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, People's Republic of China

qshua@hust.edu.cn

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* [14] 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 two orders of magnitude 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. Figure 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.

Q.-S. Hua—This work was supported in part by National Science and Technology Major Project 2022ZD0115301.

Y. Chen et al. (Eds.): COCOON 2024, LNCS 15162, pp. 538–550, 2025.
https://doi.org/10.1007/978-981-96-1093-8_44

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.

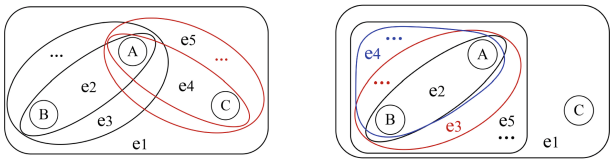

Fig. 1. Two different scenarios that cannot be distinguished in [7].

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

[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 $(e_1, e_2, e_3, e_4, e_5)$.

which can clarify different cohesiveness between different triangles, then we study 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.

Due to space constraints, this paper presents only the main results and discussion. The full version, including proofs and additional experimental results, can be found in [9].

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 $T_H[e_0(u, v)] = \{\triangle[u^{e_0} v^{e_i} w^{e_j}] \mid w \in V(H)\}$.

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 numbers of these

two kinds of triangles need 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 $sup^H_{in}[e_0(u, v)]$. Its outer support is defined as the number of outer triangles that contain $e_0(u, v)$, denoted as $sup^H_{out}[e_0(u, v)]$.*

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

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) $sup^S_{in}[e_0(u, v)] \geq k_{in}$, and
(2) $sup^S_{out}[e_0(u)] \geq k_{out}$ and $sup^S_{out}[e_0(v)] \geq k_{out}$.

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 $\tau^{out=l}_{in}[e_0(u, v)] = k$, and its outer trussness (when $k_{in} = k$) as $\tau^{in=k}_{out}[e_0(u, v)] = l$.*

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[2]. Note that we relax the restriction on the outer trussness in Definition 4, which is different from ordinary graphs. It contains not only vertex pairs whose outer supports are no less than k_{out}, but also 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. Definition 4 guarantees that the largest subgraph can be found. Lemma 1 clarifies this issue and the proof is in [9].

[2] 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.

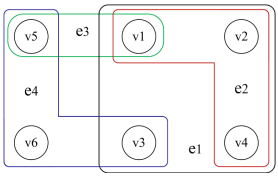

Fig. 2. Different triangles in a hypergraph

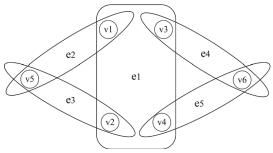

Fig. 3. Missed Cohesive Structure

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 $e_0(v_1, v_3)$, $e_0(v_1, v_4)$, $e_0(v_2, v_3)$, $e_0(v_2, v_4)$ 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. 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 [10]. 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 [9]. 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.

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 $\triangle[u^{e_a}v^{e_b}w^{e_c}]$

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. [11] and Luo et al. [6,12], 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 or a vertex update. 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 [13]. 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* [14] of vertex pairs in the hypergraph. The proofs of the subsequent Theorems 1 and 2 are covered in [9].

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 [9].

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.

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 ← 0;
 4 while H ≠ ∅ do
 5     while ∃ 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) ∈ e_i do
 9                 if k > 0 then
10                     τ_in[e_i(u, v)] ← k_in, τ_out[e_i(u, v)] ← k − 1;
11         for each sup^H_out[e_i(u, v)] ≤ 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)] ≤ sup^H_out[e_i(v)];
14                 Remove the vertex u from e_i;
15                 for u_i ∈ e_i do
16                     τ_in[e_i(u, v)] ← k_in, τ_out[e_i(u, u_i] ← 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 ≥ k_out ;
```

Theorem 1. *Alg.1 calculates the (k_{in}, k_{out})-truss correctly. Its depth is $O(m * c^2_{max} * h_{max})$ and its 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 [9].

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 [9].

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* $O(|\Delta_H| * c_{max}^2 * h_{max}^2 + n * c_{max} * h_{max} * t_{max})$.

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 [9].

Table 2 includes datasets from the real world that can be accessed through the KONECT project[3]. 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. Table 3 shows the results of the insertion of the static algorithm and the maintenance algorithm.

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

Algorithm 2: Trussness Insertion

Input : Hypergraph $H(V, E)$, truss parameters k_{in} and k_{out}, new vertex set U_Δ, update hyperedge set E_Δ, new vertex pair set Δ_H
Output : Update inner and outer trussness for each vertex pair

Insert U_Δ into H;
for *vertex pair* $e(u, v)$ *in* Δ_H *in parallel* **do**
  calculate $sup^H_{in}[e(u, v)]$ and $sup^H_{out}[e(u, v)]$;
  $h_0[e(u, v)] = sup^H_{out}[e(u, v)]$;
Calculate the maximum outer support change Δs_{max} of the neighbor vertex pairs of the inserted vertex pairs Δ_H;
if $e(u, v) \in E_\Delta$ *and* $sup^H_{in}[e(u, v)] >= k_{in}$ **then**
  $h_0[e(u, v)] = sup^H_{out}[e(u, v)]$;
for *vertex pair* $e(u, v)$ *in* $(k_{in}, 0)$*-truss in parallel* **do**
  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**
    $h_0[e(u, v)] = \tau_{out}[e(u, v)]$;
  else
    $h_0[e(u, v)] = \tau_{out}[e(u, v)] + \Delta s_{max}$;
UpdateTrussness($H(V, E)$, $H_0[e(u, v)]$);
Return each vertex pair in (k_{in}, k_{out})-truss;

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 two orders of magnitude speedup.

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 two orders of magnitude speedup in time consumption compared to the decomposition algorithm. We will focus on optimizing these algorithms to reduce their time complexities in the future.

References

1. Duan, D., Li, Y., Li, R., Lu, Z.: Incremental k-clique clustering in dynamic social networks. Artif. Intell. Rev. **38**, 129–147 (2012). https://doi.org/10.1007/s10462-011-9250-x
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. Nat. Secur. Agency Tech. Rep. **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: 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: 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 Trans. Comput. Soc. Syst. **7**(6), 1435–1446 (2020)
7. Wang, X., Chen, Y., Zhang, Z., Qiao, P., Wang, G.: Efficient truss computation for large hypergraphs. In: WISE 2022, pp. 290–305 (2022)
8. Gu, Y., et al.: Distributed hypergraph processing using intersection graphs. IEEE Trans. Knowl. Data Eng. **34**(7), 3182–3195 (2020)

9. Wang, M., Hua, Q.-S., Wang, Y., Jin, H., Shao, Z.: Parallel truss maintenance algorithms for dynamic hypergraphs. https://qiangshenghua.github.io/papers/cocoon24full.pdf
10. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT press (2022)
11. Huang, X., Cheng, H., Qin, L., Tian, W., Yu, J.X.: Querying k-truss community in large and dynamic graphs. In: SIGMOD 2014, pp. 1311–1322 (2014)
12. Luo, Q., Yu, D., Cheng, X., Sheng, H., Lyu, W.: Exploring truss maintenance in fully dynamic graphs: a mixed structure-based approach. IEEE Trans. Comput. **72**(3), 707–718 (2022)
13. Lü, L., Zhou, T., Zhang, Q.M., Stanley, H.E.: The H-index of a network node and its relation to degree and coreness. Nat. Commun. **7**(10168), 1–7 (2016)
14. Sariyuce, A.E., Seshadhri, C., Pinar, A.: Local algorithms for hierarchical dense subgraph discovery. Proc. VLDB Endowment **12**(1), 43–56 (2018)