Truss Decomposition in Hypergraphs: Supplementary Material

1 THE HYPERGRAPH MODEL V.S. TRADITIONAL 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 in Fig. 1, 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.

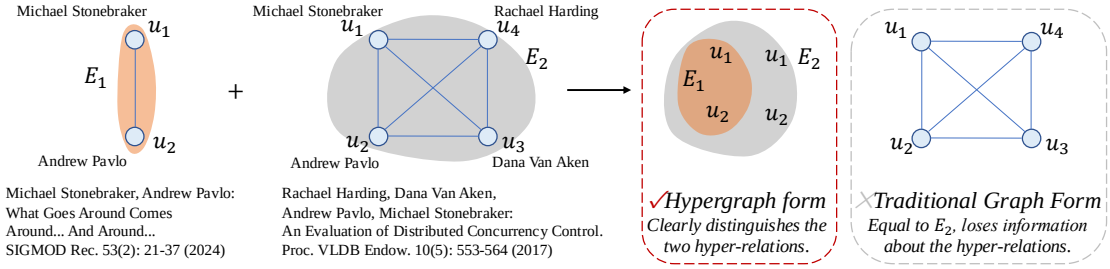


Fig. 1. The differences between the hypergraph model and traditional representations

2 DIFFERENT IMPLEMENTATIONS OF THE TRUSS MODEL

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.

Here is a comparison table of the three methods:

Author's address:

Table 1. Comparison of Truss-Based Methods

Aspect	Truss (Baseline)	Bitruss	Hypergraph Truss (WISE 2022)
Model	Bipartite graph (hyperedges as cliques)	Bipartite graph	Hypergraph
Core Definition	Traditional k-truss: edges in $\geq k-2$ triangles	Butterfly support $\geq$ k	Hyperedge triples: $E_x \cap E_y \cap E_z \geq \alpha$, $E_x \cup E_y \cup E_z \geq \beta$
Key Structure	Triangles (converted)	Butterfly (4-node)	Hyperedge intersections/unions
Strengths	Simple, graph-algorithm compatible	Efficient for bipartite graphs	Direct hypergraph analysis
Limitations	No true triangles, redundant	Bipartite-only	Rigid (α, β constraints)
Scalability	Graph-size limited	High (large bipartite)	Computationally heavy
Ref.	Baseline	Kai Wang et al. (ICDE 2020)	Xinzhou Wang et al. (WISE 2022)

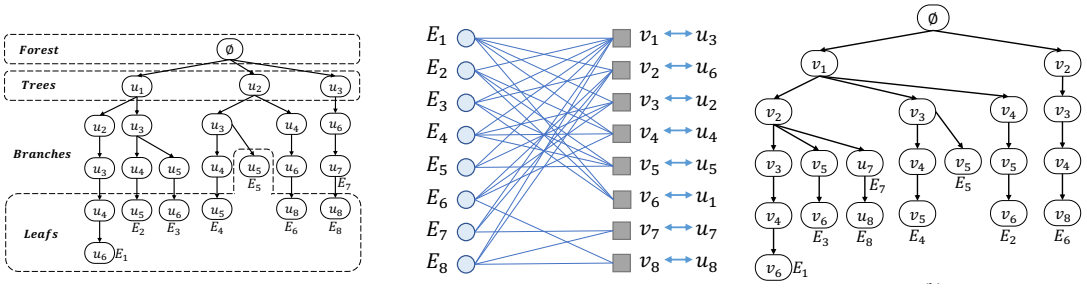
3 EXPLAINATION FOR THE EXPLORATION ORDER

In Algorithm 2, the exploration of triangles is indeed based on the constraint $z \prec y \prec x$. 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. 2(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.2(b), we obtain the graph seen in Fig.2(c). As can be seen, the partial order of the hyperedges from left to right in this tree is $E_1 \prec E_3 \prec E_7 \prec E_8 \prec E_4 \prec \ldots$. So in Fig.2(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))

Fig. 2. A toy example for the exploration order (Response to D9)

4 RUNNING OF HTC-PF+ WITH DIFFERENT NUMBERS OF THREADS

Table 2 below shows the performance of HTC-PF+ when executed with varying numbers of threads across multiple datasets. The number of threads was adjusted to 1, 2, 4, 8, 16, 32, and 64 to evaluate the parallelizability of the algorithm.

From the table, it is evident that as the number of threads increases, the runtime of HTC-PF+ decreases across all datasets. For instance, in the TMS2 dataset, the runtime reduces from 4460.152 seconds with 1 thread to 87.49 seconds

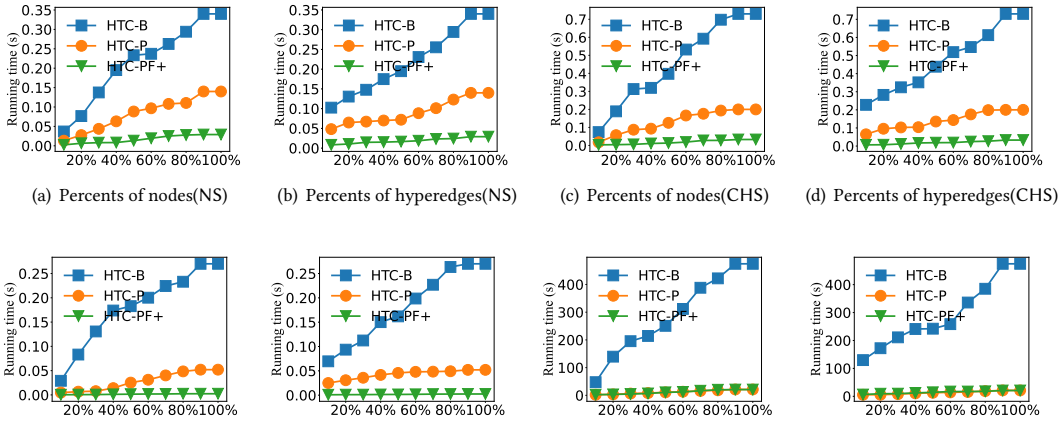
Table 2. Running of HTC-PF+ with different numbers of threads

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

with 64 threads. Similarly, in the DBLP dataset, the runtime decreases from 19.8 seconds with 1 thread to 0.31 seconds with 64 threads. This trend demonstrates the strong parallelizability of HTC-PF+. As the number of threads increases, the algorithm is able to distribute the computational workload more effectively, resulting in a significant reduction in runtime. This behavior is consistent across all datasets, confirming that HTC-PF+ is well-suited for parallel execution and can take full advantage of multi-threaded environments to improve performance.

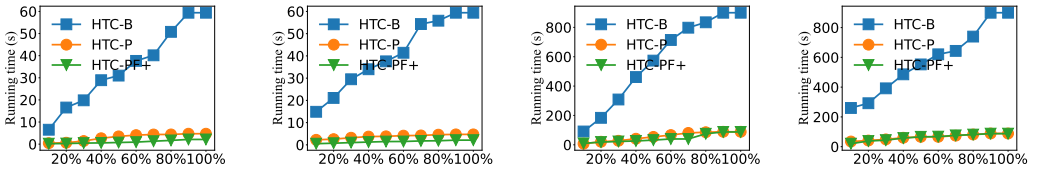
5 SCALABILITY OF THE PROPOSED ALGORITHMS(64X IN PARALLEL)

Fig. 3 shows the scalability testings with 64 threads in parallel of HTC-B, HTC-P and HTC-PF+ on all the dataset. We first generate ten hypergraphs by randomly picking 10%-100% of the nodes and the hyperedges, and evaluate the Runtimes of the proposed algorithms on those subgraphs. As shown in Fig. 3, the Runtime increases smoothly with increasing numbers of nodes or hyperedges. Also, we can see that HTC-PF+ is significantly faster than HTC-B and HTC-P with all datasets of different scale, which is consistent with our previous findings. These results suggest that our proposed algorithms are scalable when handling large hypergraphs.



(a) Percents of nodes(NS) (b) Percents of hyperedges(NS) (c) Percents of nodes(CHS) (d) Percents of hyperedges(CHS)

(e) Percents of nodes(TAU1) (f) Percents of hyperedges(TAU1) (g) Percents of nodes(TAU2) (h) Percents of hyperedges(TAU2)



(i) Percents of nodes(TMS1) (j) Percents of hyperedges(TMS1) (k) Percents of nodes(TMS2) (l) Percents of hyperedges(TMS2)

(m) Percents of nodes(CMG) (n) Percents of hyperedges(CMG) (o) Percents of nodes(CMH) (p) Percents of hyperedges(CMH)

(q) Percents of nodes(DBLP) (r) Percents of hyperedges(DBLP) (s) Percents of nodes(TSO) (t) Percents of hyperedges(TSO)

Fig. 3. Scalability of the proposed algorithms