing that a direct adaptation of graph h-index (Definition 2) without any core correction, that is, *running the local algorithm from [46, 50] may produce incorrect hypergraph core-numbers*. Figure 6(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 6(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. These results suggest that hypergraph h-index estimates core-numbers more accurately than graph h-index at intermediate iterations. 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 6(d) shows that as the number of iteration 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 the hypergraph structure and on the computation ordering of nodes. As *pref3U*, *dblp*, and *enron* has more nodes with fewer correlated neighbors, more nodes achieve correct core-numbers after the first iteration. On *pref3U* at least 98% nodes already converge by iteration 2. This observation also suggests that one can terminate the proposed **Local-core** algorithm early at the expense of a fraction of incorrect results for the *pref3U* hypergraph. Notice that even though *contact* has only 3% nodes with correct core-numbers at the first iteration, its convergence rate is steeper than that of *bin3U* and *bin4U*, since *contact* has a lower mean($|N(v)|$), and thus lesser correlated neighbors compared to that for *bin3U* and *bin4U*.

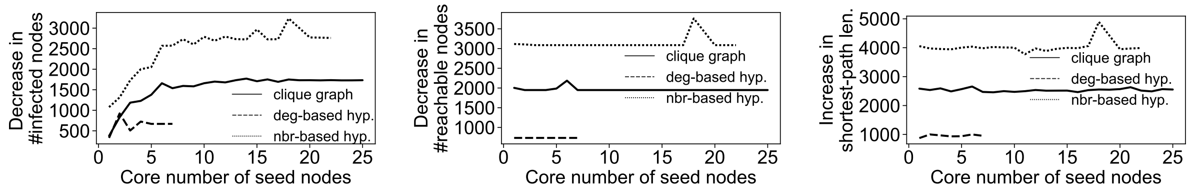
5.2 Efficiency Evaluation

Figures 7(a)-(b) show the execution times of **Peel**, **E-Peel**, optimized sequential **Local-core** algorithm **Local(OPT)**, and its parallel variant **Local(P)**. We find that **(1) E-Peel** is more efficient than **Peel** on



(a) **(b)** **(c)** **(d)**

Figure 9: (a) The number of infected nodes increases as the core-number of the seed node increases. (b) The infection time of nodes decreases for nodes with higher core-numbers indicating innermost cores are infected earlier. (c) The impact of deleting the innermost-core for disrupting diffusion. $H0$ is the input hypergraph H, $H1$ is constructed by deleting nodes in the innermost-core of $H0$, $H2$ is constructed by deleting nodes in the innermost-core of $H1$, and so on. The number of infected nodes generally decreases after each deletion of the innermost-core. (d) The reason behind such intervention to be effective is that the average number of nodes reachable from the seed node decreases due to deletion of the innermost-core (*Enron* dataset.).



(a) **(b)** **(c)**

Figure 10: Comparison between different decomposition approaches on *enron*: Different decomposition methods assign different core-numbers to the same node. Thus, not all methods have the same core-number range and the curves representing different methods have different spans along X-axis. (a) Decrease in the number of infected population due to deletion of the innermost-core from different decomposition approaches. (b) Deleting an innermost neighborhood-based core causes the most disruption in reachability among nodes. (c) Nodes becomes either unreachable (∞ is replaced by $|E|$=5 734 to compute the avg. shortest path lengths from seed nodes) or the cost of reaching them from seed nodes after deleting the innermost-core becomes higher.

majority datasets. **(2) Local(OPT)** is more efficient than **Peel** and **E-Peel** on all datasets. **(3) Local(P)** is more efficient than **Peel**, **E-Peel**, and **Local(OPT)** on all datasets.

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 7(a)-(b), **E-Peel** achieves the highest speedup on *enron* because $\alpha = 0.38$ is the smallest for this dataset. We also find that $\alpha \cong 1$ on *congress*, *bin3U*, and *contact*, thus **E-Peel** gains almost no speedup on these datasets.

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+I** incorporates optimization-I to **Local-core**, **Local+I+III** incorporates optimization-III on top of **Local+I**, **Local+I+III+IV** incorporates optimization-III on top of **Local+I+III**. Finally, **Local(OPT)** incorporates all four optimizations. Figures 7(c)-(d) show the execution times of **Local-core**, **Local+I**, **Local+I+III**, **Local+I+III+IV**, and **Local(OPT)** on all datasets. We observe that adding each optimization generally reduces the execution time. The impact of adding the same optimization on different datasets is different in general. For instance, on *enron* the speedup of **Local+I+III+IV** w.r.t **Local+I+III** is 1.5x, whereas on *congress* the speedup is 1x (no speedup). This is because on *enron*, a large number of nodes (90%) have their core-numbers equal to their respective local lower-bounds (Lemma 1). However, on congress, only 0.05% nodes have core-numbers equal to their local lower-bounds. As a result, more redundant $\mathcal{H}$ computations are saved on *enron* compared to that in *congress*.

Exp-5: Impact of parallelization. We test the parallelization performance of **Local(P)** by varying the number of threads from 1 to 64 (Figure 8). Adding more threads reduces the per-thread workload in a single iteration. As a result, individual iterations takes less time culminating in the reduction of overall execution time. For example, in Figure 8(b), **Local(P)** is **5.5x** faster on *congress* when the number of threads increases from 1 to 64. However in Figure 8(a), **Local(P)** achieves only up to **1.9x** speedup. The limited speedup is due to two reasons: 1) As shown in Exp-2, 98% nodes converge by 2 iterations in *pref3U*. In the remaining iterations, only 2% nodes are processed in parallel. This limits the parallelism because many threads remain idle. 2) *pref3U* has a skewed neighborhood-size distribution as the standard deviation (412.4 in Table 1) is the largest among all datasets. Thus, there are a few non-converged nodes with significantly large numbers of neighbors. The thread handling such a node will take longer than the rest, causing the overall execution time to increase.

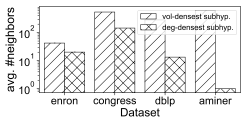
6 APPLICATIONS AND CASE STUDIES

6.1 Influence Spreading and Intervention

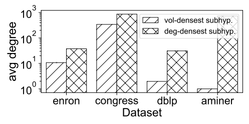
We empirically study the significance of nodes in the innermost-core derived from neighborhood-based core decomposition. We consider the *SIR* diffusion process [40]: Initially, all nodes except one—called a *seed*—are at the *susceptible* state. The seed node is initially at the *infectious* state. At each time step, each infected node infects its susceptible neighbors with probability β and then enters into *immunized* state. Once a node is immunized, it is never re-infected.

Innermost-core contains influential spreaders. We randomly select a seed node from each core-number in *enron* and run the SIR model (β=0.3). We compute the number of infected nodes after 100 time-steps. We repeat this process 1000 times and report the average number of infected nodes in Figure 9(a). We observe that inner-core nodes infect larger population compared to outer-core nodes.

Innermost-core contracts diffusion early. We select a seed node from the hypergraph uniformly at random, and run the *SIR* model for 100 time-steps. For each seed node, we record the time-step at which other nodes are infected. Repeating 1000 times with different seeds,



(a) Volume density



(b) Degree density

Figure 11: Comparison between different subhypergraphs based on average #neighbors (left) and average degree (right) per node

we report the average infection times. Figure 9(b) indicates that nodes in inner-cores are infected earlier than those in outer-cores.

Innermost-core deletion for maximum intervention in spreading. We next devise an intervention strategy to disrupt diffusion, which has significance in mitigating the spread of contagions (in epidemiology), limiting the spread of misinformation, or blocking competitive campaigns (in marketing). We *delete nodes and hyperedges in the innermost-core*, and measure the number of infected population before and after deletion. Figure 9(c) shows the effectiveness of our intervention strategy over *enron*, where $H0$ is the input hypergraph H, $H1$ is constructed by deleting nodes in the innermost-core of $H0$, $H2$ is constructed by deleting nodes in the innermost-core of $H1$, and so on. We select a seed node from the remaining hypergraph uniformly at random. We find that the number of infected population generally decreases after each deletion of the innermost-core, irrespective of the origin of seed nodes. Moreover, deleting the innermost-core also reduces the number of nodes reachable from seed nodes, making our intervention quite effective (Figure 9(d)).

Comparative analysis of intervention strategies. We compare the effectiveness of different core decomposition approaches on hypergraphs, while applying the same intervention strategy. For that, we take the difference in the number of infected population, average number of reachable nodes from the seed, and average length of shortest paths from the seed between $H0$ and $H1$, separately for all three decomposition approaches: neighborhood-based (this work), degree-based [51, 56], and clique-graph based (§ 2.2).

In Figure 10(a), we observe that the neighborhood-based intervention results in the largest decrease in infected population compared to other two core decomposition approaches, thereby showing superior effectiveness of our decomposition-based intervention. To explain why our intervention is more effective than others, Figures 10(b)-(c) show the highest decrease in the average number of reachable nodes and the highest increase in the average length of shortest paths in our decomposition-based intervention.

6.2 Densest SubHypergraph Finding

The densest subgraph can be defined as a subgraph with the maximum average node-degree among all subgraphs of a given graph [15, 22, 29], which may correspond to communities [19], filter bubbles and echo chambers [41] in social networks, and brain regions responding to stimuli [43] or diseases [65]. Following the same principal, 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 $H[S]$.

$$\rho^N[S] = \frac{\sum_{u \in S} |N_S(u)|}{|S|} \tag{8}$$

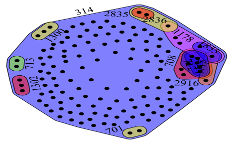


Figure 12: Volume-densest subhypergraph of *human protein complex*

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

Approximation algorithm. Inspired by Charikar [15], our approach to find the (approximate) volume-densest subhypergraph 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 (e.g., **Peel**, **E-Peel**, **Local(OPT)**, or **Local(P)**). 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 selected earlier for peeling. We finally return the subhypergraph that achieves the largest volume-density.

THEOREM 5. *Our volume-densest subhypergraph finding algorithm returns $(d_{pair}(d_{card} - 2) + 2)$-approximate densest subhypergraph if the maximum cardinality of any hyperedge is d_{card} and the maximum number of hyperedges between any pair of nodes is d_{pair} in the input hypergraph.*

The proof is given in our extended version [3]. Notice that $d_{pair} = 2$ if the hypergraph is a graph and our result gives 2-approximation guarantee for the densest subgraph discovery [15].

6.2.1 Effectiveness of volume-densest subhypergraphs. We compute volume-densest subhypergraphs from our datasets using our approximation algorithm, and in those subhypergraphs we compute the avg. number of neighbors (neighborhood-cohesion measure) and the avg. degree (degree-cohesion measure) per node. As baseline for comparison, we compute those two measures on the degree-densest subhypergraphs of the same hypergraphs extracted using [35].

In Figure 11(a), we find that the nodes in the volume-densest subhypergraph have more neighbors (on average) than that in the degree-densest subhypergraph. In Figure 11(b), we observe that the nodes in the degree-densest subhypergraph have a higher degree (on average) than that in the volume-densest subhypergraph. These observations suggest that volume-densest subhypergraph is more effective than degree-densest subhypergraph in capturing neighborhood-cohesive regions. In contrast, degree-densest subhypergraph is more effective in capturing degree-cohesive regions. Neighborhood-cohesiveness is important in many applications as follows.

6.2.2 Case Study I: Biology. We analyze a real-world hypergraph of manually-annotated human protein complexes collected from the *CORUM* database [28]. We consider each protein complex as a hyperedge consisting of proteins as nodes. There are 2611 hyperedges and 3622 nodes in the *Human protein complex* hypergraph. The volume-densest subhypergraph shown in Figure 12 has several interesting characteristics. **First,** the complexes in the subhypergraph are correlated as they participate in two fundamental biological processes: RNA metabolism and RNA localisation (e.g., some of them are listed in Table 2). **Second,** the largest hyperedge (314) is the Spliceosome complex, responsible for RNA splicing, which assists

Table 2: Functions of some complexes in the volume-densest subhypergraph of *human protein complex*

Hyper-edge id	Complex name	Function
314	Spliceosome	RNA metabolic process
708	TREX	RNA Localization
712	THO	RNA Localization
713	CBC	RNA Localization
2835	TRA2B1-SRSF9-SRSF6	RNA Metabolic process
2836	SRSF9-SRSF6	RNA Metabolic process
2916	hTREX84	RNA Metabolic process

Table 3: The top-5 highest degree nodes in volume-densest and degree-densest subhypergraphs. High-ranking executives of *enron* who make key decisions are captured in both subhypergraphs.

	Top-5 highest degree nodes	Designation	Degree in sub hyp.
Degree densest subhyp.	Kenneth Lay	CEO	202
	Greg Whalley	President	118
	Mark Koenig	Head (Investor Relations)	107
	Jeffrey McMahon	Chief Financial Officer	88
	Mark A. Frevert	Chairman and CEO (EWS)	86
Volume densest subhyp.	Kenneth Lay	CEO	122
	Greg Whalley	President	28
	Jeffery Skilling	CEO	27
	Mark A. Frevert	Chairman and CEO (EWS)	22
	Mark Koenig	Head (Investor Relations)	20

Table 4: The top-5 highest neighborhood-size nodes in volume-densest and degree-densest subhypergraphs. In the degree-densest subhypergraph, the top-5 highest neighborhood-size nodes are quite similar to those in the top-5 highest-degree nodes (high-ranking executives). However, the top-5 highest neighborhood-size nodes in the volume-densest subhypergraph are ordinary employees.

	Top-5 highest neighborhood-size nodes	Designation	#nbrs in subhyp.
Degree densest subhyp.	Kenneth Lay	CEO	165
	Greg Whalley	President	158
	Mark Koenig	Head(Investor Relations)	153
	Jeffrey McMahon	Chief Financial Officer	152
	John Sherriff	President and CEO (Enron Europe)	151
Volume densest subhyp.	Gregory Martin	**Analyst**	1948
	Dustin Collins	**Associate (Enron Global Commodities)**	1948 1948
	Andrea Richards	**Ordinary Employee**	1948
	Sladana-anna Kulic	**Ordinary Employee**	1948
	Maureen Mcvicker	**Assistant**	1948

Table 5: Subject and intent of the top-3 emails with the highest number of participants in the volume-densest subhypergraph

Email subject	Intent of the email
Updated Cougars@Enron email list	Seeking applicants for Board of directors position at Cougars@Enron (U Housten alumni group at Enron)
Associate/ analyst program	Announcing about talent-seeking program of Enron.
Analyst & associate program - e-speak invitation from Billy Lemmons	Invitation to attend an online seminer

in cell-evolution process and in the making of new and improved proteins in human body. Two subsets of Splicesome (2835 and 2836) are responsible for regulating mRNA splicing, which are known to be affected by a genetic disease called TAU-mutation causing frontotemporal dementia. **Finally,** the TREX complex (708) is responsible for transporting mRNA from the nucleus to the cytoplasm. One of its subsets, hTREX84 (2916) is found to be highly correlated with ovarian and breast cancers [32].

6.2.3 Case study II: Email communication. We extract all emails involving Kenneth Lay, who was the founder, chief executive officer, and chairman of Enron [64]. The ego-hypergraph of such a key-person, having 4718 nodes (person) and 1190 hyperedges (emails), can provide insights and difference between the volume-densest and degree-densest subhypergraphs. The degree-densest subhypergraph has 166 nodes and 202 hyperedges, whereas the volume-densest subhypergraph has 1 949 nodes and 122 hyperedges. We analyze the degree and neighborhood-sizes of the top-5 nodes in these two subhypergraphs in Tables 3 and 4. In Table 3, we find that both degree-densest and volume-densest subhypergraphs contain high-ranking key-personnel in Enron. Such personnel (nodes) participate in many emails (hyperedges) in the extracted subhypergraph. However, in Table 4, we notice that ordinary employees are communicated the most in emails (i.e., they are nodes with many neighbors), and such employees can only be extracted by analyzing the volume-densest subhypergraph. We further investigate the reason why ordinary employees have more neighbors in the volume-densest subhypergraph. We extract hyperedges (emails) where the top-5 highest neighborhood-size nodes were involved. We found that many such emails were about internal announcements, meetings/seminar invitation, and employee social gatherings (Table 5).

7 RELATED WORK

Recently, there has been a growing interest on hypergraphs data management [23, 38, 42, 54, 57, 63]. Since the focus of this paper is on core decomposition, we urge readers with a much broader interest to refer to recent surveys [10, 20, 60] for a general exposition.

Core decomposition in hypergraphs. Although core decomposition on graphs have been studied for decades [47], there are relatively few works on hypergraph core decomposition. Ramadan et al. [51] propose a peeling algorithm to find the maximal-degree-based k-core of a hypergraph. [37, 54] discuss parallel implementations of degree-based hypergraph core computation based on peeling approach. Sun et al. [56] propose a fully dynamic approximation algorithm that maintains approximate degree-based core-numbers of an unweighted hypergraph. Unlike ours, none of these works explore neighborhood-based hypergraph core decomposition, which is different from degree-based hypergraph core computation (§1), nor they consider algorithmic approaches other than peeling.

Core decomposition in graphs. The linear-time peeling algorithm for graph core decomposition was given by Batagelj and Zaveršnik [9]. Core decomposition has also been studied in disk-based [16, 39], distributed [50], parallel [18], and streaming [52] settings, and for varieties of graphs, e.g., weighted [26], directed [44], temporal [26], uncertain [13], and multi-layer [25] networks. Higher-order cores in a graph (e.g., (k, h)-core [14, 45], triangle k-core [69], h-clique-core, and pattern-core [22, 61]) and more complex cores in an attributed network (e.g., meta-path-based core [21] and (k, r)-core [68]) have been proposed. In §2.2, we reasoned that existing approaches for graph core decomposition cannot be easily adapted for neighborhood-based hypergraph core decomposition. We also depicted that the local approach [46, 50], an efficient method for graph core decomposition, produces incorrect core-numbers for neighborhood-based hypergraph core decomposition (§3.3, §5).

8 CONCLUSIONS

In this paper, we introduced the notion of neighborhood-cohesive core decomposition of hypergraphs. We proved that our decomposition has desirable properties such as **Uniqueness** and **Core-containment**. We then proposed three algorithms – **Peel**, **E-Peel**, and novel **Local-core** for hypergraph core decomposition. We also adopted four hypergraph-specific optimizations to **Local-core** and its shared-memory parallel implementation. Empirical evaluation on synthetic and real-world hypergraphs depicted that the novel **Local-core** with optimizations and parallel implementation is the most efficient among all proposed algorithms. In diffusion applications, the proposed decomposition is more effective than degree-based decomposition in intervening diffusion. In the densest subhypergraph finding application, our decomposition helps extract the volume-densest subhypergraph with an approximation guarantee. Our novel volume-densest subhypergraphs are more effective in maintaining neighborhood cohesiveness than existing degree-densest subhypergraphs. Two case studies illustrated that volume-densest subhypergraphs capture functionally significant protein-complexes in biology domains and important emails in organizational communications.

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. Neighborhood-based hypergraph core decomposition (extended version). https://github.com/toggled/vldbsubmission/nbr_core.pdf.

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

[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] Robert G. Bartle. 1976. *The elements of real analysis*. Wiley and Sons.

[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, Francesco Gullo, Andreas Kaltenbrunner, and Yana Volkovich. 2014. Core decomposition of uncertain graphs. In *The ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1316–1325.

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

[15] 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.

[16] 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.

[17] 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).

[18] Laxman Dhulipala, Guy Blelloch, and Julian Shun. 2017. Julienne: A framework for parallel graph algorithms using work-efficient bucketing. In *The ACM Symposium on Parallelism in Algorithms and Architectures*. 293–304.

[19] 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.

[20] 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.

[21] Yixiang Fang, Yixing Yang, Wenjie Zhang, Xuemin Lin, and Xin Cao. 2020. Effective and efficient community search over large heterogeneous information networks. *Proc. VLDB Endow.* 13, 6 (2020), 854–867.

[22] 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.

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

[24] 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.

[25] 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.

[26] Edoardo Galimberti, Martino Ciaperoni, Alain Barrat, Francesco Bonchi, Ciro Cattuto, and Francesco Gullo. 2021. Span-core decomposition for temporal networks: algorithms and applications. *ACM Trans. Knowl. Discov. Data* 15, 1 (2021), 2:1–2:44.

[27] 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.

[28] Madalina Giurgiu, Julian Reinhard, Barbara Brauner, Irmtraud Dunger-Kaltenbach, Gisela Fobo, Goar Frishman, Corinna Montrone, and Andreas Ruepp. 2019. CORUM: the comprehensive resource of mammalian protein complexes. *Nucleic Acids Res.* 47 (2019), D559 – D563.

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

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

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

[32] Shanchun Guo, Mingli Liu, and Andrew K Godwin. 2012. Transcriptional regulation of hTREX84 in human cancer cells. *PLoS One* (2012).

[33] 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.

[34] 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.

[35] 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.

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

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

[38] 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.

[39] 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.

[40] 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.

[41] 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.

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

[43] 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.

[44] Xuankun Liao, Qing Liu, Jiaxin Jiang, Xin Huang, Jianliang Xu, and Byron Choi. 2022. Distributed d-core decomposition over large directed graphs. *Proc. VLDB Endow.* 15, 8 (2022), 1546–1558.

[45] 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.

[46] 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.

[47] Fragkiskos D. Malliaros, Christos Giatsidis, Apostolos N. Papadopoulos, and Michalis Vazirgiannis. 2020. The core decomposition of networks: theory, algorithms and applications. *The VLDB Journal* 29, 1 (2020).

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

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

[50] 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.

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

[52] 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.

[53] 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.

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

[55] Guillaume St-Onge, Iacopo Iacopini, Vito Latora, Alain Barrat, Giovanni Petri, Antoine Allard, and Laurent Hébert-Dufresne. 2022. Influential groups for seeding and sustaining nonlinear contagion in heterogeneous hypergraphs. *Communications Physics* 5, 1 (2022), 1–16.

[56] 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.

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

[58] 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.

[59] V. Thierry. 2017. *Handbook of Mathematics*. BoD-Books on Demand.

[60] L. Torres, A. S. Blevins, D. Bassett, and T. Eliassi-Rad. 2021. The why, how, and when of representations for complex systems. *SIAM Rev.* 63, 3 (2021), 435–485.

[61] Charalampos E. Tsourakakis. 2015. The k-clique densest subgraph problem. In *The International Conference on World Wide Web, WWW*. ACM, 1122–1132.

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

[63] 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.

[64] Wikipedia contributors. 2022. Kenneth Lay — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Kenneth_Lay&oldid=1096673001. [Online; accessed 25-July-2022].

[65] Qiong Wu, Xiaoqi Huang, Adam J Culbreth, James A Waltz, L Elliot Hong, and Shuo Chen. 2021. Extracting brain disease-related connectome subgraphs by adaptive dense subgraph discovery. *Biometrics* (2021).

[66] 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.

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

[68] Fan Zhang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2017. When engagement meets similarity: efficient (k, r)-core computation on social networks. *Proc. VLDB Endow.* 10, 10 (2017), 998–1009.

[69] Yang Zhang and Srinivasan Parthasarathy. 2012. Extracting, analyzing, and visualizing triangle k-core motifs within networks. In *IEEE International Conference on Data Engineering (ICDE)*. 1049–1060.