Parallel Truss Maintenance Algorithms for Dynamic Hypergraphs

Meng Wang, Qiang-Sheng Hua*, Yefei Wang, Hai Jin, and Zhiyuan Shao

National Engineering Research Center for Big Data Technology and System/Services Computing Technology and System Lab/Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, P.R. China

Abstract. K-truss is an efficient model to detect cohesive subgraphs in ordinary graphs. Many works about trussness decomposition and maintenance on ordinary graphs have been proposed in recent years. However, few studies have focused on hypergraph trussness calculation, and the state-of-art algorithm for hypergraph trussness [7] is for static hypergraphs and is unable to distinguish certain cohesive structures. In this paper, we propose a novel truss definition on hypergraphs that considers the unique structure of hypergraphs. To recognize the structure, we present a parallel decomposition algorithm and a parallel maintenance algorithm based on the h-index. The time complexities of the decomposition and maintenance algorithms are $O(m * c_{max}^2 * h_{max})$ and $O(L * c_{max} * h_{max})$, respectively. Here m is the number of hypergraph edges, c_{max} is the maximum size of a hyperedge, h_{max} is the maximum number of hyperedges that a vertex is in, and L means the largest *Degree level* [13] of vertex pairs in the hypergraph. We also implement our algorithms on real-world hypergraphs and the results show that the maintenance algorithm can speed up at most 100x compared to the decomposition algorithm in terms of time consumption.

1 Introduction

A fundamental problem in the analysis of massive networks is to detect cohesive subgraphs in graphs, which has attracted a lot of attention in recent years. A variety of cohesive subgraphs have been proposed, such as k-clique[1], k-core[2], k-truss[3], k-peak[4], and (r, s)-nucleus[5].

In many real-world problems, more than two objects can be linked together with the same connection. The hypergraph is introduced to provide a better model to describe these polyadic relationships. In an unweighted and undirected hypergraph, a hyperedge can contain any number of vertices. Fig. 1 is a hypergraph of papers and their authors, where each vertex represents an author and each hyperedge represents a paper. A paper hyperedge contains all of its author vertices.

* Corresponding author(qshua@hust.edu.cn); This work was supported in part by National Science and Technology Major Project 2022ZD0115301.

Among the models mentioned above, k-truss achieves both high computational efficiency and cohesiveness. As an important tool to detect cohesive subgraphs, it also contributes in community searching, random walking and influence maximization [6]. It is natural and necessary to consider the k-truss on hypergraphs, because the relationships between vertices are more complex. However, the classic definition of k-truss can only work on ordinary graphs. One of the difficulties in extending k-truss to hypergraphs is that conventional triangles do not exist directly in hypergraphs. In ordinary simple graphs, an edge links exactly two vertices, and a cycle of length three is considered as a triangle. But there are no such direct links between vertices in hypergraphs.

There are few works about truss algorithms on hypergraphs. [7] presents an (α, β)-triangle on static hypergraphs. If the number of hyperedges that contain three adjacent pairs of vertices (A, B), (B, C) and (C, A) is α and the number of hyperedges that contain at least one of them is β, then these three pairs of vertices form an (α, β)-triangle[1]. However, this definition might not be appropriate. Take Fig. 1 as an example, author A co-authors 3 papers with B and C respectively in 1(a) while co-authors 5 papers with B and 1 paper with C in 1(b). They are both $(1, 5)$-triangles according to [7], but the relationship between B and C is different in the two hypergraphs. They may belong to the same institute in 1(a) while just having a one-shot cooperation in 1(b). Besides, their algorithms that recognize the (α, β)-triangle are designed for static hypergraphs and need to find the corresponding structure of their triangles in the conversion graphs, such as the clique expansion (CE) and star expansion (SE)[8]. Nonetheless, the conversion leads to considerable storage and communication costs and does not consider the scenario that different hypergraphs might be mapped to the same conversion graph. Thus, we aim to find a better way to describe the trussness on hypergraphs.

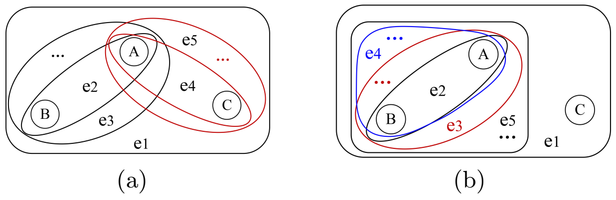

Fig. 1. Two different scenarios that cannot be distinguished in [7].

In this paper, we focus on the scenario of vertex update and maintain the trussness in hypergraphs. We propose a new truss definition on hypergraphs which can clarify different cohesiveness between different triangles, then we study

[1] In Fig. 1(a), $e_1 = \{A, B, C, \ldots\}, e_2 = \{A, B\}, e_3 = \{A, B, \ldots\}, e_4 = \{A, C\}, e_5 = \{A, C, \ldots\}$. In Fig. 1(b), $e_1 = \{A, B, C, \ldots\}$, e_2, e_3, e_4 and e_5 are different hyperedges containing A and B. For the two figures, the number of hyperedges containing the three adjacent pairs of vertices (A, B), (B, C) and (C, A) is 1 (e_1), and the number of hyperedges containing at least one of the above vertex pairs is 5 (e_1, e_2, e_3, e_4, e_5).

the change of trussness of vertex pairs. We then propose a parallel decomposition algorithm and a parallel maintenance algorithm based on the h-index. We evaluate our algorithms on real-world hypergraphs and the results show that the maintenance algorithm is much faster than the static decomposition one.

2 Problem Formulation

Consider an unweighted and undirected hypergraph $H = (V(H), E(H))$, where V is the vertex set and E is the hyperedge set. A hyperedge $e \in E(H)$ is a set of vertices in $V(H)$. The number of vertices and hyperedges in the hypergraph are denoted as n and m, respectively. For a vertex $u \in V(H)$, a hyperedge in $E(H)$ that contains u is denoted as $e_H(u)$. The set of all hyperedges that contain u is denoted as $E_H(u)$, i.e., $E_H(u) = \{e \in E(H) \mid u \in e\}$. The degree of u is denoted as $deg_H(u) = |E_H(u)|$. The neighbor set of u is denoted as $N_H(u) = \{v \in V(H) \mid \exists\, e \in E_H(u),\ v \in e\}$. The set of all hyperedges in $E(H)$ that contain both u and v is denoted as $E_H(u, v) = E_H(u) \cap E_H(v)$. Vertex u in hyperedge e_i is denoted as $e_i(u)$. The subscript of these symbols may be omitted if the context is clear.

A natural idea to extend trussness on hypergraphs is to consider which vertex pairs share the common hyperedges. If any vertex among $u, v, w \in V(H)$ is a neighbor of the other two vertices, they form a triangle in the hypergraph. Two vertices u, v that are in the same hyperedge e_i form a vertex pair $e_i(u, v)$. For a triangle formed by vertex pairs $e_i(u, v)$, $e_j(v, w)$ and $e_k(w, u)$, it is denoted as $\triangle[u^{e_i} v^{e_j} w^{e_k}]$. It should be noticed that a vertex can be contained by different hyperedges, thus there are different triangles even if the three vertices are the same. This is another difference between triangles on ordinary graphs and hypergraphs. The set of all triangles that contain $e_0(u, v)$ is denoted as $T_H[e_0(u, v)] = \{\triangle[u^{e_0} v^{e_i} w^{e_j}] \,|\, w \in V(H)\}$.

For example in Fig. 2, v_1 and v_3 share e_1, v_1 and v_5 share e_3, v_3 and v_5 share e_4. So according to the criterion above, they form a triangle. The same is true for v_1, v_2 and v_4, which share e_2. However, unlike triangles in ordinary graphs, the cohesiveness between them is different. v_1, v_3 and v_5 share three different hyperedges, while v_1, v_2 and v_4 are all in the same hyperedge e_2. As triangles in hypergraphs formed by different vertices have different cohesiveness, we consider the following two kinds of triangles in hyperedges.

Definition 1 (Inner and Outer Triangle). *Given a triangle $\triangle[u^{e_i} v^{e_j} w^{e_k}]$ in a hypergraph, if e_i, e_j, e_k are the same hyperedge, the triangle is an inner triangle. Otherwise, the triangle is an outer triangle.*

In Fig. 2, $\triangle[v_1^{e_2} v_2^{e_2} v_4^{e_2}]$ is an inner triangle and $\triangle[v_1^{e_1} v_3^{e_4} v_5^{e_3}]$ is an outer triangle. As two different triangles are defined in hypergraphs, the number of these two kinds of triangles needs to be considered separately. We consider the following two kinds of parameters to count the number of triangles.

Definition 2 (Inner and Outer Support of Vertex Pairs). *Given a vertex pair $e_0(u, v)$ in H, its inner support is defined as the number of inner triangles* *that contain $e_0(u,v)$, denoted as $sup_{in}^{H}[e_0(u,v)]$. Its outer support is defined as the number of outer triangles that contain $e_0(u,v)$, denoted as $sup_{out}^{H}[e_0(u,v)]$.*

Definition 3 (Inner and Outer Support of Vertices). *Given a vertex u in e_0, its inner support $sup_{in}^{H}[e_0(u)]$ is equal to the maximum inner support of the vertex pairs in e_0 that contain it, i.e., $sup_{in}^{H}[e_0(u)] = max\{sup_{in}^{H}[e_0(u,v)]\}, v \neq u$. Similarly, its outer support $sup_{out}^{H}[e_0(u)] = max\{sup_{out}^{H}[e_0(u,v)]\}, v \neq u$.*

The inner support is for triangles that are entirely located in a hyperedge, like $\triangle[v_1^{e_2} v_2^{e_2} v_3^{e_2}]$ in Fig. 2. This case does not exist in an ordinary graph. While the outer support is for triangles like $\triangle[v_1^{e_1} v_3^{e_4} v_5^{e_3}]$ in Fig. 2. In this situation, the triangle is formed by vertex pairs that come from three different hyperedges. Having defined how to count the triangles, we define our new truss definition on hypergraphs.

Definition 4 ((k_{in}, k_{out})-Truss). *A (k_{in}, k_{out})-truss of a hypergraph H is the largest subgraph S where each vertex pair $e_0(u,v)$ satisfies*
(1) $sup_{in}^{S}[e_0(u,v)] \geq k_{in}$, and
(2) $sup_{out}^{S}[e_0(u)] \geq k_{out}$ and $sup_{out}^{S}[e_0(v)] \geq k_{out}$.

Definition 5 (Trussness of Vertex Pairs). *Given a vertex pair $e_0(u,v)$ in H, if it is in a (k, l)-truss but neither in a ($k+1$, l)-truss nor in a (k, $l+1$)-truss, then we denote its inner trussness (when $k_{out} = l$) as $\tau_{in}^{out=l}[e_0(u,v)] = k$, and its outer trussness (when $k_{in} = k$) as $\tau_{out}^{in=k}[e_0(u,v)] = l$.*

As noted above, the inner support and outer support of a vertex pair are different in cohesiveness. In our method, we count them separately and introduce k_{in} and k_{out} as parameters to evaluate their contributions[1]. Note that we relax the restriction on the outer trussness in Definition 4, which is different from ordinary graphs. It contains vertex pairs whose outer supports are no less than k_{out} and those vertex pairs whose outer supports are less than k_{out} but the outer supports of their both vertices are no less than k_{out}. Because if we only consider the outer support of the vertex pair itself, some cohesive structures may be missed. For example in Fig. 3, $\triangle[v_1^{e_1} v_2^{e_3} v_5^{e_2}]$ and $\triangle[v_3^{e_1} v_4^{e_5} v_6^{e_4}]$ both fit the restriction of having at least 0 inner triangle and 1 outer triangle, but Fig. 3 does not fit because $e_1(v_1, v_3)$ does not have any outer triangles. Our definition 4 guarantees that the largest subgraph can be found. Lemma 1 clarifies this issue and the proof is in Appendix A.

Lemma 1. *The union operation is closed on the truss based on Definition 4. Thus the largest (k_{in}, k_{out})-truss of a hypergraph is unique.*

Take Fig. 3 as an example, the inner supports of vertex pairs in e_1 are all 2, and the inner supports of vertex pairs in other hyperedges are all 0. Thus, hyperedge e_1 and vertices in it form a $(2,0)$-truss. The outer supports of $e_0(v_1, v_3)$,

[1] The definition of k-truss in ordinary graphs requires that $sup^{G}(e) \geq (k-2)$ for each edge, which makes sure that a k-clique is also a k-truss. Without loss of generality, the original k-truss is a (0, $k-2$)-truss in our definition.

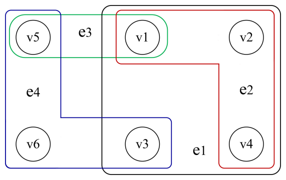

Fig. 2. Different triangles in a hypergraph

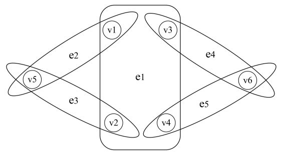

Fig. 3. Missed Cohesive Structure

Table 1. Summary of Notations

Notations	Description
$V(H), E(H)$	the set of vertices/hyperedges in H
n, m	the number of vertices/hyperedges in the hypergraph
$deg_H(u)$	the degree of vertex u in H
$\|e\|$	the number of vertices in hyperedge e
$E_H(u)$	the set of hyperedges that contain u
$N_H(u)$	the set of neighbors of vertex u in H
$e_0(u)$	the vertex u in hyperedge e_0
$e_0(u, v)$	the vertex pair made up with vertex u and v from e_0
$E_H(u, v)$	the set of all hyperedges in $E(H)$ that contain both u and v
$T_H[e_0(u, v)]$	the set of all triangles that contain $e_0(u, v)$ in H
$sup_{in}^{H}[e_0(u, v)]$	the vertex pair $e_0(u, v)$'s inner support in H
$sup_{out}^{H}[e_0(u, v)]$	the vertex pair $e_0(u, v)$'s outer support in H
$\tau_{in}^{out=l}[e_0(u, v)]$	the vertex pair $e_0(u, v)$'s inner trussness when k_{out} is l
$\tau_{out}^{in=k}[e_0(u, v)]$	the vertex pair $e_0(u, v)$'s outer trussness when k_{in} is k
$\triangle[u^{e_i} v^{e_j} w^{e_k}]$	the triangle formed by vertex pairs $e_i(u, v)$, $e_j(v, w)$ and $e_k(w, u)$
$lv(\triangle[u^{e_a} v^{e_b} w^{e_c}])$	the level of $\triangle[u^{e_a} v^{e_b} w^{e_c}]$

$e_0(v_1, v_4)$, $e_0(v_2, v_3)$, $e_0(v_2, v_4)$ are 0, but all outer supports of their vertices are 1, and the outer supports of other vertex pairs are 1. Thus, Fig. 3 is a $(0, 1)$-truss.

Note that a truss has two attributions k_{in} and k_{out}, thus the combination of inner trussness and outer trussness of a vertex pair is non-unique. If there are no requirements on inner trussness, each vertex pair discovers the $(0, \tau_{out}^{in=0})$-truss that it is in. Then if the restriction on inner trussness raises, the outer trussness of the vertex pairs may decrease. It is unnecessary to maintain every trussness combination for vertex pairs, so we set k_{in} and k_{out} as fixed values. Besides, it is time-consuming to calculate the trussness of each vertex pair in sequential when the magnitude of graphs comes large, so we tackle the problem in parallel.

Problem Definition We maintain a (k_{in}, k_{out})-truss where the two parameters are determined and recognize the vertices and hyperedges that are in the (k_{in}, k_{out})-truss of the updated hypergraphs while avoiding computing the whole hypergraph from scratch.

Performance Measure We use the **work-depth model** in [9]. The **work** of a parallel algorithm is the time to perform the entire computation on one processor and the **depth** is the execution time of the longest path of sequential dependencies in the algorithm.

The major notations and their descriptions are summarized in Table 1.

3 Theoretical Analyses

When it comes to a dynamic scenario, the following issues need to be considered: determining whose trussness will change and determining how much their trussness will change. We solve these issues in this section. The proofs of lemmas in this section are all presented in Appendix A. When we discuss the inner trussness or the outer trussness in the following subsections, we regard the requirement on the other parameter as a fixed value, such as 0, and omit the trussness superscript.

3.1 Inner Trussness Change

The change of inner or outer support of our (k_{in}, k_{out})-truss needs to be considered separately. The change of inner trussness is relatively simple and can be determined by Lemmas 2 and 3.

Lemma 2. *After a vertex u_0 is inserted into a hyperedge in $H = (V, E)$, the inner trussness of each vertex pair in H is increased by at most 1.*

Lemma 3. *After a vertex u_0 is deleted from a hyperedge in $H = (V, E)$, the inner trussness of each vertex pair in H is decreased by at most 1.*

3.2 Outer Trussness Change

However, when it comes to the outer trussness, the problem becomes more complex. Firstly, each vertex pair must be computed separately. Two pairs that have the same two vertices may be different because the pairs can come from different hyperedges. For example in Fig. 2, the vertex pair (u_1, u_2) can come from either e_1 or e_2. Secondly, the trussness is the property of vertex pairs, but we cannot insert or delete a vertex pair individually in hypergraphs. The dynamic hypergraphs are maintained in the view of vertex updates. So during each vertex insertion/deletion, vertex pairs linked to that vertex are inserted/deleted at the same time. Thirdly, the support of vertex pairs in the hypergraphs changes more than one after the insertion/deletion of a particular vertex, because a vertex pair can form more than one outer triangle. For example in Fig. 2, if v_4 is deleted from hyperedge e_2, the outer triangles of $e_1(v_1, v_2)$: $\triangle[v_1^{e_1} v_2^{e_2} v_4^{e_2}]$, $\triangle[v_1^{e_1} v_2^{e_2} v_4^{e_1}]$ and $\triangle[v_1^{e_1} v_2^{e_1} v_4^{e_2}]$ disappear. The theorems in ordinary graphs, such as methods by Huang et al.[10] and Luo et al.[6, 11], do not apply here for they are all based on the fact that the support of any edge changes by at most 1 with an edge/vertex insertion/deletion. We present the following Lemmas 4 and 5 to determine the change.

Lemma 4. *After a vertex u_0 is inserted into a hyperedge e_0 in $H = (V, E)$, the outer trussness of each vertex pair in H is increased by at most Δs_{max}, where Δs_{max} is the maximum of the newly formed outer triangles of the neighbor vertex pairs of the inserted vertex pairs $e_0(u_0, v_i), v_i \in e_0$.*

Lemma 5. *After a vertex u_0 is deleted from a hyperedge e_0 in $H = (V, E)$, the outer trussness of each vertex pair in H is decreased by at most Δs_{max}, where Δs_{max} is the maximum of the newly disappeared outer triangles of the neighbor vertex pairs of the deleted vertex pairs $e_0(u_0, v_i), v_i \in e_0, v_i \neq u_0$.*

3.3 Affected Vertex Pairs

In this subsection, we propose several lemmas to help determine the affected vertex pairs whose trussness may change and the range of changes. According to Definition 2 and Lemmas 2 and 3, it is clear that the update of hypergraphs will only affect the inner trussness of vertex pairs which are in the same hyperedges with the inserted/deleted vertices. Thus, we get Lemma 6.

Lemma 6. *After a vertex u is inserted into or deleted from a hyperedge e_0 in $H = (V, E)$,*
(1) the inner trussness of each vertex pair in e_0 is increased or decreased by at most 1;
(2) the inner trussness of each vertex pair outside e_0 remains unchanged.

The maintenance of outer trussness is based on h-index [12]. The h-index of a set is defined as the largest integer h where there are at least h elements in the set that are no less than h. For example, the h-index of a set $\{1, 1, 2, 2, 3\}$ is 2, since there are 3 elements that are no less than 2 and only 1 element that is no less than 3. If elements in the set are the supports of neighboring vertex pairs of a given vertex pair, the h-index converges to its trussness after iterations. An important issue in convergence is to set the initial h-indices as low as possible for vertex pairs whose outer trussness may change. Definition 6 explains the k-triangle and Lemma 7 shows the range of outer trussness change for different vertex pairs.

Definition 6 (K-Triangle and Triangle Level). *For a triangle $\triangle[u^{e_a} v^{e_b} w^{e_c}]$, it is a k-triangle if the minimum outer trussness of the vertex pairs $e_a(u, v), e_b(v, w), e_c(u, w)$ is k. And k is the triangle level, denoted as $lv(\triangle[u^{e_a} v^{e_b} w^{e_c}]) = k$.*

Lemma 7. *Suppose a vertex u_0 is inserted into a hyperedge e_0 in H. Δs_{max} is the maximum outer support change of the neighboring vertex pairs of the inserted vertex pairs and $0 \leq \Delta s \leq \Delta s_{max}$. $lv_{min}[e(u, v)]$ is the lowest level of the outer triangle that contains $e(u, v)$. $minsup^H_{out}[e(u), e(v))]$ is the minimum of $sup^H_{out}[e(u)]$ and $sup^H_{out}[e(v)]$. For the new outer trussness $\tau'_{out}[e(u, v)]$ of a vertex pair $e(u, v)$ in H, we have:*

$$\tau'_{out}[e(u,v)] = \begin{cases} \tau_{out}[e(u,v)], & minsup^H_{out}[e(u), e(v))] \leq \tau_{out}[e(u,v)] - \Delta s_{max} \\ \tau_{out}[e(u,v)] + \Delta s, & others \\ \tau_{out}[e(u,v)], & lv_{min}[e(u,v)] > \tau_{out}[e(u,v)] + \Delta s_{max} \end{cases}$$

4 Parallel Algorithms

In this section, we will give the details of our parallel trussness algorithms for hypergraphs, i.e., the decomposition algorithm for static hypergraphs and the maintenance algorithm for dynamic hypergraphs. We also analyze the performance of the algorithm.

The following notations are used in the time complexity analysis of our algorithms: n means the number of vertices in the hypergraph H. m means the number of hyperedges in the hypergraph. $c_{max} = max_{e \in E(H)}|e|$ means the maximum number of vertices in a hyperedge in H. $h_{max} = max_{u \in V(H)}|E_H(u)|$ means the maximum number of hyperedges that a vertex is in. s_{max} means the maximum outer support of the vertex pairs that make new triangles with the updated vertex pairs. $t_{max} = max_{u,v \in V(H)}|T_H[e(u,v)]|$ means the maximum number of triangles that a vertex pair can form in H. $|\Delta_H|$ means the size of the updated vertex pair set. L means the largest *degree level* [13] of vertex pairs in the hypergraph.

4.1 Parallel Decomposition Algorithm

Algorithm 1 provides a natural idea to compute the inner and outer trussness for each vertex pair according to Definition 4. Vertex pairs that do not fit the conditions are peeled out during each step. Note that we maintain the precise outer trussness for each vertex pair in the condition that the inner trussness is k_{in}. This is preprocessing for the maintenance. An example of the decomposition algorithm in Fig.2 is presented in Appendix B.

At the beginning of the algorithm, the inner support and outer support of each vertex pair are calculated. Then we peel the vertex pairs that do not satisfy the requirements from the hypergraph repeatedly. Specifically, the purpose of lines 5-18 is to obtain (k_{in}, k)-truss. In lines 7 to 11, vertex pairs whose inner supports are less than k_{in} are deleted. Note that if a vertex pair is deleted when $k = 0$, it will not get an inner trussness or an outer trussness. Because the vertex pair does not meet the condition of $(k_{in}, 0)$-truss and is not worthy of further discussion. In lines 12 to 17, the vertex pair will be deleted if the outer supports of this vertex pair and its two vertices are all less than k. Because we can only delete a vertex from a hyperedge rather than delete an individual vertex pair in hypergraphs, the vertex with a lower outer support is chosen according to Definition 4. When outer supports of all vertex pairs in the hypergraph are higher than k, the calculation of (k_{in}, k)-truss is finished and we move to the $(k_{in}, k+1)$-truss. The peeling continues until all vertex pairs get their trussness. As the output of the algorithm, vertex pairs whose inner trussness $= k_{in}$ and outer trussness $\geq k_{out}$ are obtained.

Theorem 1. *Alg.1 outputs the* (k_{in}, k_{out})*-truss correctly. Its depth is* $O(m * c_{max}^2 * h_{max})$ *and its work is* $O(m * n * c_{max}^3 * h_{max}^2)$.

Proof. The decomposition algorithm calculates the outer trussness according to Definition 4. During each step, we remove the vertices that do not meet the

Algorithm 1: Trussness Decomposition

```
Input  : Hypergraph $H(V,E)$, truss parameters $k_{in}$ and $k_{out}$
Output : vertex pairs that are in $(k_{in}, k_{out})$-truss
1  for each $e_i(u,v)$ in $H$ in parallel do
2      Compute inner support and outer support of each $e_i(u,v)$;
3  $k \leftarrow 0$;
4  while $H \neq \emptyset$ do
5      while $\exists$ $sup^H_{out}[e_i(u,v)] < k$ or $sup^H_{in}[e_i(u,v)] < k_{in}$ do
6          for each $e_i$ that contains $sup^H_{in}[e_i(u,v)] < k_{in}$ in parallel do
7              Remove all vertices from $e_i$;
8              for each $e_i(u,v) \in e_i$ do
9                  if $k > 0$ then
10                     $\tau_{in}[e_i(u,v)] \leftarrow k_{in}$, $\tau_{out}[e_i(u,v)] \leftarrow k-1$;
11         for each $sup^H_{out}[e_i(u,v)] \leq k$ in parallel do
12             if $sup^H_{out}[e_i(u)] < k$ and $sup^H_{out}[e_i(v)] < k$ then
13                 W.l.o.g, assume that $sup^H_{out}[e_i(u)] \leq sup^H_{out}[e_i(v)]$;
14                 Remove the vertex $u$ from $e_i$;
15                 for $u_i \in e_i$ do
16                     $\tau_{in}[e_i(u,v)] \leftarrow k_{in}$, $\tau_{out}[e_i(u,u_i] \leftarrow k-1$;
17         for each neighbouring vertex pair $e(u,w)$ of each $e_i(u,v)$ in parallel do
18             Compute inner support and outer support of $e(u,w)$;
19     $k = k + 1$;
20 return vertex pairs whose inner trussness = $k_{in}$ and outer trussness $\geq k_{out}$ ;
```

necessary condition to be in (k_{in}, k)-truss. Once there are no vertex pairs that fail to fulfill the condition, we go to a higher level until all vertex pairs receive their outer trussness (except those vertex pairs that are deleted in the first step).

Considering the worst case, each vertex pair meets the condition of inner trussness and will only be deleted in the processing of outer trussness comparison. If only one vertex is deleted in a step, the total number of steps is $O(m * c_{max})$. During each step, it takes $O(1)$ to remove vertices from hyperedges and note their inner and outer trussness. The time to recompute the inner supports and outer supports of all affected vertex pairs is $O(c_{max} * h_{max})$. Thus the total time complexity is $O(m{*}c^2_{max}{*}h_{max})$. As for the work of Alg.1, in the total $O(m{*}c_{max})$ steps, there are $O(n{*}c_{max}{*}h_{max})$ vertex pairs and the calculation of trussness for each vertex pair takes $O(c_{max}{*}h_{max})$. Thus the work is $O(m{*}n{*}c^3_{max}{*}h^2_{max})$. □

4.2 Parallel Maintenance Algorithm

In this subsection, we propose a parallel maintenance algorithm to update the vertex pair trussness in hypergraphs when a set of vertices are inserted into or deleted from the hypergraph. Differing from the decomposition algorithm, the maintenance algorithm updates the trussness of vertex pairs in parallel according to the original values and can deal with batch processing. Because the maintenance algorithm does not need to recalculate the trussness for all vertex pairs in the hypergraphs, it can make a further improvement in time complexity. Due to the limited space, we take the insertion algorithm as an example. The deletion algorithm is presented in Appendix C.

According to Lemma 7, we get the range of new outer trussness of vertex pairs after the update. For those vertex pairs whose outer trussness may increase after the update, we set their initial h-indices to the upper change bounds Δs_{max}. In lines 2 to 4, Alg.2 computes inner and outer supports of $e_0(u_0, v_i)$ for each vertex pair in the new edge set. In line 5, the maximum outer support s_{max} of the vertex pairs that make new triangles with $E(u_0)$ is gained. It is an important parameter to distinguish different affected vertex pairs. Then in lines 6 to 13, the algorithm sets the initial h-index for each vertex pair. The insertion increases the inner trussness of vertex pairs in E_0, and their outer trussness also needs recomputing. These vertex pairs are removed in the processing of the inner trussness check during the decomposition, and do not get a specific outer trussness. Thus, their initial h-indices are set to their outer trussness supports. For other vertex pairs, we set their initial h-indices to the upper bounds of their new values according to Lemma 7. Finally, the algorithm calls the h-index-based algorithm $UpdateTrussness$ to update their trussness. The `UpdateTrussness` algorithm is presented in Appendix C.

Algorithm 2: Trussness Insertion

```
   Input  : Hypergraph $H(V,E)$, truss parameters $k_{in}$ and $k_{out}$, new vertex set
            $U_\Delta$, update hyperedge set $E_\Delta$, new vertex pair set $\Delta_H$
   Output : Update inner and outer trussness for each vertex pair
1  Insert $U_\Delta$ into $H$;
2  for vertex pair $e(u,v)$ in $\Delta_H$ in parallel do
3      calculate $sup^H_{in}[e(u,v)]$ and $sup^H_{out}[e(u,v)]$;
4      $h_0[e(u,v)] = sup^H_{out}[e(u,v)]$;
5  Calculate the maximum outer support change $\Delta s_{max}$ of the neighbor vertex
     pairs of the inserted vertex pairs $\Delta_H$;
6  if $e(u,v) \in E_\Delta$ and $sup^H_{in}[e(u,v)] >= k_{in}$ then
7      $h_0[e(u,v)] = sup^H_{out}[e(u,v)]$;
8  for vertex pair $e(u,v)$ in $(k_{in}, 0)$-truss in parallel do
9      if $minsup^H_{out}[e(u), e(v))] \leq \tau_{out}[e(u,v)] - Deltas_{max}$ or
         $lv_{min}[e(u,v)] > \tau_{out}[e(u,v)] + \Delta s_{max}$ then
10         $h_0[e(u,v)] = \tau_{out}[e(u,v)]$;
11     else
12         $h_0[e(u,v)] = \tau_{out}[e(u,v)] + \Delta s_{max}$;
13 UpdateTrussness($H(V,E)$, $H_0[e(u,v)]$);
14 Return each vertex pair in $(k_{in}, k_{out})$-truss;
```

Theorem 2. *Alg.2 updates the (k_{in}, k_{out})-truss correctly. Its depth is $O(L * c_{max} * h_{max})$ and its work is $O(|\Delta_H| * c_{max}^2 * h_{max}^2 + n * c_{max} * h_{max} * t_{max})$.*

Proof. Firstly, each vertex pair in U_Δ needs to update its support, and each pair will take $O(c_{max} * h_{max})$ because a vertex needs to traverse all the hyperedges it is in and all vertices in these hyperedges. Then the calculation of s_{max} needs to traverse through the vertex pairs that make new triangles with Δ_H and get their supports. So these two preprocessing steps will take $O(c_{max} * h_{max})$. Then in lines 6-12, the initial step needs $O(1)$ to set the h_0 for each vertex pair in parallel.In the `UpdateTrussness` procedure, each iteration has a time complexity of $O(c_{max} * h_{max})$ because each vertex pair must update the h-values of its adjacent vertex pairs during each h-index computation. Assuming that the parallel decomposition algorithm for hypergraphs trussness based on the h-index is considered, the number of iterations is $O(L)$. Given that our maintenance algorithm provides a lower initial value, the number of iterations in the `UpdateTrussness` algorithm is also $O(L)$. The explanation of *degree level* and proof of the fact above is presented in Appendix C. Therefore, the time complexity of the `UpdateTrussness` procedure is $O(L * c_{max} * h_{max})$. Above all, the time complexity of Alg. 2 is bounded by $O(c_{max} * h_{max}) + O(1) + O(L * c_{max} * h_{max})$.

As for the work of Alg. 2, the two preparation steps will take $O(|\Delta_H| * (c_{max} * h_{max})^2)$. The initial step takes $O(n * c_{max} * h_{max})$ to initialize h_0 for each vertex pair. The h-index of each vertex pair converges from their initial value to the outer trussness. This takes $O(n * c_{max} * h_{max} * s_{max}) = O(n * c_{max} * h_{max} * t_{max})$. Thus the work is $O(|\Delta_H| * (c_{max} * h_{max})^2) + O(n * c_{max} * h_{max}) + O(n * c_{max} * h_{max} * t_{max}) = O(|\Delta_H| * c_{max}^2 * h_{max}^2 + n * c_{max} * h_{max} * t_{max})$. □

5 Evaluation

We perform experiments on real-world hypergraphs to evaluate the stability, scalability, parallelism, and generality of our algorithms. Owing to space limitations, we only present the stability evaluation, and other experimental results are presented in Appendix D.

Table 2 includes datasets from the real world that can be accessed through the KONECT project[2]. BC and PD are static hypergraphs and VI is a temporal hypergraph. In the table, *accu.v* refers to the sum of the number of vertices contained by all hyperedges in the hypergraph and the *sum v.p.* refers to the total number of vertex pairs in the hypergraph. We evaluate our algorithms under different sizes of updated sets on these hypergraphs. For each algorithm, we randomly choose 10^i hyperedges of the original hypergraph and then choose a random vertex in these hyperedges to update, where $i = 0, 1, 2, 3$. The average cost time of each hyperedge is the total time divided by the size of the updated set. Tab. 3 shows the results of the insertion of the static algorithm and the maintenance algorithm.

[2] http://www.konect.cc/

Table 2. Attributes of Datasets

Dataset	$\|V\|$	$\|E\|$	**accu. v**	**sum v.p.**
BookCrossing(BC)	105K	341K	1.15M	27.7M
Producers(PD)	48.8K	139K	207K	0.14M
vi.sualize.us(VI)	17.1K	495K	2.30M	3.86M

As for the static algorithm, the average update time decreases exponentially as the size of the update set increases. This is because the static algorithm treats the hypergraphs after each update as a new one and recomputes the whole hypergraph. The total time does not change apparently when the magnitude of the update becomes larger. This is because the updated set with a limited size does not influence the structure of the whole hypergraph. As for the maintenance algorithm, the average update time decreases as the magnitude of the update set increases in general. Because the number of the h-indices that can converge at the same round will increase. However, the average time may sometimes increase as the update set becomes larger because the increasing update set size will also influence the initial value of the h-index, and the vertex pairs may need more rounds to converge. Compared with the static algorithm, the maintenance algorithm gets a speedup ratio of at most 100x.

Table 3. The average insertion time in milliseconds spent on each edge

Size of Updated Set	10^0	10^1	10^2	10^3
static alg. on BC	100.53	11.065	1.4436	0.14172
static alg. on VI	8363.2	884.99	90.946	8.7208
static alg. on PD	4580.4	473.33	46.883	4.6862
maintenance alg. on BC	0.90712	0.09427	0.024582	0.014045
maintenance alg. on VI	94.993	8.8412	7.9276	7.6037
maintenance alg. on PD	35.993	3.9247	2.1783	0.47645

6 Conclusion

In this paper, we propose a novel definition for trussness which can describe the unique structures of hypergraphs, and study the patterns of trussness change. Compared with [7], our work supports dynamic hypergraphs and thoroughly considers the complex situations of dynamic hypergraphs. To identify such structures in hypergraphs, we develop and implement a peeling-based parallel decomposition algorithm and a parallel maintenance algorithm based on h-index. The experimental results on real-world hypergraphs show that the maintenance algorithm can achieve up to 100 $\times$ speedup in time consumption compared to the decomposition algorithm. In the future research, we will focus on exploring a better way to improve the iteration of trussness updates and optimizing these algorithms to reduce their time complexities.

References

1. Duan, D., Li, Y., Li, R., Lu, Z.: Incremental k-clique clustering in dynamic social networks. Artificial Intelligence Review **38**, 129–147 (2012)
2. Batagelj, V., Zaversnik, M.: An O(m) algorithm for cores decomposition of networks. arXiv preprint cs/0310049 (2003)
3. Cohen, J.: Trusses: Cohesive subgraphs for social network analysis. National security agency technical report **16**(3.1), 1–29 (2008)
4. Govindan, P., Wang, C., Xu, C., Duan, H., Soundarajan, S.: The k-peak decomposition: Mapping the global structure of graphs. In: Proceedings of the 26th International Conference on World Wide Web (WWW 2017), pp. 1441-–1450 (2017)
5. Sariyuce, A.E., Seshadhri, C., Pinar, A., Catalyurek, U.V.: Finding the hierarchy of dense subgraphs using nucleus decompositions. In: Proceedings of the 24th International Conference on World Wide Web (WWW 2015), pp. 927-–937 (2015)
6. Luo, Q., Yu, D., Cheng, X., Cai, Z., Yu, J., Lv, W.: Batch processing for truss maintenance in large dynamic graphs. IEEE Transactions on Computational Social Systems **7**(6), 1435–1446 (2020)
7. Wang, X., Chen, Y., Zhang, Z., Qiao, P., Wang, G.: Efficient truss computation for large hypergraphs. In: Chbeir, R., Huang, H., Silvestri, F., Manolopoulos, Y., Zhang, Y. (eds.) WISE 2022. pp. 290—305. Springer International Publishing, Cham (2022)
8. Gu, Y., Yu, K., Song, Z., Qi, J., Wang, Z., Yu, G., Zhang, R.: Distributed hypergraph processing using intersection graphs. IEEE Transactions on Knowledge and Data Engineering **34**(7), 3182–3195 (2020)
9. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms. MIT press (2022)
10. Huang, X., Cheng, H., Qin, L., Tian, W., Yu, J.X.: Querying k-truss community in large and dynamic graphs. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD 2014), pp. 1311—1322 (2014)
11. Luo, Q., Yu, D., Cheng, X., Sheng, H., Lyu, W.: Exploring truss maintenance in fully dynamic graphs: A mixed structure-based approach. IEEE Transactions on Computers **72**(3), 707–718 (2022)
12. Lü, L., Zhou, T., Zhang, Q.M., Stanley, H.E.: The h-index of a network node and its relation to degree and coreness. Nature communications **7**(1), 10168 (2016)
13. Sariyuce, A.E., Seshadhri, C., Pinar, A.: Local algorithms for hierarchical dense subgraph discovery. Proceedings of the VLDB Endowment **12**(1), 43-56 (2018)

A The Proofs of Lemmas

A.1 The Proof of Lemma 1

Lemma 1. *The union operation is closed on the truss based on Definition 4. Thus the largest (k_{in}, k_{out})-truss of a hypergraph is unique.*

Proof. Assume that there are two different (k_{in}, k_{out})-truss S_1, S_2. The sub-hypergraph $S_0 = S_1 \cup S_2$ contains two types of vertex pairs : (1) vertex pairs come from S_1 or S_2; (2) newly generated vertex pairs in S_0. For the first kind of vertex pairs, they meet the requirements of inner trussness and outer trussness because they originally belong to (k_{in}, k_{out})-truss S_1 or S_2. For the second kind of vertex pairs, they appear if and only if a hyperedge e in H is in both S_1 and S_2 and the vertices that it contains in S_1 and S_2 are different. A new vertex pair $e(u,v)$ can appear in S_0 but we cannot guarantee its outer support only by the original condition in ordinary graphs, where $u \in V(S_1)$, $u \notin V(S_2)$, $v \notin V(S_1)$, $v \in V(S_2)$. The inner support of new vertex pair $e(u,v)$ is no less than k_{in} for there are at least k_{in} vertices in hyperedge e of S_1 or S_2 and each of them will form an inner triangle with $e(u,v)$. According to the second condition of Definition 4, the vertex pair is in (k_{in}, k_{out})-truss if both of the outer support of its two vertices are no less than k_{out}. The support of $e(u)$ and $e(v)$ are no less than k_{out} for they are in (k_{in}, k_{out})-truss S_1 or S_2 respectively, thus $e(u,v)$ also meets the requirements of (k_{in}, k_{out})-truss. Above all, S_0 is a (k_{in}, k_{out})-truss according to the definition. Thus the union operation is closed on the truss based on Definition 4.

If we can find a new (k_{in}, k_{out})-truss S_i which is not a sub-hypergraph of S_0, we can union them to form a larger (k_{in}, k_{out})-truss. Repeat this process then the unique largest (k_{in}, k_{out})-truss of a hypergraph can be found. □

A.2 The Proof of Lemma 2

Lemma 2. *After a vertex u_0 is inserted into a hyperedge in $H = (V, E)$, the inner trussness of each vertex pair in H increases by at most 1.*

Proof. A vertex u_0 is inserted into e_0 can be regarded as a set of vertex pairs $E_0 = \{(u_0, v_i) \mid v_i \in e_0\}$ are inserted. The inner support of each of the existing vertex pairs in e_0 increases by 1 and the inner support of vertex pairs in other hyperedges does not change obviously. Assume that there is a vertex pair $e(a,b)$ whose inner trussness increases by more than 1, such as $\tau_{in}[e(a,b)] = k$ in H and $\tau_{in}^{'}[e(a,b)] = k + x$ in the updated hypergraph $H^{'} = H \cup E_0$, where $x > 1$. Denote the $(k + x, 0)$-truss in $H^{'}$ that contains $e(a,b)$ as $S^{'}$. Because $S^{'}$ is a $(k + x, 0)$-truss, $sup_{in}^{S^{'}}[e(u,v)] \geq k + x$ for each vertex pair $e(u,v)$ in it. Then we remove those newly inserted edges from $S^{'}$ and consider $S = S^{'} \backslash (S^{'} \cap E_0)$. After $S^{'} \cap E_0$ is removed, the inner support of each vertex pair in S is decreased by at most 1 according to the definition of E_0. $sup_{in}^{S}[e(u,v)] \geq k + x - 1$ is held for these vertex pairs, making S is a $(k + x - 1, 0)$-truss. However, since $\tau_{in}[e(a,b)] = k < k + x - 1$, which leads to a contradiction. □

The proof of Lemma 3 is part of the proof of Lemma 2 and is omitted here to avoid duplication.

A.3 The Proof of Lemma 4

Lemma 4. *After a vertex u_0 is inserted into a hyperedge e_0 in $H = (V, E)$, the outer trussness of each vertex pair in H increases by at most Δs_{max}, where Δs_{max} is the maximum outer support of the vertex pairs who make new triangles with the inserted vertex pairs $e_0(u_0, v_i), v_i \in e_0$.*

Proof. A vertex u_0 is inserted into e_0 can be regarded as a set of vertex pairs $E_0 = \{(u_0, v_i) \mid v_i \in e_0\}$ being inserted. Assume that there is a vertex pair $e(a, b)$ whose outer trussness increases by more than Δs_{max}, such as $\tau_{out}[e(a, b)] = k$ in H and $\tau^{'}_{out}[e(a, b)] = k + x$ in the updated hypergraph $H^{'} = H \cup E_0$, where $x > \Delta s_{max}$. Denote the $(k+x, 0)$-truss in $H^{'}$ that contains $e(a, b)$ as $S^{'}$. Because $S^{'}$ is a $(k+x, 0)$-truss, each vertex pair $e(u, v)$ satisfies that $sup^{S^{'}}_{out}[e(u, v)] \geq k+x$ or $sup^{S^{'}}_{out}[e(u)]$ and $sup^{S^{'}}_{out}[e(v)]$ are both $\geq k + x$. Then we remove those newly inserted edges from $S^{'}$ and consider $S = S^{'} \backslash (S^{'} \cap E_0)$. After $S^{'} \cap E_0$ is removed, the outer support of each vertex pair in S is decreased by at most s_{max} according to the definition of E_0. The outer support of each vertex in S is also decreased by at most Δs_{max} for it is equal to the maximum outer support of the vertex pairs in a specific hyperedge that contained it. Thus $sup^{S}_{out}[e(u, v)] \geq k + x - \Delta s_{max}$ is held for these vertex pairs, which implies to that S is a $(k+x-s_{max}, 0)$-truss. However, $\tau_{out}[e(a, b)] = k < k + x - \Delta s_{max}$, which leads to a contradiction. □

The proof of Lemma 5 is similar to the proof of Lemma 4 and is omitted here.

A.4 The Proof of Lemma 6

Lemma 6. *After a vertex v_0 is inserted into or deleted from a hyperedge e_0 in $H = (V, E)$,*
(1) the inner trussness of each vertex pair in e_0 increases/ decreases by at most 1;
(2) the inner trussness of each vertex pair outside e_0 remains unchanged.

Proof. Assume there are k vertices in e_0, then there are $k(k-1)/2$ vertex pairs. Each vertex pair is in $(k - 2)$ triangles, so they form a $(k - 2, 0)$-truss. After the vertex v_0 is inserted, each vertex pair is in $(k - 1)$ triangles, including the newly inserted vertex pairs linked to v_0, so the inner trussness of all vertex pairs in e_0 is $(k - 2) + 1$ now if there is no limitation on outer trussness ($k_{out} = 0$). The inner trussness of each vertex pair in e_0 is increased by 1 in this scenario. They may also not increase if there is a limitation on outer trussness ($k_{out} > 0$) and some of the newly inserted vertex pairs do not meet the condition. As for vertices outside e_0, the change of e_0 has no influence on them according to the definition of inner triangles, thus the inner trussness of each vertex pair outside e_0 remains unchanged. □

A.5 The Proof of Lemma 7

Lemma 7. *Suppose a vertex u_0 is inserted into a hyperedge e_0 in H. Δs_{max} is the maximum outer support change of the neighbor vertex pairs of the inserted vertex pairs and $0 \leq \Delta s \leq \Delta s_{max}$. $lv_{min}[e(u,v)]$ is the lowest level of the outer triangle that contain $e(u,v)$. $minsup^H_{out}[e(u),e(v))]$ is the smaller one of $sup^H_{out}[e(u)]$ and $sup^H_{out}[e(v)]$. For the new outer trussness $\tau'_{out}[e(u,v)]$ of a vertex pair $e(u,v)$ in H we have:*

$$\tau'_{out}[e(u,v)] = \begin{cases} \tau_{out}[e(u,v)] & ,minsup^H_{out}[e(u),e(v))] \leq \tau_{out}[e(u,v)] - \Delta s_{max} \\ \tau_{out}[e(u,v)] + \Delta s & ,others \\ \tau_{out}[e(u,v)] & ,lv_{min}[e(u,v)] > \tau_{out}[e(u,v)] + \Delta s_{max} \end{cases} \tag{1}$$

Proof. According to Lemma 4, the outer trussness of each vertex pair will be increased by at most Δs_{max}. The increase of outer trussness of vertex pairs means that the levels of the newly formed triangles are high enough or the levels of triangles in original hypergraphs increase to a higher level. For the first condition, if $minsup^H_{out}[e(u),e(v))] \leq \tau_{out}[e(u,v)] - \Delta s_{max}$, then none of triangles that contain $e(u,v)$ has a higher enough level to increase trussness of $e(u,v)$. For the third condition, if $lv_{min}[e(u,v)] > \tau_{out}[e(u,v)] + \Delta s_{max}$, then these triangles support the vertex pairs not only in the new hypergraphs but also in the original hypergraphs. They will not lead to trussness change either. Only the trussness of the vertex pairs meeting the second condition may increase. Because the upper bound of the increase of a vertex pair is Δs_{max} according to Lemma 4, the exact change Δs will not exceed this bound. □

B An Example of Decomposition Algorithm on Fig.2

Table 4 shows the execution process of a $(0,3)$-truss decomposition on the hypergraph in Fig.2. In this table, the first column lists different vertex pairs. The following columns are their outer supports in each step, and the last column is their outer trussness. Firstly, each vertex pair calculates its inner and outer support according to the definition. Then the algorithm checks whether they meet the conditions of k_{in}. All vertex pairs are eligible for $k_{in} = 0$ here. Next, the algorithm checks for vertex pairs whose outer support is no greater than $k = 0$. We find that $e_4(u_3,u_6)$ and $e_4(u_5,u_6)$ do not have any outer triangles, so they should be deleted in this step. When we decide which vertex should be removed, the outer supports of these vertices are considered. The outer support of $e_4(u_6)$ is 0 for the maximum support of vertex pairs that contained it ($e_4(u_3,u_6)$, $e_4(u_5,u_6)$) is 0. Similarly, the outer supports of $e_4(u_3)$ and $e_4(u_5)$ are 3 and 1. Thus in this step, u_6 is removed from e_4 and two vertex pairs $e_4(u_3,u_6)$, $e_4(u_5,u_6)$ have their outer trussness reduced to 0. After repeating this check, no vertex pairs need to be removed and the algorithm goes to $k = 1$.

In the step $k = 1$, vertex pairs $e_3(u_1,u_5)$ and $e_4(u_3,u_5)$ need to be deleted and u_5 is chosen to be removed from e_3 and e_4. In the step $k = 2$, vertex pairs

Table 4. The execution of truss decomposition for the hypergraph in Fig.2

Vertex Pair	*init*	$k=0$	$k=1$	$k=2$	$k=3$	*sup*
$e1(u1,u2)$	3	3	3	3	-	3
$e2(u1,u2)$	4	4	4	3	-	3
$e1(u2,u4)$	3	3	3	3	-	3
$e2(u2,u4)$	4	4	4	3	-	3
$e1(u1,u4)$	3	3	3	3	-	3
$e2(u1,u4)$	4	4	4	3	-	3
$e1(u1,u3)$	3	3	2	-	-	2
$e1(u2,u3)$	2	2	2	-	-	2
$e1(u3,u4)$	2	2	2	-	-	2
$e3(u1,u5)$	1	1	-	-	-	1
$e4(u3,u5)$	1	1	-	-	-	1
$e4(u3,u6)$	0	-	-	-	-	0
$e4(u5,u6)$	0	-	-	-	-	0

$e_1(u_1,u_3)$, $e_1(u_2,u_3)$ and $e_1(u_3,u_4)$ need to be deleted and u_3 is chosen to be removed from e_1. Finally, in the step $k=3$, all vertex pairs are deleted and the peeling process terminates.

C Other Algorithms and Proofs in Section 4

Algorithm 3: Trussness Deletion

```
Input  : Hypergraph $H(V,E)$, truss parameters $k_{in}$ and $k_{out}$, deleted vertex
         set $U_\Delta$, update hyperedge set $E_\Delta$, deleted vertex pair set $\Delta_H$
Output : Update inner and outer trussness for each vertex pair
1 Delete $U_\Delta$ from $H$;
2 for vertex pair $e(u,v)$ in $E_\Delta$ do
3     calculate $sup_{in}^{H}[e(u,v)]$;
4 for vertex pair $e(u,v)$ in $(k_{in},0)$-truss do
5     if $e(u,v) \notin E_\Delta$ or $e(u,v) \in E_\Delta$ and $sup_{in}^{H}[e(u,v)] \geq k_{in}$ then
6         $h_0[e(u,v)] = \tau_{out}[e(u,v)]$;
7 UpdateTrussness($H(V,E)$, $H_0[e(u,v)]$);
8 Return each vertex pair in $(k_{in},k_{out})$-truss;
```

C.1 Deletion Algorithm

Similar to Lemma 7, if the highest level of the outer triangle that contains a vertex pair is less than the original outer trussness of the vertex pair minus Δs_{max} or if both vertices of a vertex pair have a support that is larger than the original outer trussness of the vertex pair plus Δs_{max}, the outer trussness of the vertex pair remains unchanged after the update. However, the initial h-index of each vertex pair does not need to be recomputed since vertex pairs only decrease their outer trussness, with the original outer trussness serving as a natural upper bound. After excluding the vertex pairs whose inner trussness will not meet the condition, each vertex pair sets its initial h-index as $\tau_{out}[e(u,v)]$ and then the algorithm calls $UpdateTrussness$.

C.2 Trussness Update Algorithm

Algorithm 4: UpdateTrussness

```
Input  : Hypergraph $H(V,E)$, the initial h-index array of vertex pairs $H$
Output : Update outer trussness for each vertex pair
1  for $e(u,v) \in E(H)$ do
2      if $H[e(u,v)] \neq \tau_{out}[e(u,v)]$ then
3          $Activate[e(u,v)] \leftarrow TRUE$;
4      else
5          $Activate[e(u,v)] \leftarrow FALSE$;
6  $Update \leftarrow TRUE$;
7  while $Update$ do
8      $Update \leftarrow FALSE,\ L, N \leftarrow \emptyset$;
9      for $u \in V(H)$ in parallel do
10         $H_v[e(u)] = max\{H[e(u,v_i)]\}, v_i \in e$;
11     for $e(u,v) \in E(H)$ in parallel do
12         if $Activate[e(u,v)]$ is $FALSE$ then
13             continue;
14         for $w \in N(u) \cap N(v)$ do
15             $L = L \cup min\{H[e_i(u,w)], H[e_j(v,w)]\},\ N = N \cup e_i(u,w), e_j(v,w)$;
16         $h \leftarrow max\{h-index(L), min\{H_v[e(u)], H_v[e(v)]\}\}$;
17         if $h \neq H(u,v)$ then
18             $Update \leftarrow TRUE$;
19             for $e'(u',v') \in N$ and $h < H[e'(u',v')] < H[e(u,v)]$ do
20                 $Activate[e'(u',v')] \leftarrow TRUE$;
21         $H[e(u,v)] \leftarrow h,\ Activate[e(u,v)] \leftarrow FALSE$;
22 for $e(u,v) \in E(H)$ do
23     $\tau_{out}[e(u,v)] = H[e(u,v)]$;
24 return outer trussness for each vertex pair;
```

Algorithm 4 updates the outer trussness of vertex pairs based on the h-index. In lines 1-5, the h-index of each vertex pair is initialized with the upper bounds in Algorithm 2. If the h-index of a vertex pair is larger than its original outer trussness, then its *Activate* flag is true and will be updated in the first round of convergence. We use a boolean indicator *updated* to check the convergence and terminate the program. *updated* flag stays true if any edge's h-index is changed. In lines 9-10, the supports of vertices in hyperedges are calculated. For each vertex pair $e(u, v)$, lines 14-15 find all triangles containing the edge $e(u, v)$ and collect the h-indices of the neighbor vertex pairs of $e(u, v)$ by choosing the smaller one. Line 16 computes the h-index of $e(u, v)$ and compares it with the smaller supports of vertices u and v. The larger value will be set as the new h-index of vertex pair $e(u, v)$ according to Definition 4. If the new h-index is smaller than its current one, it means the h-index of $e(u, v)$ is updated and *updated* is set to be true. In lines 19-20, affected neighbors of $e(u, v)$ update their *Activate* flag to true and are ready for the next round of convergence.

C.3 Degree Level

Degree level is a concept presented by Ahmet et al. in [13]. They relate it to the convergence of their (r, s)-nucleus and generalize the iterative h-index computation idea for any nucleus decomposition, including k-core and k-truss. We propose the upper bound of our algorithm based on this idea.

Note the set of all vertex pairs as $V(G)$, the set of all outer triangles as $T(G)$, the union of these two sets as $C(G) = V(G) \cup T(G)$ and the i-th level as the set L_i. L_0 is the set of vertex pairs that are in the minimum number of $T(G)$ in $C(G)$. L_i is the set of vertex pairs that are in the minimum number of $T(G)$ in $C(G)/\bigcup_{j<i} L_j$. For example in Fig.C.3, edges in red are all in 0 triangles, so they are in L_0. Similarly, edges in green are all in 1 triangles, so they are in L_1. And the left edges are all in L_2. We prove that the h-indices in the i-th level converge within i iterations of the update operation.

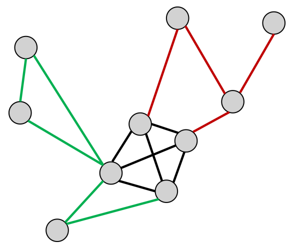

Fig. 4. An example graph of degree level

Lemma 8. *Let $i \leq j$. For any vertex pair $vp_i \in L_i$ and vertex pair $vp_j \in L_j$, $\tau_{out}[vp_i] \leq \tau_{out}[vp_j]$.*

Proof. Assume $L^{'} = \bigcup_{r \geq i} L_r$, the union of all levels i and above and $G^{'}$ is the induced subgraph based on $L^{'}$. According to the definition, the outer support of vp_i in $G^{'}$ $sup_{out}^{G^{'}}[vp_i]$ is the minimum outer support of vertex pairs in $G^{'}$ and $sup_{out}^{G^{'}}[vp_j] \geq sup_{out}^{G^{'}}[vp_i]$. Assume that vp_i is in a $(0, \tau_{out}[vp_i])$-truss T.

If the vertex pairs in T are all in $L^{'}$, then $\tau_{out}[vp_i]$ is not greater than the minimum outer support of vertex pairs in $G^{'}$. While $\tau_{out}[vp_j]$ is equal to $min_{vp_j \in P}$ $\{$the minimum outer support of vertex pairs in $P\}$. Thus, $\tau_{out}[vp_i] \leq \tau_{out}[vp_j]$.

In another case, if there are some vertex pairs in T are not in $L^{'}$. i.e., there are some vertex pairs are in L_k, where $k < i$. Note the vertex pairs that makes the mimimum k as vp_k. Since T is a $(0, \tau_{out}[vp_i])$-truss, $sup^P_{out}[vp_k] \geq \tau_{out}[vp_i]$. Assume $L^* = \bigcup_{r \geq k} L_r$ and G^* is the induced subgraph based on L^*. Thus the vertex pairs in T are all in L^*. We have $sup^{G^*}_{out}[vp_k] \geq sup^T_{out}[vp_k] \geq \tau_{out}[vp_i]$. Because that $vp_k \in L_k$, $sup^{G^*}_{out}[vp_k]$ is equal to the minimum outer support of vertex pairs in G^*. Because that $j > k$ and $vp_j \in G^*$, $\tau_{out}[vp_j]$ is no less than the minimum outer support of vertex pairs in G^*. Considering all above, we have $\tau_{out}[vp_i] \leq \tau_{out}[vp_j]$. □

D More Experimental Results in Section 5

We performed experiments on real-world hypergraphs to evaluate the stability, scalability, parallelism, and generality of our algorithms. For stability, we measured the average time of update in the scenario of sets with different sizes. For scalability and parallelism, we change the size of the original hypergraphs and the number of threads. For generality, we change the values of k_{in} and k_{out}.

The program reads two files as input: the original hypergraph and the update operations. In the file of the original hypergraph, each row represents a hyperedge. The index of the row is the label of the hyperedge and the numbers in that row are the labels of the vertices in the hyperedge. In the file of the update operations, each row represents the updated vertices in a hyperedge. The first integer in a row is the label of the hyperedge, the following integers are the labels of vertices that will be inserted into or deleted from the hyperedge. During the experiment, vertices in the original hypergraph are deleted from the hyperedges according to the second file and then inserted back to test the deletion and insertion algorithms in succession.

D.1 Scalability Evaluation

In this subsection, we evaluated the scalability of our algorithms by changing the size of the original hypergraphs. Specifically, we sample the original hypergraphs from a ratio of 10% to 100%. For each sub-hypergraph, we randomly choose 1000 hyperedges to update. The thread numbers are fixed to 8. The results of the insertion and deletion of the static algorithm and maintenance algorithm are shown in Tables 5 and 6. The first column represents the sample ratio of the updated set. The first row represents different algorithms and hypergraphs. $BC(st.)$ means the time of the static algorithm spent on BC, and $VI(m)$ means the time of the maintenance algorithm spent on VI. The units of the results are milliseconds. Updated hyperedge sets are sampled randomly for static hypergraphs, and the latest hyperedges are selected for temporal hypergraphs.

Table 5. Influence of the size of the hypergraph (insertion)

Percentage	BC(st.)	VI(st.)	PD(st.)	BC(m)	VI(m)	PD(m)
10%	6396.8	3297.1	82.346	4374.5	302.56	6.2914
20%	6573.7	3516.3	84.556	4388.2	305.62	4.7745
30%	6429.7	3713.2	87.457	4708.1	312.46	4.3940
40%	6732.8	3994.7	91.723	4114.3	329.33	5.2292
50%	7092.3	4100.2	95.167	4896.3	360.71	6.0310
60%	7430.1	4131.2	105.35	6628.5	394.88	8.7710
70%	7276.3	4209.5	119.53	6444.9	375.89	8.8568
80%	7370.1	4344.5	131.82	6801.6	378.71	11.597
90%	7919.8	4430.03	137.74	6830.6	421.50	11.802
100%	8183.7	4713.2	141.72	7268.3	431.23	13.364

Overall, as shown in Tables 5 and 6, the cost time increases with the increase of the sizes of hypergraphs, since there are more potential vertices that are affected as the hypergraph becomes larger. The trussness of more vertex pairs needs computing. The static algorithm needs more steps to peel the vertex pairs and the maintenance algorithm needs more rounds to converge from the initial h-index to the exact trussness value. The impact of hypergraph size is more pronounced in larger hypergraphs. This is because the same percentage of a large hypergraph contains more vertices and hyperedges than a small hypergraph. Given that the time cost increases nearly linearly with the hypergraph scale, our algorithms perform desirable scalability.

Table 6. Influence of the size of the hypergraph (deletion)

Percentage	BC(st.)	VI(st.)	PD(st.)	BC(m)	VI(m)	PD(m)
10%	7208.2	3324.2	88.243	3478.9	343.67	4.003
20%	6899.1	3549.9	85.758	4008.5	351.40	4.165
30%	7046.6	3862.8	95.207	4128.0	376.90	5.350
40%	7275.7	4141.3	97.645	4437.5	394.40	6.837
50%	7355.8	4205.7	105.25	6250.2	447.35	8.233
60%	7587.3	4663.6	113.24	6352.3	440.18	10.595
70%	7780.5	4697.0	119.35	6624.1	446.00	15.194
80%	7370.1	4344.5	131.82	6801.6	378.71	11.597
90%	8031.9	4657.9	133.49	6838.4	466.76	12.808
100%	8450.9	4776.2	153.36	7495.5	473.74	14.890

In some cases, the time cost of a part of the whole hypergraph may be less than others, such as 30% of $BC(st.)$. This is from variations in sample areas. If vertex pairs in the sample area do not meet the requirements of inner support, they will be deleted in the early steps and will not influence the remaining vertices. Thus the number of updated vertex pairs can be small, resulting in a shorter update time compared to other scenarios.

D.2 Parallelism Evaluation

In this subsection, we evaluated the parallelism of our parallel algorithms by setting different numbers of threads. For each hypergraph, 1000 vertices among the random sample hyperedges are deleted first and then inserted back. The number of threads is set to 1, 2, 4, 8, 16, and 32. The experimental results are presented in Tables 7 and 8, where the first column represents the number of threads, and the first row represents different algorithms and hypergraphs. $BC(st.)$ means the time of the static algorithm spent on BC, and $VI(m)$ means the time of the maintenance algorithm spent on VI. The units of the results are milliseconds.

Table 7. Influence of the thread number (insertion)

Thread Number	BC(st.)	VI(st.)	PD(st.)	BC(m)	VI(m)	PD(m)
1	30935	19405	548.89	15851	975.05	27.577
2	16611	10864	354.99	10556	648.79	21.621
4	11425	7193.1	169.34	8493.5	491.51	16.488
8	8505.9	4420.1	154.35	7687.1	407.65	16.597
16	8740.8	4811.3	135.83	6238.2	412.23	16.207
32	8849.1	4934.2	165.98	6064.8	480.67	15.756

Table 8. Influence of the thread number (deletion)

Thread Number	BC(st.)	VI(st.)	PD(st.)	BC(m)	VI(m)	PD(m)
1	26255	17360	600.03	16091	944.35	29.562
2	14690	9367.1	337.84	10713	654.46	22.299
4	9419.2	6255.5	263.07	9721.9	521.90	16.367
8	9059.4	4265.8	144.36	7440.7	435.97	16.163
16	9101.7	3142.9	103.44	7974.2	436.90	20.208
32	9779.4	3038.9	104.25	7653.5	508.57	20.044

Generally, the execution time of the algorithms decreases as the number of threads increases. The larger the scale of a hypergraph, the more significant the speedup of the parallel algorithm is. When the number of threads increases to around 16, the time cost reaches the minimum point. It is mainly because the vertex pairs with lower trussness are peeled off first in the convergence of the static algorithm and account for a more significant limitation of parallelism.

More threads can handle more of these vertex pairs simultaneously. However, the update of the support map in each step is in a critical zone of the parallel process, increasing the number of threads does not significantly impact the update time when the thread count is high. Similarly, the critical zone in the maintenance algorithm is the update of the activated vertex pair set and the h-index of each vertex pair that updates in this round. Therefore, as the number of threads increases continually, there is no significant acceleration effect.

D.3 Generality Evaluation

Table 9. Influence of different k_{in} and k_{out} (insertion)

(k_{in}, k_{out})-truss	BC(st.)	VI(st.)	PD(st.)	BC(m)	VI(m)	PD(m)
(0,5)	11733	5427.4	198.48	8402.1	556.94	23.564
(2,5)	8585.6	4481.0	156.78	7264.3	420.57	16.937
(2,10)	8573.1	4349.8	159.15	7442.0	406.89	16.512
(5,10)	7600.9	3774.2	143.79	6970.9	321.86	15.833

Table 10. Influence of different k_{in} and k_{out} (deletion)

(k_{in}, k_{out})-truss	BC(st.)	VI(st.)	PD(st.)	BC(m)	VI(m)	PD(m)
(0,5)	12210	5584.8	206.52	8802.7	580.14	23.263
(2,5)	9237.6	4663.9	148.43	7834.6	439.50	18.989
(2,10)	9307.2	4670.5	147.42	7752.0	426.78	18.997
(5,10)	8288.3	3641.8	138.76	7259.0	334.45	17.263

In this subsection, we test the performance of our algorithms when k_{in} and k_{out} changes. We randomly sample 1,000 hyperedges from the hypergraphs, then randomly select several vertices from each hyperedge and delete them before inserting them. Then we record the time when the combinations of k_{in} and k_{out} are different: (0,5), (2,5), (2,10), (5,10). The thread numbers are fixed to 8. The results of the insertion and deletion of the static algorithm and maintenance algorithm are shown in Tables 9 and 10, where the first column represents different restrictions on k_{in} and k_{out}, and the first row represents different algorithms and hypergraphs. $BC(st.)$ means the time of the static algorithm spent on BC and $VI(m)$ means the time of the maintenance algorithm spent on VI. The units of the results are milliseconds.

It can be observed that the execution time of the static and maintenance algorithms mainly depends on k_{in} and nearly does not change as the k_{out} increases. Because our algorithms maintain the exact outer trussness for each vertex pair when the requirement is that the inner trussness is larger than k_{in}, i.e., the level of (k_{in}, k_i)-truss where $k_i = 0, 1, 2, \ldots$. The changes in k_{out} will not influence the process of algorithms. When k_{in} increases, more vertex pairs will be free from being calculated because they do not meet the condition of k_{in} and will be removed before the first step. Additionally, the reduction in time is less noticeable for dense hypergraphs for there are fewer vertex pairs with low inner trussness in the hypergraph and they may all fit the restriction when k_{in} becomes larger.