Summary of Changes in Revision: Truss Decomposition in Hypergraphs

Hongchao Qin, Guang Zeng, Rong-Hua Li, Longlong Lin, Ye Yuan, Guoren Wang

We are very grateful to the anonymous reviewers for their insightful and valuable comments. We have revised the manuscript and tried our best to address all the comments. The main changes are summarized below.

- **Clarified the positioning with respect to the state of the art:** We have revised Section 1 to introduce different implementations of the truss model, including the general baseline Truss based on the bipartite representation of a graph, Bitruss for bipartite graph models, and the hypergraph truss model proposed by our team members. We have also added comparisons in Section 5 to evaluate the performance of these models.
- **Provided more detailed analysis of the DBLP case study:** We have revised the DBLP case study in Section 5.3 to better illustrate the advantages of the hyper k-truss model over traditional k-truss methods. We have highlighted how the hyper k-truss model can more accurately identify cohesive subgraphs and reveal meaningful patterns in the collaboration network.
- **Added missing results in the plots and elaborated further on them:** We have added the missing figures to an online supplementary material for a more comprehensive presentation of the results across all datasets.
- **Enhanced the understanding of the differences of the proposed method compared to existing ones:** We have provided a more detailed comparative analysis of our direct hypergraph approach and traditional methods applied to bipartite representations. We have explained how our approach captures the cohesiveness of hyperedges and provides a more structured representation of groupwise relationships.
- In addition, we have addressed the minor issues pointed out by the reviewers, such as typos and formatting errors. The revised manuscript presents a clearer and more polished version of our work.

The point-to-point responses are given below.

1 Response to Reviewer #3

Thank you for recognizing this work during D1-D3. Below are the responses to W1-W3 (which correspond to D4-D6 respectively).

Comment 1. *(W1) It is not sufficiently clear how is the solution provided by HTRUSS directly on a hypergraph qualitatively different from running normal k-truss (e.g., TRUSS) on a bipartite representation of a graph.*

(D4) The paper needs to better articulate the qualitative differences between their direct hypergraph approach and traditional methods applied to bipartite representations. While they briefly discuss this in the motivation, a more detailed comparative analysis would strengthen their argument for working directly with hypergraphs. This is especially needed when running normal k-truss on a bi-partite representation of a hypergraph is more efficient than the proposed approach...

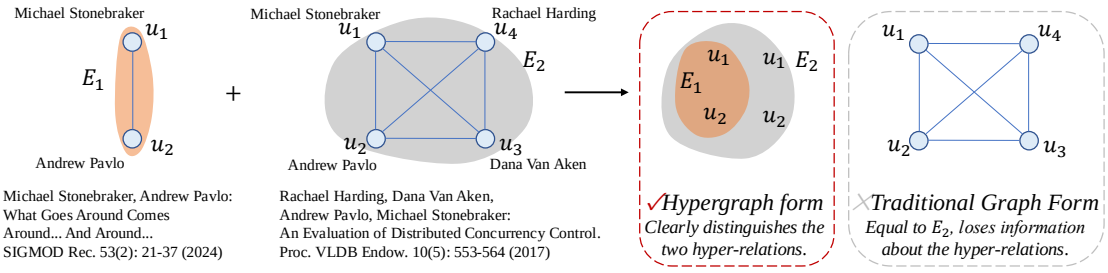


Figure 1: The differences between the hypergraph model and traditional representations

Response: Thank you for your insightful comments. Indeed, there are several implementations of the truss model.

(i) The general baseline Truss used in our experiments is based on the bipartite representation of a graph, where each hyperedge is transformed into a complete graph (note that the previous description was slightly incorrect, as there are no triangles in a bipartite graph, see Section 5). We then apply the k-truss models on top of this representation.

(ii) For bipartite graph models, Kai Wang, Xuemin Lin, et al. (Efficient Bitruss Decomposition for Large-scale Bipartite Graphs. ICDE 2020: 661-672) introduced *Bitruss*, where the key definition is that the butterfly support of each edge in a k-bitruss is greater than or equal to k. Here, a butterfly can be understood as a tuple $\{E_i, E_j, u_k, u_l\}$ with connections $\{E_i, u_k\}$, $\{E_j, u_k\}$, $\{E_i, u_l\}$, and $\{E_j, u_l\}$.

(iii) Additionally, regarding hypergraph trusses, as you mentioned in D6, our team memebers (Xinzhou Wang, Yinjia Chen, Zhiwei Zhang, Pengpeng Qiao, Guoren Wang: Efficient Truss Computation for Large Hypergraphs. WISE 2022: 290-305) has previously studied truss models in hypergraph. However, the truss model in that paper has certain limitations in defining hypergraph triangles: given parameters α and β, it requires $E_x \cap E_y \cap E_z \geq \alpha$ and $E_x \cup E_y \cup E_z \geq \beta$, which now appears restrictive. The model we propose in this paper is more general than this definition.

We have revised **Section 1** to introduce the above implementations of the truss model and added comparisons in **Section 5** to evaluate the performance of these models.

Below we will discuss the qualitative differences between our direct hypergraph approach and traditional methods applied to bipartite representations. As mentioned, if we do not utilize hypergraph modeling, we risk losing the semantic information related to the overlapping parts of dense subgraphs, which is crucial for accurately modeling truss structures. For example, consider the collaboration between Michael Stonebraker and Andrew Pavlo in their paper presented at SIGMOD Rec. 2024, which forms the first relationship E_1. If we then include the collaborative work of Rachael Harding, Dana Van Aken, Andrew Pavlo and Michael Stonebraker in their VLDB 2017 paper, represented as E_2 using a hypergraph allows us to clearly distinguish these two hyper-relations.

In contrast, if we were to represent these relationships using traditional graph forms, it will be equivalent to E_2, thereby losing critical information about the hyper-relations. This loss of information can lead to inaccuracies in the truss computation, as the unique contributions of each hyperedge are essential for understanding the overall structure and relationships within the data.

In the revised version, we will expand on these points and provide additional examples to illustrate the advantages of our hypergraph approach over traditional methods. We believe this will strengthen our argument for the necessity of working directly with hypergraphs in the context of truss computation.

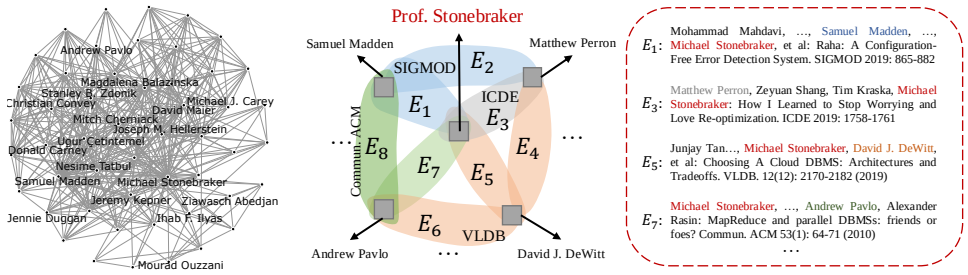
Comment 2. *(W2) The DBLP case study is not sufficiently clear what it is showing. This need to be stronger.*

(D5) The DBLP case study, while interesting, needs more detailed analysis. The authors should provide more concrete insights about what the discovered structures reveal about the underlying collaboration network and why these insights couldn't be obtained through traditional graph analysis.

Response: Thank you for your feedback on the DBLP case study. We understand that the

analysis needs to be more detailed and provide deeper insights. We will revise the case study to better illustrate the advantages of the hyper k-truss model over traditional k-truss methods.

In the revised version **(Section 5.3)**, we will provide a more in-depth analysis of the discovered structures in the DBLP dataset. Specifically, we will highlight how the hyper k-truss model can more accurately identify cohesive subgraphs and reveal meaningful patterns in the collaboration network. We will also explain why these insights could not be obtained through traditional graph analysis.



(a) Traditional k-truss of Prof. Stonebraker

(b) Hyper k-truss of Prof. Stonebraker

Figure 2: Case study of hyper k-truss on DBLP

As shown in Fig.2(a), the cohesive subgraph *k-truss* ($k = 5$) includes a large number of collaborators, making it difficult to identify meaningful real-life communities. A further complication is that multiple edges connect to Prof. Stonebraker in Fig.2(a), making it even harder to extract well-defined dense subgraphs. In *hyper k-truss*($k = 5$), to improve readability, we provide a summary of the frequent relationships in Fig.2(b). Since each hyperedge represents a publication—for example, E_1 denotes a SIGMOD paper co-authored by Samuel Madden and Prof. Stonebraker, while E_3 corresponds to an ICDE paper co-authored by Matthew Perron and Prof. Stonebraker—we can infer Prof. Stonebraker's frequent participation in conferences such as SIGMOD, VLDB, and ICDE, allowing us to better interpret these relationships. The key difference between these two models lies in how they capture collaborations. In the *k-truss* model, all co-authorships are blended together: while we know that Samuel Madden frequently collaborates with Prof. Stonebraker, the model does not distinguish between their work in *Communications of the ACM* and *SIGMOD*. In contrast, the *hyper k-truss* model accounts for individual publications, ensuring that different collaborations influence the results separately. Therefore, *hyper k-truss* provides a more structured representation of groupwise relationships, making it superior to *k-truss* for modeling cohesive subgraphs in hypergraphs.

Comment 3. *(W3) Comparison to crucial related work is missing. For example [1]. Authors should elaborate what is different.*

(D6) The paper should include a more thorough comparison with crucial related work, particularly Wang et al. [1] who also address hypergraph truss computation. The authors should clearly explain how their approach differs in both methodology and capabilities.

Response: Thank you for your comment regarding the comparison with crucial related work. This work is an attempt by our team member, Xinzhou Wang, on the similar problem. However, the truss model in that paper has certain limitations in defining hypergraph triangles: given parameters α and β, it requires $E_x \cap E_y \cap E_z \geq \alpha$ and $E_x \cup E_y \cup E_z \geq \beta$, which now appears restrictive. The model we propose in this paper is more general.

In addition to our more general definition, we introduce a novel prefix forest technique in this paper to encode all hyperedges and count triangles within the prefix forest. This significantly reduces redundant calculations and improves computational efficiency.

We have revised the paper and added a comparison with the model [1] in **Section 5**.

2 Response to Reviewer #4

Thank you for your detailed comments. Your feedback has been invaluable in improving the quality of the paper. Below are our responses to D1–D16 (where D1–D10 correspond to W2, D11–D15 correspond to W3, and D16 corresponds to W1).

Comment 1. *D1: Alg 1: Why is a removal of a hyperedge guaranteed to reduce the support of its neighbors? That is not obvious. For instance in Figure 2, E4 is in a triangle with E6 and E8, but not with E1 and E6. Still, E4 has E1 as a neighbor. it is reasonable that removing E6 would reduce the support for E4 (and vice versa), because doing so destroys a triangle. But, it is not clear why removing E1 would reduce the support of E4. Should removing an edge reduce the support only of the neighbors it is forming a triangle with?*

Response: Thank you for bringing up this important point regarding the impact of hyperedge removal on the support of its neighbors. We realize that this is a common-sense error, and indeed, removing an edge reduces the support only when its neighbors form a triangle.

We appreciate your comments and have updated this issue in the revised version (**Section 3**). We have also checked other relevant parts to ensure that similar errors do not occur.

Comment 2. *D2: Sec 3, Should that stress its importance? "Combining these steps, we get [...] largely depends on the efficiency of hyper-triangles counting. ": This paragraph is there twice in a row.*

Response: Thank you for pointing out the repetition in **Section 3**. We apologize for this oversight. The repeated paragraph was indeed an error, and we have removed the duplicate in the revised version to ensure clarity and conciseness. We have also carefully reviewed the entire manuscript to check for any other potential repetitions or errors and have made necessary adjustments to improve the overall quality and readability of the paper.

We appreciate your attention to detail and are grateful for your feedback. We believe the revised manuscript now presents a clearer and more polished version of our work.

Comment 3. *D3: Sec 4.2, "Orientation steps focus on reducing the redundant counting since E1, E2, E3 and E3, E2, E1 are same triangles.": How would Algorithm 2 explore (or find) triangle E3, E2, E1? It only explores (or finds) solutions there $y < x$ and $z < y$. So "orientation" by HTC-B?*

Response: Thank you for your question regarding the exploration of triangles in Algorithm 2 and the role of the "orientation" step. We understand your concern about how Algorithm 2 would explore or find the triangle $\{E_3, E_2, E_1\}$ given its specific constraints.

In Algorithm 2, the exploration of triangles is indeed based on the constraints $z < y < x$. This approach ensures that each triangle is counted only once, thereby reducing redundant counting. The "orientation" step in Algorithm 2 is designed to handle the ordering of the vertices in a way that ensures consistency and avoids counting the same triangle multiple times in different orders. Therefore, it would explore the triangle in the order of $\{E_3, E_2, E_1\}$, which satisfies $z < y < x$; and it would not explore $\{E_1, E_2, E_3\}$ because it does not satisfy these constraints.

The orientation step in Algorithm 2 is similar to the one in Algorithm 3 (HTC-B), but it uses a different method to ensure consistent ordering. In contrast, Algorithm 3 uses the internal node IDs of the hyperedges to determine the orientation, where the node-ordered hyperedges $E_y \prec E_z$, there exists a k such that the indices of the nodes from 0 to k are the same, and the index of the $(k+1)$-th node in E_z is larger than in E_y. For example, the hyperedge $u_1, u_3, u_4, u_5 \prec u_1, u_4, u_5$ because the index of the first node is the same $(1 = 1)$, and the index of the fourth node is $3 < 4$.

We will provide a detailed explanation and examples in the revised version to clarify how Algorithm 2 explores triangles and the role of the "orientation" step, addressing the reviewer's query comprehensively (**Section 4.2**).

Comment 4. *D4: Alg 3, line 5, "s.t. $E_y \prec E_x$": Where is E_y bound? Should that rather say "s.t. $E_c \prec E_x$"?*

Response: Thank you for pointing out the potential confusion regarding the notation in Algorithm 3, line 5. You are correct that the notation should be "s.t. $E_c \prec E_x$". This was an oversight in the original manuscript.

We have revised the notation in the algorithm to ensure clarity and consistency.

Comment 5. *D5: Alg 3, line 7, pointless. "$Prec[E_y]! = emptyset$": How can $Prec[E_y]$ ever be empty? $Prec[E_y]$ is the set of common nodes of E_y and E_x.E_y is selected to be a neighbor of E_x. Be definition of neighbor (cf. Sec 2), E_y and E_x have to have at least one common node. So, the condition appears to be pointless.*

Response: You are absolutely correct that, by definition, $Prec[E_y]$ should not be $\emptyset$ if E_y is a neighbor of E_x, as they must share at least one common node.

However, we originally included this condition to enhance the algorithm's robustness in cases where it might encounter unexpected input or irregular data structures. This check was intended as a safeguard to prevent potential errors arising from data anomalies or edge cases during execution.

In the revised manuscript, we have removed this condition, as our current analysis confirms that it is indeed redundant.

Comment 6. *D6: Alg 3, line 9, "for each v in E_y, s.t. v is not in E_x do": It seems you could also say "for each v in $E_y - Prec[E_y]$"*

Response: Thank you for your suggestion on the phrasing in Algorithm 3, line 9. In the revised version, we have updated it to: "**for** each $v \in (E_y \setminus Prec[E_y])$ **do**". This change simplifies the expression and aligns it better with the notation used throughout the manuscript.

Comment 7. *D7: Alg 3, "flag": This is mostly an editorial comment. As far as I understand, the sole purpose of flag and Flag is to detect when E_z is bound to an edge we have look at before for the same combination of E_y and E_x. Fine. It would be way clear, presentation wise, if line 8 would initialize an empty set for the hyperedges already looked at, line 13 would add E_z to that set and line 10 would test E_z against the set. Sure, you can implement that with these counters, but that is an implementation detail that does not help communicating what the algorithm does in principle.*

Response: Thank you for your thoughtful feedback regarding the use of the "flag" variable in Algorithm 3. In the revised version, we have implemented your suggestion by introducing a set $HSet$ to store the hyperedges that have already been considered. This change enhances the clarity of the algorithm and better communicates its logic. Here's how the algorithm now reads:

Line 8: Initialize an empty set $HSet$ for the hyperedges already looked at.

Line 11: Test whether z is in $HSet$.

Line 13: Add z into $HSet$ after processing it.

This approach directly reflects the purpose of the "flag" variable and avoids the potential confusion caused by using counters. We believe this change will make the algorithm more intuitive and easier to understand (**Section 4.2**).

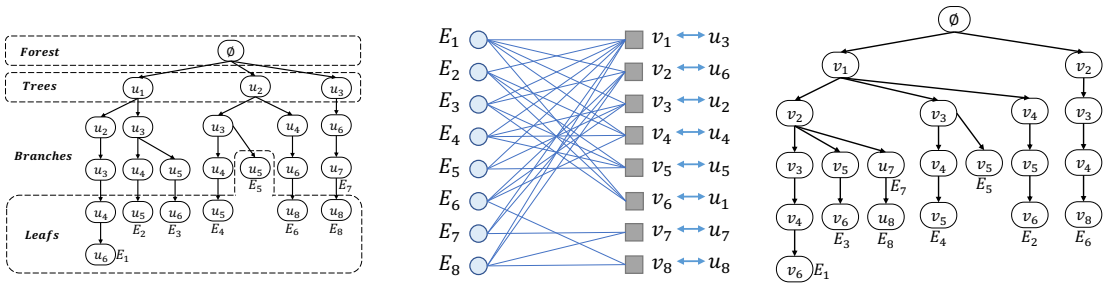
Comment 8. *D8: Sec 4.2: I do not understand what $E_y < E_x$ achieves over $y < x$. I understand its definition, but not its effect. I do not see it explained anywhere.*

Response: Recall D3: In Algorithm 2, the exploration of triangles is indeed based on the constraint $z < y < x$. However, in the later Algorithms 3–5, one hyperedge $E_y \prec E_z$ if the ID of the i-th node in E_y is no larger than that in E_z.

Let me give you an example. In the following Fig. 3(a), the partial order of the hyperedges from left to right in this tree is $E_1 \prec E_2 \prec E_3 \prec \ldots$. Additionally, if $E_y \prec E_z$, there exists a k such that the indices of the nodes from 0 to k are the same, and the index of the $(k+1)$-th node in E_z is larger than in E_y. For instance, $E_6 = u_2, u_4, u_6, u_8 \prec E_8 = \{u_3, u_6, u_7, u_8\}$ because the index of the first node is $2 < 3$; and $E_7 = \{u_3, u_6, u_7\} \prec E_8 = \{u_3, u_6, u_7, u_8\}$ because the indices of the nodes from 0 to 3 are the same ($3 = 3$, $6 = 6$, $7 = 7$), and the index of the 4-th node is n/a < 8.

However, if we reorder the hyperedges as shown in Fig.3(b), we obtain the graph seen in Fig.3(c). As can be seen, the partial order of the hyperedges from left to right in this tree is $E_1 \prec E_3 \prec E_7 \prec E_8 \prec E_4 \prec \dots$. So in Fig.3(c), $E_8 \prec E_4$ but the index $8 > 4$.

Overall, the purpose of setting this partial order is to facilitate and improve the efficiency of constructing the prefix forest. We will provide more details on the definition and significance of this partial order in the revised version.



(a) the prefix forest (Figure 4) (b) reorder the nodes (Figure 6(a)) (c) the new prefix forests (Figure 6(b))

Figure 3: A toy example for the exploration order (Response to D9)

Comment 9. *D9: Figure 3: The illustration is inconsistent. The line out of E_7 labeled $Prec[E_7]$ ends in the set of nodes $\{u_3, u_6, u_7\}$ that is equal $Prec[E_7]$. However, the line out of E_6 labeled $Prec[E_6]$ ends in the set of nodes $\{u_2, u_4\}$ that is _not_ equal $Prec[E_6]$ but rather equal $E_6 - Prec[E_6]$. That is very confusing. Unnecessarily confusing.*

Response: Thank you for pointing out the inconsistency in Figure 3. You are correct in your understanding. In the original figure, the line originating from E_7 labeled $Prec[E_7]$ correctly ends at the set of nodes $\{u_3, u_6, u_7\}$. However, the line originating from E_6 labeled $Prec[E_6]$ incorrectly ends at the set of nodes $\{u_2, u_4\}$, which actually corresponds to $E_6 \setminus Prec[E_6]$.

We understand that this inconsistency can be confusing. The issue arose because we omitted a step in the figure, leading to the inconsistency between the two lines. We have updated the figure (**Section 4.2**) to correct this issue, ensuring that the illustration is now consistent and accurately reflects the relationships between the sets.

Comment 10. *D10: Alg 4, line 3: This eliminates hyper edges only including a single node. 1) The text does not explain the reason for that. Presumably, it is because such edges cannot form a triangle. 2) Why is that filter not applied in all algorithms? It is very basic and not specific to the use of forests.*

Response: Thank you for your questions regarding Algorithm 4, line 3. We understand your concerns and would like to provide the following explanations:

Reason for eliminating hyperedges with a single node: The primary reason for this step is to reduce the number of single-node trees in the prefix forest, which do not contribute to the formation of triangles. Such hyperedges are not essential for the algorithm's main purpose, which is to identify and count hyper-triangles.

Why this filter is not applied in all algorithms: The filter is not applied universally because other algorithms do not require the construction of a prefix forest. In contrast, Algorithm 4 specifically includes this step to improve the efficiency of prefix forest construction.

We have included an explanation in the revised version (**Section 4.3**).

Comment 11. *D11: Figure 7: Do the algorithms compared in this experiment produce the same result? It is not obvious that they do. If indeed they do not, what is the point of the comparison? This needs some more explanation.*

Response: Thank you for your question regarding the comparison in Figure 7. You are correct that the algorithms compared in this experiment may not produce the same results. The primary

purpose of the comparison is to evaluate the runtime efficiency of our proposed HTRUSS algorithm against other state-of-the-art (*SOTA*) methods for densely sub-hypergraph decomposition.

Comparison with *SOTA* Methods: The algorithms being compared, including TRUSS, nbHCORE, and kgHCORE, are all considered *SOTA* methods for dense sub-hypergraph decomposition. While they may not produce identical results, our goal is to evaluate the efficiency of our algorithm relative to these established methods.

General Baseline: The comparison serves as a general baseline to show how our algorithm performs in terms of runtime. As illustrated in Figure 7, our HTRUSS algorithm outperforms the core decomposition algorithms on all four datasets, even though these algorithms are not native and include additional constraints such as neighborhood-based constraints (nbHCORE) and g-distance influence constraints (kgHCORE).

Efficiency and Pruning: The figure also highlights that our algorithm is only slightly slower than TRUSS, indicating that our pruning strategy during hyper-triangle counting is highly effective. This is a significant advantage, as it shows that our algorithm can achieve better performance without the need for additional constraints.

We hope this explanation clarifies the purpose of the comparison in Figure 7 (**Section 5**). We have included this information in the revised version to provide a more comprehensive understanding of the results.

Comment 12. *D12: Sec 5.1, "Additionally, our algorithm is only slightly slower than TRUSS. ": In the Figure, there seems to be a different of something between 0.5 and 1 order of magnitudes. It is in the eye the beholder if that still qualifies as "only slightly slower".*

Response: Thank you for your comment regarding the phrasing in Section 5.1. You are correct that the difference in runtime between our algorithm and TRUSS is more than "only slightly slower." We understand that the difference is between 0.5 and 1 order of magnitude, which is significant.

We have revised the sentence to more accurately reflect the performance difference. The updated sentence reads: "Additionally, our algorithm's runtime is slower than TRUSS by 0.5 orders of magnitude. This is because the number of triangles in traditional graphs is much larger than in hypergraphs."

Comment 13. *D13: Sec 5.1: Are these experiments run in parallel or single threaded? What is the degree of parallelism in each of the compared algorithm.? The same or not? From the fact the Table 3 and 4 sac "(64x in parallel)" in their caption, I may guess that Figure 7 and Table 2 are single threaded. But I do not want to be left guessing. The paper should be explicit about this.*

Response: You are correct that Figure 7 is based on single-threaded implementations, as the algorithms being compared only provided single-threaded versions. In Table 2, the counting part is performed in parallel (64x), while other parts are executed in single-threaded mode.

Following your suggestion, in the revised version, we have explicitly indicated whether each experiment was run in single-threaded or parallel mode in the captions of the figures.

Comment 14. *D14: Table 2/3/4: What is the meaning of the numbers in bold? Commonly, bold indicates the highest (or lowest) number. Here it just seems to mark out one of the columns*

Response: Thank you for your question regarding the use of bold numbers in Tables 2, 3, and 4. We understand that bold text is commonly used to indicate the highest (or lowest) values.

In Table 3 and Table 4, the bold numbers indicate that HTC-PF and HTC-PF+ achieve the highest (or lowest, depending on the metric) values, confirming their stronger performance. For instance, in Table 3, the bold numbers show that these methods have significantly higher counts of hyper-triangles compared to other methods.

Regarding Table 2, we have revised the bold numbers to underlined numbers to avoid confusion. In Table 2, the underlined numbers in the TAU2, TMS1, TMS2, and TSO datasets indicate that $t_{counting}$ accounts for over 60% of t_{total} due to the higher degree of nodes in these datasets. This is consistent with the findings in Table 3, where the number of hyper-triangles in these datasets is significantly higher compared to other datasets.

Comment 15. *D15: Sec 5.3: Are the benefits described here an effect of using a hypergraph or of using hypergraph k-truss?*

Response: The benefits described are primarily due to using hypergraph k-truss. While hypergraphs themselves can represent complex relationships more accurately than traditional graphs, the hypergraph k-truss specifically captures the dense subgraph structures by considering the cohesiveness of hyperedges. This allows for a more precise identification of meaningful patterns and relationships within the data. For example, using hypergraph k-truss, we can distinguish between different collaborations such as E_1 and E_2, which represent different papers co-authored by Michael Stonebraker and Andrew Pavlo. In contrast, traditional graph representations would lose this distinction, leading to inaccuracies in the truss computation. Therefore, the benefits are a result of both the hypergraph representation and the k-truss decomposition, with the latter providing a more detailed and accurate analysis of the data.

Comment 16. *D16: Some work that appears related but is not mentioned is* `https://ieeexplore.ieee.org/document/10015766` *So, the statement "we are the first to study the problem of triangle counting in hypergraphs" is not perfectly accurate. Further, this would be also an interesting candidate to compare to. It samples, so it is likely not accurate. But what is the accuracy–runtime tradeoffs?*

Response: Indeed, our statement "we are the first to study the problem of triangle counting in hypergraphs" is not perfectly accurate. I am aware of the work by Lingling Zhang, Zhiwei Zhang, Guoren Wang, Ye Yuan, and Zhao Kang: Efficiently Counting Triangles for Hypergraph Streams by Reservoir-Based Sampling (IEEE Trans. Knowl. Data Eng. 35(11): 11328-11341, 2023), which was also conducted by colleagues from our team. However, their work differs from ours in that it focuses on counting triangles in hypergraph streams. Their key approach is to sample a probability p to represent the likelihood of forming a triangle and then estimate the number of triangles using h/p. In contrast, our truss decomposition requires enumerating specific triangles for peeling, so their method cannot be applied directly to truss decomposition.

Since the definition of hyper-triangles in the *TKDE* paper differs slightly from ours, and their sampling method cannot support truss decomposition, we will correct our inaccurate statement in the related work section and provide further clarification on these details.

Comment 17. *Minor issues*

- Sec 1, Definition of $N_G^V(v)$: should it say "$\{u, v\} \subset E_x$" instead of (u,v)?

- Sec 2, "In a traditionnal graph": typo -> "In a traditional graph"

- Sec 2, "the initial support calculation(i.e. the hyper-triangle counting [...]). ": missing space -> "the initial support calculation (i.e. the hyper-triangle counting [...]). " or rather without parentheses anyway -> "the initial support calculation, i.e. the hyper-triangle counting [...]."

- The paper has inconsistent capitalization of algorithm references: "Algorithm 3" vs. "algorithm 2". With a capital letter is preferrable. However, in any case it should be consistent.

- Sec 5.3, "so we get one summery of the frequent relationships": summery or summary? Presumably the latter.

- Sec 5.3, "Note that, we are hard to classify the groups which are highly over lapped": Something seems off here. You are hard to classify? (also "over lapped" -> "overlapped")

Response: Thank you for pointing out these minor issues. We have addressed them in the revised version and marked the changes in blue for your reference.

3 Response to Reviewer #5

Thank you for acknowledging this work during D1-D2. Below are our responses to D3-D5, which correspond to W1-W3.

Comment 1. *D3 The use case study on DBLP is not very clear. The Figure 12a shows a large network. Figure 12b is concise and tight – this is nice. However, how does one read its use – what*

do the unnamed connections with Prof. Stonebraker reflect? They are not conferences, right? How does this get interpreted for each researcher? In general, what would be a generic usecase and interpretation of such a k-truss?

Response: Thank you for your feedback on the DBLP case study. We understand that the use of the case study needs to be clearer, especially regarding the interpretation of the figures and the connections with Prof. Stonebraker. We have revised the case study to provide a more detailed explanation **(Section 5.3)**.

Comment 2. *D4 The figures do not show results over all datasets (Figures 8, 9, etc.). They have been omitted entirely. It would be nice to have them in a large technical report and provide under supplementary material for review.*

Response: We have added the missing figures to an online supplementary material for a more comprehensive presentation of the results across all datasets. Thank you for your feedback, and we hope this improves the clarity and completeness of our work.

Below, we include some relevant figures in our response. For more details, please refer to `https://github.com/VeryLargeGraph/HTRUSS/blob/main/TR/report.pdf`.

Table 1: Running of HTC-PF+ with different numbers of threads

dataset	1x	2x	4x	8x	16x	32x	64x
NS	0.821	0.428	0.214	0.113	0.072	0.042	0.029
CHS	0.911	0.453	0.222	0.118	0.074	0.045	0.033
TAU1	0.0086	0.0054	0.0031	0.0028	0.00026	0.0026	0.0024
TAU2	1200.12	621.64	311.79	158.38	79.53	39.12	20.78
TMS1	111.82	50.96	27.99	14.53	7.22	4.48	2.12
TMS2	4460.152	2479.968	1241.344	650.655	319.9566	164.32	87.49
CMG	10.88	5.32	2.78	1.33	0.67	0.33	0.17
CMH	0.37	0.19	0.11	0.05	0.031	0.015	0.012
DBLP	19.8	9.6	4.8	2.4	1.2	0.62	0.31
TSO	2667.97	1604.81	808.634	403.784	202.596	102.32	50.25

Comment 3. *D5 Results:*

- Exp-1: "As illustrated in the figure, our algorithm is highly efficient, even outperforming the core decomposition algorithms for hypergraphs on all four datasets."

– It does not seem to be the case. HTRUSS has about similar performance as nbHCORE. Why have other datasets not been shown?

- The significance of the small gains in Table 4 is not clear.

Response: Thank you for your comment regarding the results in the experiments.

You are correct that HTRUSS has about similar performance as nbHCORE. While truss decomposition methods are generally similar to core decomposition methods (which typically do not require triangle counting), HTRUSS has shown to be highly efficient, even slightly outperforming core decomposition algorithms for hypergraphs on the four datasets presented. It is worth noting that those methods produce different results, but both are considered state-of-the-art (*SOTA*) methods for dense sub-hypergraph decomposition. Our goal is to evaluate the efficiency of our algorithm relative to these established methods, even if they do not produce identical results.

Regarding the omission of results from other datasets, this was due to space limitations and the fact that differences on smaller datasets were not significant. We will include the results from all datasets in the supplementary material for a more comprehensive comparison.

The optimizations in Table 4, including reordering and work balance, indeed provide a relatively small performance boost of around 10%-20%. While these gains are not as significant as those achieved through the Prefix Forest optimization, they still contribute to the overall efficiency of the algorithm. Moreover, these optimizations are relatively simple to implement, making them a worthwhile addition to the algorithm.

Comment 4. *Minor:*
Pg. 3: traditionnal –> traditional
Pg. 12: summery –> summary

Response: We are grateful for pointing out these minor issues. We've made the corresponding corrections in the revised version and highlighted the changes in blue.