On Improving the Cohesiveness of Graphs by Merging Nodes: Formulation, Analysis, and Algorithms

Fanchen Bu
School of Electrical Engineering, KAIST
Daejeon, South Korea
boqvezen97@kaist.ac.kr

Kijung Shin
Kim Jaechul Graduate School of AI, KAIST
Seoul, South Korea
kijungs@kaist.ac.kr

ABSTRACT

Graphs are a powerful mathematical model, and they are used to represent real-world structures in various fields. In many applications, real-world structures with high connectivity and robustness are preferable. For enhancing the connectivity and robustness of graphs, two operations, adding edges and anchoring nodes, have been extensively studied. However, merging nodes, which is a realistic operation in many scenarios (e.g., bus station reorganization, multiple team formation), has been overlooked. In this work, we study the problem of improving graph cohesiveness by merging nodes. First, we formulate the problem mathematically using the size of the k-truss, for a given k, as the objective. Then, we prove the NP-hardness and non-modularity of the problem. After that, we develop BATMAN, a fast and effective algorithm for choosing sets of nodes to be merged, based on our theoretical findings and empirical observations. Lastly, we demonstrate the superiority of BATMAN over several baselines, in terms of speed and effectiveness, through extensive experiments on fourteen real-world graphs.

CCS CONCEPTS

• **Mathematics of computing** → **Combinatorial optimization**; **Graph algorithms**; • **Information systems** → *Data mining*.

KEYWORDS

Graph algorithms, Combinatorial optimization, k-Trusses

ACM Reference Format:

Fanchen Bu and Kijung Shin. 2023. On Improving the Cohesiveness of Graphs by Merging Nodes: Formulation, Analysis, and Algorithms. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3580305.3599449

1 INTRODUCTION

As a powerful mathematical model, graphs have been widely used in various fields to represent real-world structures. Some typical applications of graphs are recommendation systems [62], social network analysis [59], and biological system analysis on molecular graphs [51] and protein-protein interactions [9]. Moreover, many optimization problems on real-world structures have been formulated as ones on the abstracted graphs.

KDD '23, August 6–10, 2023, Long Beach, CA, USA

ACM ISBN 979-8-4007-0103-0/23/08…$15.00
https://doi.org/10.1145/3580305.3599449

In many real-world applications, it is desirable to have a well-connected and robust structure. For example, in transportation systems, it is preferable that stations are connected tightly with each other so that traffic routes are resilient even if some accidents happen [32, 81]; in organizations like companies, often several interconnected projects or tasks are carried out at the same time, and thus several teams are supposed to form dense and highly-connected communities in an underlying graph so that the teams can closely collaborate with each other [2, 4, 29].

A straightforward operation to enhance the connectivity and robustness of graph structures is adding edges [6, 64]. Besides, anchoring nodes (i.e., forcefully including some nodes in a cohesive subgraph) [7, 36, 44, 72, 75, 76] has also been widely studied.

However, merging nodes, which is another realistic operation in many applications, has been overlooked. Merging nodes, or formally *vertex identification* [55], is the operation where we merge two nodes into one, and any other node adjacent to either of the two nodes will be adjacent to the "new" node. Merging nodes may strike you as too radical at first sight, but it is indeed a very realistic and helpful operation in several real-world examples such as:

(1) **Bus station reorganization.** Merging some nearby stations not only makes traffic networks more compact and systematic but also reduces maintenance expenses since the total number of stations is reduced [67]. For example, CTtransit, a bus-system company in the united states, proposed to merge multiple bus stations in New Haven and discussed the benefits [18].
(2) **Multiple Team formation.** Forming teams (i.e., "merging" individuals) within an organization can increase individual performance and cultivate a collaborative environment [15]. How to form well-performing and synergic teams is an important research topic [2, 4, 29] in business and management [35, 54].

In this paper, we study the problem of improving the connectivity and robustness of graphs by merging nodes. To the best of our knowledge, we are the first who study this problem. We propose to use the size (spec., the number of edges) of a k-truss [17] as the objective quantifying the connectivity and robustness. Given a graph G and an integer k, the k-truss of G is the maximal subgraph of G where each edge is in at least $k-2$ triangles; and we say that an edge has trussness k if the edge is in the k-truss but not the $(k+1)$-truss. Specifically, k-trusses have the following merits:

(1) **Cohesiveness.** k-Trusses require both engagements of the nodes and interrelatedness of the edges compared to some other cohesive subgraph models. Specifically, given any graph, a k-truss is always a subgraph of the $(k-1)$-core [60] but not vice

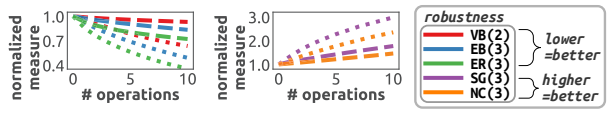


Figure 1: Merging nodes is much more effective than adding edges in enhancing graph robustness. For each robustness measure, we do 10 rounds of merging nodes (dotted) or adding edges (dashed). In each round, we greedily choose the node pair or edge that improves the measure most. In the legend, we include the minimum times (≤ 3 for all measures) of merging nodes that are needed to achieve a better improvement achieved by adding 10 edges. See Section 6.1 for the details.

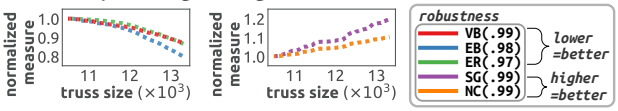


Figure 2: Maximizing the size of a k-truss is effective: graph robustness improves when we enlarge a k-truss. For each robustness measure, we report the relation between it and the truss size along the process of enlarging a k-truss by merging nodes using our proposed method BATMAN. We include the absolute value (0.97-0.99 for all measures) of Pearson's r in the legend. See Section 6.1 for the details.

versa, and each connected component of a k-truss is $(k-1)$-edge-connected [10, 33] with bounded diameter [30].[1]

(2) **Computational efficiency.** k-Trusses can be computed efficiently with time complexity $O(m^{1.5})$ [66], where m is the number of edges; in contrary, given a graph, enumerating all the cliques or many variants (n-cliques [48], k-plexes [61], n-clans and n-clubs [53]) is NP-hard.

(3) **Applicability.** k-Trusses, especially their sizes, ably capture connectivity and robustness of transportation [20], social networks [78, 85], communication [27], and recommendation [69]. Specifically, k-trusses also have realistic meanings in the two aforementioned real-world examples (bus station [19, 83] reorganization and multiple team formation [8, 21]).[2]

Due to the desirable theoretical properties and practical meaningfulness of k-trusses, several existing works [12, 13, 64, 73, 85] used the size of a k-truss as the objective.

Therefore, we consider the problem of maximizing the size of a k-truss in a given graph by merging nodes. In Figures 1 and 2, we show the effectiveness of merging nodes (spec., its superiority over adding edges) and maximizing the size of a k-truss (spec., the correlations between the truss size and various robustness measures), respectively (see Section 6.1 for more details). We mathematically formulate the problem as an optimization problem on graphs named **TIMBER** (**T**russ-s**I**ze **M**aximization **B**y m**ER**gers), and prove the NP-hardness and non-modularity of the problem.

For the TIMBER problem, we develop BATMAN (**B**est-merger se**A**rcher for **T**russ **MA**ximizatio**N**), a fast and effective algorithm equipped with **(1)** search-space pruning based on our theoretical analysis, and **(2)** simple yet powerful heuristics for choosing promising mergers. Starting from a computationally prohibitive naive greedy algorithm, we theoretically analyze the changes on a graph after mergers and use the findings to design speed-improving heuristics. For example, we prove that after merging two nodes, the trussness of an edge that is not incident to either of the merged nodes changes by at most one. Hence, we only need to consider the edges with original trussness at least $k-1$ for an input k. We first reduce the search space by **(1)** losslessly pruning the space of *outside nodes* (nodes that are not in the $(k-1)$-truss) using a maximal-set-based algorithm, **(2)** proposing and using a new heuristic to efficiently find promising *inside nodes* (nodes that are in the $(k-1)$-truss), and **(3)** excluding the mergers of two outside nodes with the rationality of doing so. Our fast and effective heuristics for finding promising pairs among the selected nodes are based on the number of edges with trussness $k-1$ gaining (and losing) support.

Through extensive experiments on 14 real-world graphs, we compare our proposed algorithm, BATMAN, to several baseline methods and show that BATMAN consistently performs best w.r.t the final increase in the size of k-trusses, achieving 1.38× to 10.08× performance superiority over the baseline methods on all the datasets.

In short, our contributions are four-fold:

(1) **A novel Problem:** We introduce and formulate TIMBER (Problem 1), a novel optimization problem on graphs with several potential real-world applications, as listed above.
(2) **Theoretical Analysis:** We prove the NP-hardness (Theorem 1) and non-submodularity (Theorem 2) of TIMBER.
(3) **A fast Algorithm:** We design BATMAN (Algorithm 4), a fast and effective algorithm for TIMBER, based on our theoretical (Lemmas 1-7) and empirical findings (Section 6.3). We also theoretically analyze the time complexity of BATMAN (Theorem 4).
(4) **Extensive Experiments:** We compare BATMAN with several baseline methods and demonstrate the advantages of BATMAN and its components using 14 real-world graphs (Section 6).

For **reproducibility**, the code and datasets are available at [1].

Table 1: Notations.

Notation	Definition
$G=(V,E)$	a graph with node set V and edge set E
$N(v;G)$	the set of neighbors of $v \in V$
$d(v;G)$	the degree of $v \in V$
$G[V']$	the induced subgraph of G on $V' \subseteq V$
$s(e;G)$	the support of $e \in E$
$T_k(G)$	the k-truss of G
$t(e;G)$, $t(v;G)$	the trussness of $e \in E$ and $v \in V$
$\hat{E}_k(G)$	the shell edges with trussness k, i.e., $E(T_{k-1}) \setminus E(T_k)$
$PM(v_1,v_2;G)$	the graph after merging v_1 and $v_2 \in V$ into v_1 in G
$\tilde{N}_k(v;G)$	the inside neighbors of $v \in V$, i.e., $N(v) \cap V(T_{k-1})$

2 RELATED WORK

k-Trusses. Based on the concept of k-cores [60], the concept of k-trusses is introduced in [17]. In [66], the authors propose an efficient truss decomposition algorithm with time complexity $O(m^{1.5})$, where m is the number of edges in the input graph. In [30], the authors use k-trusses to model the communities in graphs (see also [3]) and study the update of k-trusses in dynamical graphs (see also [49, 77]). Related problems are also studied for weighted graphs [80], signed graphs [78], directed graphs [47], uncertain graphs [31, 65], and simplicial complexes [56]. In [14], higher-order neighbors are considered to generalize the concept of k-trusses.

Graph structure enhancement and attacks. Several studies of graph structure enhancement or attacks are conducted based on cohesive subgraph models. Specifically, the problems of maximizing

[1] See Appendix F of [1] for detailed discussions on why the size of a k-truss is a better measure of cohesiveness and robustness than the size of a k-core.

[2] See Appendix C of [1] for more detailed real-world application scenarios.

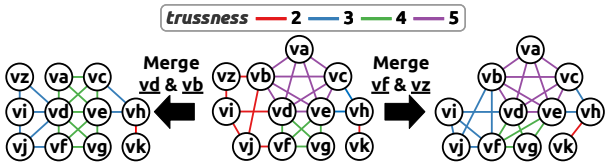


Figure 3: Two different ways of merging a pair of nodes in a graph with different consequences.

the size of a k-truss by anchoring nodes [72, 76] and by adding edges [12, 64] have been studied; and the opposite direction, i.e., minimizing the size of a k-truss, has also been considered [12, 13]. There are also a series of counterparts considering the model of k-cores [7, 36, 44–46, 52, 63, 73–76, 79, 82, 84] using the operations of adding (or deleting) edges (or nodes) and anchoring nodes. However, no existing work studies graph structure enhancement or attacks by merging nodes, while merging nodes is indeed a basic operation on graphs [55] and practically meaningful.

3 PRELIMINARIES

Let $G = (V, E)$ be an unweighted, undirected graph without self-loops, multiple edges, or isolated nodes. The set $N(v; G)$ of *neighbors* of a node v consists of the nodes adjacent to v, and the *degree* $d(v; G)$ of v in G is the number of neighbors of v. Given a subset $V' \subseteq V$ of nodes, the *induced subgraph* $G[V'] = (V', E')$ of G induced on V' is defined by $E' = \{e \in E : e \subseteq V'\}$, and we use $G \setminus V'$ to denote the graph $G[V \setminus V']$[3] obtained by removing the nodes in V' from G. The *support* $s(e; G)$ of an edge $e = (v_1, v_2)$ is the number of triangles that contain both v_1 and v_2.

Definition 1 (k-truss and trussness)**.** Given a graph $G = (V, E)$ and $k \in \mathbb{N}$,[4] the ***k-truss*** of G, denoted by $T_k = T_k(G)$, is the maximal subgraph of G where each edge in T_k has support at least $k - 2$ within T_k, i.e., $s(e; T_k) \geq k - 2, \forall e \in E(T_k)$.[5] We call the number $|E(T_k)|$ of edges in T_k its ***size***. The ***trussness*** $t(e; G)$ of an edge e (w.r.t G) is the largest k such that e is in $T_k(G)$, i.e., $t(e; G) = \max\{k \in \mathbb{N} : e \in E(T_k(G))\}$. The trussness $t(v; G)$ of a node v is the largest trussness among the trussness of all the edges containing (i.e. incident to) v, i.e., $t(v; G) = \max\{t(e; G) : v \in e\}$.

In this paper, merging two nodes in a graph means identifying the two nodes [55] into one node, as described in Definition 2.

Definition 2 (mergers)**.** Given a graph $G = (V, E)$ and two nodes $v_1, v_2 \in V$. If we merge v_1 and v_2 into v_1 in G, then the ***post-merger graph*** $PM(v_1, v_2; G) = (V', E')$ after the ***merger*** between v_1 and v_2 is defined by $V' = V \setminus \{v_2\}$ and E' derives from E by "shifting" the edges incident to v_2 to v_1 *without adding multiple edges or self-loops*, i.e., $E' = E \cup \{(v_1, u) : u \in N(v_2), u \neq v_1\} \setminus \{(v_2, u) : u \in N(v_2)\}$. We use $PM(P; G)$ to denote the post-merger graph when we merge multiple pairs in P in G (note that the order does not matter).

Recall the example in Figure 3. Let G_o denote the original graph in the middle, then the two post-merger graphs on the left and right are $PM(vd, vb; G)$ and $PM(vf, vz; G)$, respectively.

We summarize the notations in Table 1. In the notations, the input graph G can be omitted when the context is clear.

[3]We use $\setminus$ to denote the set subtraction operation.

[4]We use $\mathbb{N}$ to denote the set $\{1, 2, 3, \ldots\}$ of positive integers.

[5]k-Trusses are meaningful only when $k \geq 3$ since otherwise the k-truss is just the whole graph. In this paper, we assume that $k \geq 3$ without further clarification.

Algorithm 1: Naive greedy algorithm

Input : graph $G = (V, E)$; trussness k; budget b
Output: P: the pairs of nodes to be merged

```
P ← ∅
while |P| < b do
    f({p}) ← |E(T_k(PM(p; G)))|, ∀p ∈ (V' choose 2)
    p* ← arg max_p f({p}); P ← P ∪ {p*}
    if |P| < b then G = (V, E) ← PM(p*; G)
return P
```

4 PROBLEM STATEMENT AND HARDNESS

Problem 1. (TIMBER: Truss-sIze Maximization By mERgers)

- ***Given:*** *a graph $G = (V, E)$, $k \in \mathbb{N}$, and $b \in \mathbb{N}$,*
- ***Find:*** *a set P of up to b node mergers in G, i.e., $P \subseteq \binom{V}{2}$ and $|P| \leq b$,*
- ***to Maximize:*** *the size of the k-truss after the mergers,*[6] *i.e.,*

$$f(P) = f(P; G, k) = |E(T_k(PM(P; G)))|.$$

As mentioned before, for the example in Figure 3, merging vf and vz maximizes the size of the 3-truss, i.e., with the original graph in Figure 3, $k = 3$, and $b = 1$ as the inputs, $P = \{(vf, vz)\}$ is the solution that maximizes our objective function $f(P) = f(P; G, k)$.

Theorem 1. *The TIMBER problem is NP-hard for all $k \geq 3$.*[7]

Proof. **See Appendix A for all the proofs.** □

Theorem 2. *The function $f(P)$ is not submodular.*

Considering the NP-hardness and non-submodularity of the TIMBER problem, we aim to find a practicable and efficient heuristic.

5 METHODOLOGY

In this section, starting from the naive greedy algorithm, we first analyze the changes occurring when we merge a pair of nodes, and then based on our findings, we improve the computational efficiency while maintaining effectiveness as much as possible.

5.1 Naive greedy algorithm

First, we present the naive greedy algorithm in Algorithm 1. At each iteration, we merge each possible pair, compute the size of the k-truss after each merger, and find and operate the merger with the best performance. We repeat the above process until b mergers are selected. Although Algorithm 1 is algorithmically simple it suffers from prohibitive complexity, as shown in the following theorem.

Theorem 3. *Given an input graph $G = (V, E)$ and budget b, Algorithm 1 takes $O(b|V|^2|E|^{1.5})$ time and $O(|E|)$ space for any k.*

Remark 1. *In the time complexity, $|V|^2$ is from the space of all possible pairs and $|E|^{1.5}$ is from the truss decomposition algorithm.*

5.2 Theoretical analyses: changes after mergers

We shall show several theoretical findings regarding the changes occurring when we merge a pair of nodes. The following lemma shows that when we merge two nodes, the trussness of each edge containing neither of them changes (both increase and decrease are possible) by at most 1.

Lemma 1. *Given any G, v_1, and v_2, for any $e \in E(G)$, if $v_1, v_2 \notin e$, then $|t(e; PM(v_1, v_2)) - t(e; G)| \leq 1$.*

[6]The counterpart problem considering k-cores is technically similar to an existing problem considered in [7]. See Appendices D and F of [1] for details.

[7]That is, for all meaningful k values.

Note that (1) the trussness can both increase and decrease and (2) the above lemma does not apply to the edges incident to the merged nodes. After a merger, only (1) the edges in the original $(k-1)$-truss and (2) those between a node in the original $(k-1)$-truss and a merged node are *possibly* in the new k-truss.

Corollary 1. *Given any $G, k,$ and $v_1, v_2 \in V(G)$, $T_k(PM(v_1, v_2; G)) = T_k(G')$, where $V(G') = V(G)$ and $E(G') = E(T_{k-1}(G) \setminus \{v_1, v_2\}) \cup \{(v_1, x) : x \in (N(v_1) \cup N(v_2) \setminus \{v_1, v_2\}) \cap V(T_{k-1})\})$.*

The following lemma shows that each edge with trussness larger than that of any merged node cannot lose its trussness.

Lemma 2. *Given any G and $v_1, v_2 \in V(G)$, without loss of generality, we assume $t(v_1) \geq t(v_2)$. For any $e \in E(G)$, if $t(e) > t(v_2)$, then $t(e; PM(v_1, v_2)) \geq t(e; G)$.*

Notably, mergers between nodes with low trussness can result in an increase in trussness for edges with higher trussness. Lemma 3 shows a connection to k-cores.

Lemma 3. *Given any G, k, and $v_1, v_2 \in V(G)$, let N^* denote $N(v_1) \cup N(v_2) \setminus \{v_1, v_2\}$. For any $x \in N^*$, (v_1, x) is in $T'_k := T_k(PM(v_1, v_2))$ if and only if x is in the $(k-2)$-core of $T'_k[N^*]$.*

Based on the above analyses, we find it useful to consider the nodes *inside and outside* T_{k-1} separately and the neighbors *inside* T_{k-1} of a node need our special attention. Below, we formally define these concepts that will be frequently used throughout the paper.

Definition 3 (inside/outside nodes and inside neighbors)**.** Given a graph $G = (V, E)$ and $k \in \mathbb{N}$, we call a node $v \in V$ an **inside node** (w.r.t G and k) if $v \in V(T_{k-1})$ (i.e., $t(v) \geq k-1$) and we call v an **outside node** (w.r.t G and k) if $v \notin V(T_{k-1})$ (i.e., $t(v) < k-1$). Given any node u, the set of u's **inside neighbors** (w.r.t G and k) is defined as $\tilde{N}_k(u; G) = N(u; G) \cap V(T_{k-1})$.

Lemma 4 provides a simple way to compare the performance of two outside nodes w.r.t. the considered objective.

Lemma 4. *Given G and k, for any $u_1, u_2 \notin V(T_{k-1})$, if $\tilde{N}_k(u_1) \subseteq \tilde{N}_k(u_2)$, then $T_k(PM(v, u_1)) \subseteq T_k(PM(v, u_2)), \forall v \in V$; if further $\tilde{N}_k(u_1) = \tilde{N}_k(u_2)$, then $T_k(PM(v, u_1)) = T_k(PM(v, u_2)), \forall v \in V$.*

5.3 Reduce the number of pairs to consider

We shall first introduce several approaches to reduce the number of pairs to consider for a merger.

Maximal-set-based pruning for outside nodes. Lemma 4 shows that for any given outside node $u \notin V(T_{k-1})$, we do not need to consider u if there exists another outside node $u' \notin V(T_{k-1})$ with $N(u') \cap V(T_{k-1}) \supseteq N(u) \cap V(T_{k-1})$. Therefore, we only need to consider those nodes u with maximal set $\tilde{N}(u)$ of inside neighbors.

Lemma 5. *Given G and k, let $V_o = V(G) \setminus V(T_{k-1})$ denote the set of outside nodes, and let $\tilde{V}_o = \{u \in V_o : \nexists u' \in V \setminus V(T_{k-1})\ s.t.\ \tilde{N}(u') \supsetneq \tilde{N}(u)\}$ denote the set of outside nodes with a maximal set of inside neighbors. Then, $\max\{|E(T_k(PM(v_1, v_2)))| : v_1, v_2 \in V(G)\} = \max\{|E(T_k(PM(v_1, v_2)))| : v_1, v_2 \in V(T_{k-1}) \cup \tilde{V}_o\}$.*

Moreover, by Lemma 4, if several outside nodes have the same set of inside neighbors, only one of them needs to be considered. Finding maximal sets among a given collection of sets is a well-studied theoretical problem [70] with a number of fast algorithms. Based on [26], we present in Algorithm 5 (in Appendix B) a simple yet practical way to find the outside nodes with a maximal set of inside neighbors. Lemma 6 shows the correctness, time complexity, and space complexity of Algorithm 5.

Lemma 6. *Given the set of outside nodes V_o and the sets of their inside neighbors $\tilde{N}_k(v), \forall v \in V_o$, Algorithm 5 correctly finds the set of nodes $v' \in V_o$ with maximal $\tilde{N}_k(v')$ in $O(\sum_{v \in V_o} |\tilde{N}_k(v)||V_o|) = O(|V||E|)$ time and $O(|E|)$ space.*

In our implementation, among all the outside nodes with a maximal set of inside neighbors, we further sort the outside nodes by the number of inside neighbors and choose the ones with the most inside neighbors as the candidates.

A heuristic for finding promising inside nodes. Notably, our maximal-set-based pruning scheme does not apply to inside nodes, and thus we need different techniques for inside nodes. We propose and use a heuristic based on *incident prospects* (IPs) to evaluate the inside nodes and select the promising ones.

Definition 4 (incident prospects)**.** Given a graph $G = (V, E)$ and $k \in \mathbb{N}$, for each $v \in V$, the set of the **incident prospects** (IPs) of v is defined as $\tilde{N}_k(v) \setminus N(v; T_k)$.

Intuitively, the IPs of a node v correspond to the edges that are not in the current k-truss but possibly enter the new k-truss after a merger involving v (see Corollary 1). Therefore, if a node v has more IPs, then a merger involving v is preferable since it is more likely that the size of the k-truss will increase more because more edges incident to v may enter the new k-truss after the merger. Moreover, the number of the IPs of a node v is a lower bound of the number of inside neighbors of a node v, and thus if a node v has a larger number of IPs, then v also has a larger number of inside neighbors, i.e., more non-incident edges may benefit from the merger. See Section 6.3 for the empirical support of the proposed heuristic, including the comparison of multiple heuristics.

In our implementation, we sort the inside nodes by the number of IPs of each inside node and choose the ones with the most IPs as our candidate inside nodes.

Exclude outside-outside mergers. After dividing nodes into inside nodes and outside nodes, we now have three types of mergers: (1) *inside-inside mergers* (IIMs) where two inside nodes are merged, (2) *outside-outside mergers* (OOMs) where two outside nodes are merged, and (3) *inside-outside mergers* (IOMs) where one inside node and one outside node are merged. We shall show that OOMs are less desirable than the other two types in general. Merging two nodes v_1 and v_2 can equivalently be seen as (1) removing all edges incident to v_2 and (2) adding each "new" edge (v_1, x) for $x \in N(v_2) \setminus N(v_1) \setminus \{v_1\}$. Proposition 7 shows that if we do not include an inside node in the merger (i.e., for an OOM), then each single "new" edge cannot increase the size of T_k.

Lemma 7. *Given any $G = (V, E)$, k, and $v_1, v_2 \notin T_{k-1}$, for any $x \in N(v_1) \cup N(v_2) \setminus \{v_1, v_2\}$, $T_k(G) = T_k(G')$, where $V(G') = V(G)$ and $E(G') = E(G) \cup \{(v_1, x)\}$.*

See also Section 6.3 for the empirical evidence supporting our choice. Therefore, from now on we assume that we always include at least one inside node in the merger. Then there are two cases that we need to consider: IOMs and IIMs, and no one is necessarily better than the other.

Algorithm 2: Find IOM candidates

Input : pruned outside nodes V_o^*; inside nodes V_i; inside neighbors $\tilde{N}_k(v), \forall v \in V_o^* \cup V_i$; shell edges $\hat{E}_k$; k-truss T_k; number of inside nodes to check n_i; number of outside nodes to check n_o; number of pairs to choose n_c

Output : C_{IOM}: the chosen IOM candidates

1 $\hat{V}_i \leftarrow$ the n_i inside nodes v_i in V_i with most incident prospects
2 $\hat{V}_o \leftarrow$ the n_o outside nodes v_o in V_o^* with most inside neighbors
3 **for** $v_i \in \hat{V}_i$ **do**
4 $\quad H(t_i) \leftarrow \tilde{N}_k(t_i) \cup \tilde{N}_k(v_i), \forall t_i \in V_i \setminus \tilde{N}_k(v_i) \setminus \{v_i\}$
5 $\quad$ **for** $v_o \in \hat{V}_o$ **do**
6 $\quad\quad Z = Z(v_i, v_o) \leftarrow (\tilde{N}_k(v_i) \cup \tilde{N}_k(v_o)) \setminus (N(v_i; T_k) \cup \{v_i\})$
7 $\quad\quad H_i \leftarrow \bigcup_{z \in Z} H(z)$
8 $\quad\quad H_n \leftarrow \{(x, y) \in \hat{E}_k : (x \in Z \vee y \in Z) \wedge x \in Z \cup \tilde{N}_k(v_i) \wedge y \in Z \cup \tilde{N}_k(v_i)\}$
9 $\quad\quad \hat{H}_k(v_i, v_o) \leftarrow H_i \cup H_n$
10 $C_{IOM} \leftarrow$ the n_c IOMs $(v_i, v_o) \in \hat{V}_i \times \hat{V}_o$ with largest $|\hat{H}_k(v_i, v_o)|$ (tie broken by $|Z(v_i, v_o)|$)
11 **return** C_{IOM}

5.4 Promising pairs among promising nodes

Even with the above analyses, it is still computationally expensive to compute the size of the new k-truss after each possible merger, even when the number of candidate nodes is relatively small. For example, in the *youtube* dataset (to be introduced in Section 6) with $k = 10$, the total number of possible IOMs and IIMs is 6.6 billion. Although after pruning the outside nodes, the number is reduced to 374 million. Therefore, it is still imperative to further reduce the number of times that we check the actual size of the k-truss. To this end, we shall propose and use some heuristics to efficiently find promising mergers (IOMs and IIMs). For both cases, our algorithmic framework is in the following form:

(1) We first find the *promising nodes* as described above.
(2) Among all the possible pairs between the promising nodes, we use novel heuristics to find a small number of *promising pairs*.
(3) We check the increase in the size of the k-truss for each of the promising pairs and merge a pair with the greatest increase.
(4) We repeat the above process until we exhaust the budget.

Inside-outside mergers. By Corollary 1, we know that an IOM between an inside node v_1 and an outside node v_2, w.r.t. the size of the k-truss, brings "new" edges (v_1, z) for each "new" neighbor $z \in (\tilde{N}_k(v_2) \cup \tilde{N}_k(v_1)) \setminus (N(v_1, T_{k-1}) \cup \{v_1\})$ into the current $(k-1)$-truss. Note that $\tilde{N}_k(v_1) \supseteq N(v_1, T_{k-1})$ may hold since an edge between two nodes in the $(k-1)$-truss may exist in the original graph but not in the $(k-1)$-truss.

To efficiently evaluate the candidate IOMs, we propose to use the concept of *potentially helped shell edges* (PHSEs). For given G and k, we use $\hat{E}_k(G)$ to denote $\hat{E}_k(G) = E(T_{k-1}) \setminus E(T_k)$ (the edges with trussness $k-1$) and call such edges *shell edges* (w.r.t G and k).

Definition 5 (potentially helped shell edges)**.** Given a graph $G = (V, E)$, $k \in \mathbb{N}$, an inside node v_1, and an outside node v_2, the set of the **potentially helped shell edges** (PHSEs) w.r.t the IOM between v_1 and v_2, denoted by $\hat{H}_k(v_1, v_2; G)$, consists of the shell edges $(x, y) \in \hat{E}_k(G)$ such that at least a triangle containing (x, y) is newly formed because of the "new" edges brought into T_{k-1} by the IOM. Formally, $\hat{H}_k(v_1, v_2; G) = \{e \in \hat{E}_k(G) : s(e; G \cup \{(v_1, z) : z \in Z\}) > s(e; G)\}$, where $Z = (\tilde{N}_k(v_2) \cup \tilde{N}_k(v_1)) \setminus (N(v_1, T_{k-1}) \cup \{v_1\})$.

In the above definition, we require that (v_1, z) and (x, y) are in the same triangle, thus we have $x \in \{v_1, z\}$ or $y \in \{v_1, z\}$. Accordingly, there are two ways in which some shell edges (x, y) can be helped: **(1)** the IOM brings a "new" neighbor z to v_1 and thus forms a new triangle $\triangle_{v_1 z z'}$ for some z' that is adjacent to v_1 in the original graph, and **(2)** the IOM brings two "new" neighbors z_1 and z_2 and thus forms a new triangle $\triangle_{v_1 z_1 z_2}$. The first case (1) further includes two sub-cases: **(1a)** some shell edge (v_1, z') incident to v_1 is helped, and **(1b)** some shell edge (z, z') not incident to v_1 is helped. In Figure 4(a), we provide an illustrative example.

We present the whole heuristic-based procedure for choosing IOM candidates in Algorithm 2. Among the inputs of Algorithm 2, V_o^*, V_i, $\tilde{N}_k$, $\hat{E}_k$, and T_k are computed from the inherent inputs G and k of the TIMBER problem, while n_i, n_o, and n_c are set by the user to control the computational cost.

We first choose the most promising inside nodes and outside nodes using some heuristics as presented in Section 5.3 (Lines 1-2). After choosing the promising nodes, for each chosen inside node v_i, we first compute the incident PHSEs that each "new" neighbor may bring (Lines 3-4). Then, for each outside node v_o, we compute the "new" neighbors the IOM between v_i and v_o brings to v_i (Line 6), collect all the incident PHSEs of the "new" neighbors (Line 7), compute the non-incident PHSEs (Line 8), and take the union to get all the PHSEs (Line 9). Finally, we use the computed PHSEs to select the most promising IOMs (Line 10). See Section 6.3 for the empirical support of the proposed heuristics.

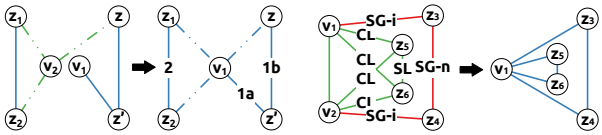
Lemma 8. *Given pruned outside nodes V_o^*, inside nodes V_i, inside neighbors $\tilde{N}_k$, shell edges $\hat{E}_k$, and k-truss T_k, Algorithm 2 takes $O(|V_o^*| \log n_o + n_i n_o (|V_i| + |\hat{E}_k| + \log n_c))$ time to find n_c IOM candidates from n_i and n_o promising inside and outside nodes, respectively.*

Inside-inside mergers. Consider an IIM between two nodes v_1 and v_2. In Figure 4(b), we provide an example of an IIM between v_1 and v_2. An IIM may incur three kinds of changes that may affect the size of the k-truss. The first kind is **support gains (SGs)**, which are also caused by IOMs. For IIMs, SGs further include two sub-cases:

- **SG-n (Support gains of non-incident edges).** It may happen for an edge between a node adjacent to v_1 but not to v_2 and another node adjacent to v_2 but not to v_1. In Figure 4(b), (z_3, z_4) gains one support after the IIM between v_1 and v_2.
- **SG-i (Support gains of incident edges).** Incident edges are the edges incident to either of the merged nodes. In Figure 4(b), both (v_1, z_3) and (v_2, z_4) gain one support after the IIM.

The latter two kinds can only be caused by IIMs but not by IOMs:

- **CL (Collisions).** IIMs can directly make some edges collide and disappear. Specifically, each pair of edges (v_1, x) and (v_2, x) incident to the same node x and the two merged nodes collide and only one of them remains. In Figure 4(b), there are collisions between (z_5, v_1) and (z_5, v_2); and between (z_6, v_1) and (z_6, v_2).
- **SL (Support losses).** IIMs can reduce the support of some edges in the current $(k-1)$-truss, potentially decreasing their trussness. Specifically, each edge between the common neighbors of v_1 and v_2 loses a common neighbor after the merger between v_1 and v_2. In Figure 4(b), the edge (z_5, z_6) loses one support after the IIM.



(a) an example of an IOM (b) an example of an IIM

Figure 4: Illustrative examples of the changes caused by an IOM (left) or an IIM (right) between v_1 and v_2.

Algorithm 3: Find IIM candidates

Input : inside nodes V_i; inside neighbors $\tilde{N}_k(v), \forall v \in V_i$; shell edges $\hat{E}_k$; k-truss T_k; number of inside nodes to check n_i; number of pairs to choose n_c

Output : C_{IIM}: the chosen IIM candidates

1 $\hat{V}_i \leftarrow$ the n_i inside nodes v_i in V_i with most incident prospects

2 **for** $(v_1, v_2) \in \binom{\hat{V}_i}{2}$ **do**

3 $\quad h(v_1, v_2) \leftarrow -|\{u \in V(T_k) : \{(v_1, u), (v_2, u)\} \subseteq E(T_k)\}|$

4 $\quad$ **for** $(x, y) \in \hat{E}_k$ *with* $x, y \notin \{v_1, v_2\}$ *and* $\{x, y\} \subseteq \tilde{N}_k(v_1) \cup \tilde{N}_k(v_2)$ **do**

5 $\quad\quad$ **if** $x, y \notin \tilde{N}_k(v_1) \cap \tilde{N}_k(v_2)$ **then**

6 $\quad\quad\quad h(v_1, v_2) \leftarrow h(v_1, v_2) + 1$

7 $\quad\quad$ **else if** $\{x, y\} \subseteq \tilde{N}_k(v_1) \cap \tilde{N}_k(v_2)$ **then**

8 $\quad\quad\quad h(v_1, v_2) \leftarrow h(v_1, v_2) - 1$

9 $C_{IIM} \leftarrow$ the n_c IIMs $(v_1, v_2) \in \binom{\hat{V}_i}{2}$ with largest $h(v_1, v_2)$

10 **return** C_{IIM}

Due to the new types of changes that we need to consider, there are several noticeable points that we shall discuss below.

Lemma 4 tells us that for IOMs, outside nodes with large neighborhoods are generally preferable, while including inside nodes with large neighborhoods does not always give better performance. One of the reasons is that including inside nodes with larger neighborhoods may cause more collisions and support losses described above.

For IOMs, we have used the number of all potentially helped shell edges (PHSEs, Definition 5) to find the candidate IOMs (Lines 7 to 9 in Algorithm 2). Specifically, we consider both incident PHSEs (Line 7) and non-incident PHSEs (Line 8). However, there are two subtleties: **(a)** for IIMs, the computation of incident PHSEs becomes tricky due to the collisions mentioned above; **(b)** moreover, we also need to additionally take the support losses into consideration. To address the two subtleties, we slightly modified the heuristic we have used for IOMs. Regarding subtlety (a), since the incident shell edges have been considered when we choose the inside nodes w.r.t the incident prospects (IPs), for simplicity, we only consider the immediate collisions among the edges in the current k-truss without computing the support gains and support losses of the incident shell edges. Regarding subtlety (b), we count both the shell edges with support gains and those with support losses. To conclude, for each shell edge with support gains, we give +1 score (reward) to the corresponding IIM; for each shell edge with support losses and each collision between two edges in the current k-truss, we give -1 score (penalty). See Section 6.3 for the empirical comparisons of different heuristics in choosing candidate IIMs.

In Algorithm 3, we present the whole procedure for choosing candidate IIMs. Among the inputs, V_i, $\tilde{N}_k$, $\hat{E}_k$, and T_k are computed from the inherent inputs G and k of the TIMBER problem, while n_i and n_c are set by the user to control the computational cost.

We first choose the most promising inside nodes and outside nodes using the heuristics mentioned in Section 5.3 (Lines 1). After that, for each pair (v_1, v_2) between two chosen inside nodes, we compute its score using the heuristic described above. Specifically, for each pair, we first initialize the score by giving -1 score to each collision between two edges in the current k-truss (Line 3), then for each non-incident shell edge (x, y) whose support changes (Line 4), add +1 score for each one whose support increases (Line 6), and give -1 score for each one whose support decreases (Line 8). Finally, we use the computed scores to select the most promising IIMs (Line 9).

Lemma 9. *Given inside nodes V_i, inside neighbors $\tilde{N}_k$, shell edges $\hat{E}_k$, and the k-truss T_k, Algorithm 3 takes $O(|V_i| \log n_i + n_i^2(|\hat{E}_k| + \log n_c))$ time to find n_c IIM candidates from n_i promising inside nodes.*

5.5 Considering both IIMs and IOMs

Theoretically, merging IOMs is not always better than IIMs, and vice versa. Indeed, as empirically shown in Section 6.2, neither IOMs nor IIMs can be consistently superior to the other. In general, when k is small, IIMs are more desirable, while IOMs gain strength when k increases. Intuitively, when k increases, the k-truss is denser, and thus IIMs inevitably cause more collisions and support losses due to the high overlaps among the neighborhoods of the inside nodes. Therefore, it is necessary to consider both IOMs and IIMs.

We propose a strategy to take both IIMs and IOMs into consideration without wasting too much computation on the less-promising case. The key idea is to *adaptively distribute* the number of candidates between the two cases. Specifically, we fix the total number of pairs to choose in each round (i.e., the sum of n_c's for Algorithms 2 and 3) and divide it into two parts for IIMs and IOMs. Initially, the number is equally divided. Then in each round, we shift $1/b$ fraction of the total number, *to* the case where the best-performing pair in this round belongs *from* the other case. We make sure that the n_c for each case does not decrease to zero. See Section 6.2 for the empirical support for considering both IIMs and IOMs and the adaptive distribution of the number of candidates. The pseudo-code of the process mentioned above is given in Algorithm 4 (see Lines 15 to 18), which will be described in detail in Section 5.7.

5.6 Check the result after each merger

By proposing techniques to reduce the search space and proposing heuristics to find promising pairs efficiently, we have been addressing the problem of the $O(|V|^2)$ space of all possible pairs. Another overhead (see Remark 1) is the truss decomposition which takes $O(|E|^{1.5})$ time.

For checking the size of the k-truss after each possible merger between two nodes v_1 and v_2, we do not need to compute it from the whole post-merger graph. We use Corollary 1 by which the computation takes only $O(|E(T_{k-1})|^{1.5})$ time since $|\{(v_1, x) : x \in (N(v_1) \cup N(v_2) \setminus \{v_1, v_2\}) \cap V(T_{k-1})\}| = O(|E(T_{k-1})|)$.

Remark 2. *It is theoretically possible to use incremental algorithms for updating k-trusses after edge additions and removals [30, 49]. However, their efficiency is limited in our case since even a single node merger can cause a large number of edge additions and removals.*

Algorithm 4: BATMAN: final proposed algorithm

Input : graph $G = (V, E)$; trussness k; budget b; number of inside nodes to check n_i; number of outside nodes to check n_o; number of pairs to choose n_c

Output : P: the pairs of nodes to be merged

1 $P \leftarrow \emptyset$; $n_{io} \leftarrow \lfloor n_c/2 \rfloor$
2 **while** $|P| < b$ **do**
3 compute or update $t(e)$ using truss decomposition
4 $\hat{E}_k \leftarrow \{e \in E : t(e) = k-1\}$
5 $t(v) \leftarrow \max_{e \ni v} t(e), \forall v \in V$
6 $V_i \leftarrow \{v \in V : t(v) \geq k-1\}$; $V_o \leftarrow \{V\} \setminus V_i$
7 $\tilde{N}_k(v) \leftarrow N(v) \cap V(T_{k-1(G)}), \forall v \in V$
8 $V_o^* \leftarrow$ Alg. 5 with inputs V_o and $\tilde{N}_k$
9 $C_{IOM} \leftarrow$ Alg. 2 with inputs $V_o^*, V_i, \tilde{N}_k, \hat{E}_k, T_k(G), n_i, n_o, n_{io}$
10 $C_{IIM} \leftarrow$ Alg. 3 w/ inputs $V_i, \tilde{N}_k, \hat{E}_k, T_k(G), n_i, n_c - n_{io}$
11 $p^* \leftarrow \arg\max_{c=(v_1,v_2) \in C_{IOM} \cup C_{IIM}} T_k(PM(v_1, v_2))$
▷ Corollary 1 is used for simplification
12 $P \leftarrow P \cup \{p^*\}$
13 **if** $|P| < b$ **then**
14 $G = (V, E) \leftarrow PM(p^*; G)$
15 **if** $p^* \in C_{IOM}$ **then**
16 $n_{io} \leftarrow \min(n_{io} + \lfloor n_c/b \rfloor, \lceil n_c(b-1)/b \rceil)$
17 **else**
18 $n_{io} \leftarrow \max(n_{io} - \lfloor n_c/b \rfloor, \lfloor n_c/b \rfloor)$
19 **return** P

5.7 Overall algorithm (BATMAN)

In Algorithm 4, we present the procedure of the proposed algorithm BATMAN (**B**est-merger se**A**rcher for **T**russ **MA**ximizatio**N**). The inputs are the inherent inputs of the TIMBER problem (an input graph G, trussness k, and a budget b) and the parameters that control the computational cost (n_i, n_o, and n_c).

In each round, we first compute or update the edge trussness (Line 3), and prepare the information that we need later (Lines 4 to 7). Then we use Algorithm 5 to prune the set of outside nodes using the maximal-set-based technique (Line 8). After that, we use Algorithms 2 and 3 to obtain the candidate IOMs and IIMs, respectively (Lines 9 and 10). Then we check the performance of all the candidate mergers and find the best one (Line 11), and update the graph together with its edge trussness accordingly if not all budget has been exhausted (Line 13). Regarding the distribution of the number of pairs to check in each round, initially the number is equally distributed between IOMs and IIMs (Line 1), and in each round we increase the number of the case where the best-performing pair belongs and decrease that of the other case (Lines 15 to 18). We make sure that both cases are considered throughout the process.

Theorem 4. *Given an input graph G, trussness k, a budget b, and the parameters n_i, n_o, and n_c, Algorithm 4 takes $O(b(|E|^{1.5} + n_c|E(T_{k-1})|^{1.5} + |V_o^*| \log n_o + n_i n_o(|V_i| + |\hat{E}_k| + \log n_c) + n_i^2(|\hat{E}_k| + \log n_c)))$ time and $O(|E| + n_c)$ space to find b pairs to be merged.*[8]

6 EXPERIMENTAL EVALUATION

In this section, through extensive experiments on fourteen real-world graphs, we shall answer each of the following questions:

- **Q1**: how effective are merging nodes and maximizing the size of a k-truss in enhancing graph robustness?
- **Q2**: how effective and computationally efficient is BATMAN in maximizing the size of a k-truss by merging nodes?
- **Q3**: how effective is each algorithmic choice in BATMAN?

Experimental settings. For each dataset, we conduct experiments for each $k \in \{5, 10, 15, 20\}$. We use $b = 10$, check 100 inside nodes and 50 outside nodes ($n_i = 100$, $n_o = 50$ in Algorithm 4), and the number of pairs to check in each round (n_c in Algorithm 4) is set to 10 by default. We conduct all the experiments on a machine with i9-10900K CPU and 64GB RAM. All algorithms are implemented in C++, and complied by G++ with O3 optimization.

Datasets. In Appendix B, we report some statistics (the number of nodes/edges, max trussness, and sizes of k-trusses for different k values) of the real-world graphs used for the experiments.[9]

6.1 Q1: Effectiveness of merging nodes and truss-size maximization

We shall first show that merging nodes is an effective operation to enhance graph robustness. Then, we show that when we maximize the size of a k-truss, we effectively improve graph robustness.

Effectiveness of merging nodes. First, we show that merging nodes is an effective way to enhance cohesiveness and robustness, specifically, compared to adding edges. We consider different robustness measures [22, 25]: **(1)** VB (average vertex betweenness), **(2)** EB (average edge betweenness), **(3)** ER (effective resistance) [23, 28], **(4)** SG (spectral gap) [50], and **(5)** NC (natural connectivity) [11]. On the Erdös-Rényi model [24] with $n = 50$ and $p = 0.1$,[10] for each measure, we use greedy search to find the mergers or new edges that improve the measure most. In Figure 1, we report the change of the measure in each setting when we merge nodes or add edges 10 times, where merging nodes is much more effective. Mean values over five independent trials are reported.

Effectiveness of enlarging a k-truss. Second, we conduct a case study on the *email* dataset. In Figure 2, we show how the five robustness measures mentioned above change along with the truss size, when we apply our proposed algorithm BATMAN on the *email* dataset to maximize the size of its 10-truss by 100 mergers. The measures are linearly normalized so that all the original values correspond to 1. The chosen mergers increase the robustness even though BATMAN only directly aims to increase the size of a k-truss, showing that maximizing the size of a k-truss is indeed a reasonable way to reinforce graph cohesiveness and robustness.[11]

6.2 Q2: Effectiveness & efficiency of BATMAN

We shall compare BATMAN with several baseline methods, showing BATMAN's high effectiveness and high efficiency.

Considered algorithms. Since the TIMBER problem is formulated for the first time by us, no existing algorithms are directly available. Therefore, we use several intuitive baseline methods as the competitors and also compare several variants of the proposed algorithm. For all algorithms, the maximal-set-based pruning for outside nodes described in Section 5.3 is always used. In each round,

[8] See Appendix G.3 of [1] for more discussions on the time complexity.

[9] See Appendix E of [1] for additional experiments on real-world bus station datasets, where we consider distance constraints and the proposed method still outperforms the baseline methods overall.

[10] See Appendix I of [1] for the results using other random graph models.

[11] Considering the size of a k-core is less reasonable. See Appendix F of [1].

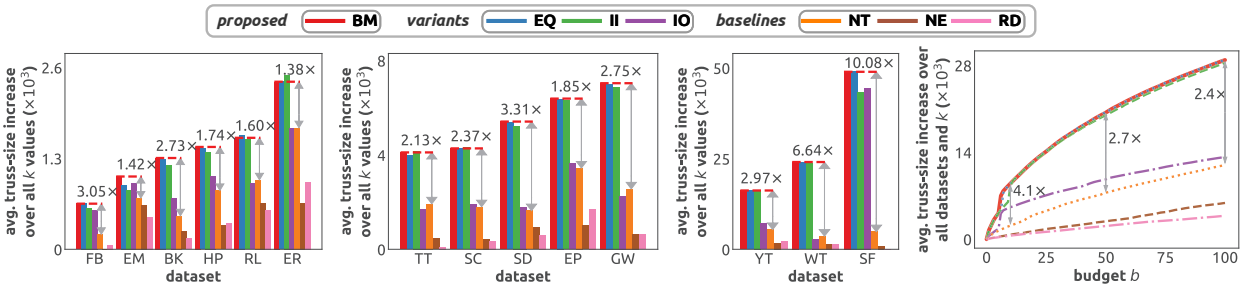


Figure 5: The first three subfigures on the left: The average increase in the truss size of each considered algorithm on each dataset over all the considered k values. The proposed algorithm (BM), with its variants (EQ, II, and IO) consistently outperforms the baseline methods (NT, NE, and RD) by 1.38× to 10.08×. Overall, BM performs better than its variants, showing the usefulness of our algorithmic designs. The rightmost subfigure: The results with each algorithm with different budget b. The proposed method (BM) and its variants consistently outperform the baseline methods by 2.4× to 4.1×.

all the algorithms find candidate mergers and operate the best one after checking all the candidates. The considered algorithms are:

- **RD (Random)**: uniform random sampling among all the IIMs and IOMs. Average performances over five trials are reported.
- **NE (Most new edges)**: among all the IOMs,[12] choosing the ones that increase the number of *edges* among the nodes in the current $(k-1)$-truss most.
- **NT (Most new triangles)**: among all the IIMs and IOMs, choosing ones that increase the number of *triangles* consisting of the nodes in the current $(k-1)$-truss most.
- **BM (BATMAN)**: the proposed method (Algorithm 4).
- **EQ (BATMAN-EQ)**: a BATMAN variant always **equally distributing** the number of pairs to check between IIMs and IOMs.
- **II (BATMAN-II)**: a BATMAN variant considering **IIMs** only.
- **IO (BATMAN-IO)**: a BATMAN variant considering **IOMs** only.

Evaluation metric. We evaluate the **performance** of each algorithm by the increase in the size of the k-truss, i.e., we measure $|E(T_k(PM(P;G)))| - |E(T_k(G))|$, for each output P for a graph G.

Results on each dataset. In Figure 5 (the first 3 subfigures), for each dataset, we report the average performance over all $k \in \{5, 10, 15, 20\}$ of each algorithm. The proposed algorithm BATMAN with its variants consistently outperforms the baseline methods, and the overall performance of BATMAN is better than that of its variants. Specifically, compared to the best baseline method on each dataset, BATMAN gives 1.38× to 10.08× better performance, and the ratio is above 2× on 9 out of 14 datasets. Overall, BATMAN performs better than its variants that consider only IIMs or IOMs, or always equally distribute the number of candidate mergers to check. This shows the usefulness of considering both IIMs and IOMs and adaptive distribution of the number of candidate mergers.

Results on different budgets b. In Figure 5 (the 4th subfigure), for each $1 \le b \le 100$, we report the average performance of each algorithm over all datasets and all k values. As shown in the results, BATMAN clearly outperforms the baseline methods regardless of b values. When $b = 10$, 50, and 100, BATMAN performs 4.1×, 2.7×, and 2.4× better than the best baseline method, respectively.

Results on different # candidates. In Figure 6 (the first 3 subfigures), for each algorithm checking different numbers (1, 2, 5, 10, 15, 20) of candidates in each round, we report the average running time and the average performance over all datasets and all k values. The proposed algorithm BATMAN clearly outperforms the baseline methods, even when the baseline methods check more candidate mergers than BATMAN in each round; BATMAN is also more effective and more stable than its variants, especially when we check a larger number of candidate pairs. Also, for different algorithms (except NT), the running time is similar when we check the same number of candidate pairs in each round, validating the theoretical analyses on the time complexity of BATMAN.

Results on different k. We divide the considered k values into two groups: *low* (5/10) and *high* (15/20), and compare the performance of the algorithms in each group. In Figure 6 (the 4th subfigure), for each group, we report the average increase in the truss size of each algorithm over all the datasets and over the two k values in the group. Again, BATMAN consistently outperforms the baseline methods, regardless of the k value. Notably, when k is low, IIMs perform much better than IOMs w.r.t the increase in the truss size; but when k is high, this superiority is decreased, even reversed, and thus considering both IIMs and IOMs shows higher necessity.

The above results show from different perspectives that BATMAN overall outperforms the baselines as well as its variants. The **full results** in each considered setting (datasets and parameters) are in Appendix H of [1].

Case Study. In Appendix I of [1], we provide a case study on the *relato* dataset showing which nodes (companies) are merged together by BATMAN. We observe, e.g., that in many cases, a giant company gets merged with a large number of companies in various fields, which is also a common case in the real world [37].

6.3 Q3: Effectiveness of the algorithmic choices

We shall show several results that empirically support our three algorithmic choices: (1) excluding outside-outside mergers, and (2 & 3) the heuristics for choosing promising inside nodes and mergers.

The considered **heuristics for choosing inside nodes** are:

- **IP (Most inside prospects)**: choosing the inside nodes with most inside prospects (Definition 4);
- **IN (Most inside neighbors)**: choosing the inside nodes with most inside neighbors (Definition 3);
- **RD (Random)**: sampling inside nodes uniformly at random. We report the average performance over three independent trials.

The considered **heuristics for choosing IOMs** are:

- **SE (Most potentially helped shell edges)**: choosing the mergers with most potentially helped shell edges (Definition 5);

[12] Note that IIMs cannot increase the number of such edges.

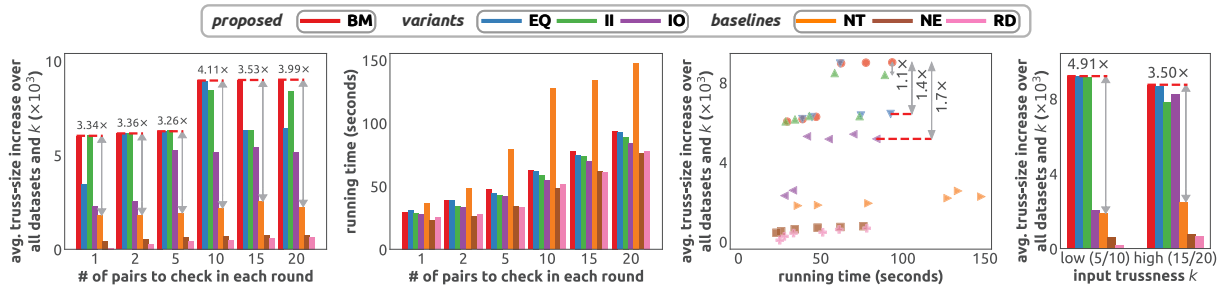


Figure 6: The first three subfigures on the left: The average performance and running time of each considered algorithm over all datasets and k values when the number of candidate pairs checked in each round varies. The proposed algorithm (BM) and its variants (EQ, II, and IO) clearly outperform the baseline methods (NT, NE, and RD). BM outperforms the baseline methods even when the baseline methods check more candidates; and BM is more effective and more stable than its variants, especially when we check more candidates (see Appendix J of [1] for the tests for statistical significance of the superiority of the proposed method over the variants). The rightmost subfigure: The average performance of each algorithm overall datasets when the trussness k varies. The proposed method (BM) and its variants (EQ, II, and IO) outperform the baseline methods (NT, NE, and RD) for both low and high k values.

Table 2: Empirical support of our algorithmic choices: excluding outside-outside (OO) mergers and the proposed heuristics (IP and SE). The results are averaged over all the datasets. See Appendix H of [1] for the results on each individual dataset.

(a) Justification of excluding outside-outside mergers.

Type	Perf.	#
II*	408.9	$10^{8.76}$
IO*	269.1	$10^{9.81}$
OO	152.1	$10^{11.39}$

* used in BATMAN

(b) Justification of using the heuristic IP to choose inside nodes.

Heur.	Performance IOM	IIM
IP*	562.3	1496.3
IN	547.9	1407.9
RD	397.9	326.3

* used in BATMAN

(c) Justification of using the heuristic SE to choose mergers.

Heur.	Performance IOM	IIM
SE*	524.1	1445.5
NN/AE	203.9	1348.9
RD	182.8	339.7
BE	562.3	1496.3

* used in BATMAN

- **NN (Most new neighbors)**: choosing the mergers that bring most "new" neighbors to the inside node in the mergers;
- **RD (Random)**: sampling mergers uniformly at random. [13]

The considered **heuristics for choosing IIMs** are:

- **SE**[14] **(Scoring using shell edges)**: choosing the mergers with highest scores that are described in Section 5.4,[15]
- **AE (Scoring using all edges in the $(k-1)$-truss)**: choosing the mergers that with highest scores that are measured as in SE but based on all edges in the $(k-1)$-truss instead of shell edges;
- **RD (Random)**: sampling mergers uniformly at random.[13]

We provide summarized results in Table 2 and full results in Appendix H of [1]. We summarize the results in the table as follows:

- In Table 2(a), we show the best performance (Perf.) among 10000 random inside-inside (II) / inside-outside (IO) / outside-outside (OO) mergers, and the total number of mergers (#) of each case. Compared to IIMs and IOMs, the number of OOMs is much higher, while their performance is much lower, which justifies excluding them in BATMAN.
- In Table 2(b), we show the best performance among all the IOMs / IIMs using the 100 inside nodes chosen by each heuristic. Overall, the heuristic used in BATMAN for choosing inside nodes, IP, outperforms the competitors.
- In Table 2(c), we show the best performance among all the IOMs / IIMs using the 10 outside nodes chosen by each heuristic and the 100 inside nodes chosen by IP. Overall, the heuristic used in BATMAN for choosing mergers, SE, outperforms the competitors, achieving a performance close to the best possible (BE) results achievable using the 100 inside nodes chosen by IP.

7 CONCLUSION

In this work, motivated by real-world scenarios and applications, we formulated and studied the problem of improving the connectivity and robustness of graphs by merging nodes (Problem 1), for which we used the number of edges in the k-truss for some given k as the objective. Then, we proved the NP-hardness and non-submodularity of the problem (Theorems 1 and 2). For the problem, based on our theoretical findings regarding mergers between nodes and k-trusses (Lemmas 1-5), we proposed BATMAN (Algorithm 4), a fast and effective algorithm equipped with strong search-space-pruning schemes (Algorithms 2-3 and 5) and analyzed its time and space complexity (Theorem 4). Through experiments on real-world graphs, we demonstrated the superiority of BATMAN over several baselines and the effectiveness of every component of BATMAN (Figures 5-6). For reproducibility, we made the code and datasets publicly available at [1]. We plan to consider this problem on weighted/uncertain graphs and explore other cohesive models.

Acknowledgements. This work was supported by National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2020R1C1C1008296) and Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2022-0-00871, Development of AI Autonomy and Knowledge Enhancement for AI Agent Collaboration) (No. 2019-0-00075, Artificial Intelligence Graduate School Program (KAIST)).

[13] We report the average performance over three independent trials.

[14] The SE for IOMs can be seen as a special case of the SE for IIMs since the -1 scores are only possible for IIMs, and thus we use the same abbreviation for both cases.

[15] +1 / -1 for each non-incident shell edge with support gains / losses; also -1 for each collision between two edges in the current k-truss;

REFERENCES

[1] 2023. *Online supplementary material.* https://github.com/bokveizen/cohesive-truss-merge

[2] Ramesh Bobby Addanki et al. 2020. Multi-team Formation using Community Based Approach in Real-World Networks. *arXiv preprint arXiv:2008.11191* (2020).

[3] Esra Akbas and Peixiang Zhao. 2017. Truss-based community search: a truss-equivalence based indexing approach. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1298–1309.

[4] Vivek Singh Baghel and S Durga Bhavani. 2018. Multiple team formation using an evolutionary approach. In *IC3*.

[5] Richard Barrett, Michael Berry, Tony F Chan, James Demmel, June Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk Van der Vorst. 1994. *Templates for the solution of linear systems: building blocks for iterative methods*. SIAM.

[6] Alina Beygelzimer, Geoffrey Grinstein, Ralph Linsker, and Irina Rish. 2005. Improving network robustness by edge modification. *Physica A: Statistical Mechanics and its Applications* 357, 3-4 (2005), 593–612.

[7] Kshipra Bhawalkar, Jon Kleinberg, Kevin Lewi, Tim Roughgarden, and Aneesh Sharma. 2015. Preventing unraveling in social networks: the anchored k-core problem. *SIAM Journal on Discrete Mathematics* 29, 3 (2015), 1452–1475.

[8] Edward C Brewer and Terence L Holmes. 2016. Better communication= better teams: A communication exercise to improve team performance. *IEEE Transactions on Professional Communication* 59, 3 (2016), 288–298.

[9] Sylvain Brohee and Jacques Van Helden. 2006. Evaluation of clustering algorithms for protein-protein interaction networks. *BMC bioinformatics* 7, 1 (2006), 1–19.

[10] Guo-Ray Cai and Yu-Geng Sun. 1989. The minimum augmentation of any graph to a K-edge-connected graph. *Networks* 19, 1 (1989), 151–172.

[11] Hau Chan, Leman Akoglu, and Hanghang Tong. 2014. Make it or break it: Manipulating robustness in large networks. In *SDM*.

[12] Chen Chen, Mengqi Zhang, Renjie Sun, Xiaoyang Wang, Weijie Zhu, and Xun Wang. 2022. Locating pivotal connections: The K-Truss minimization and maximization problems. *World Wide Web* 25, 2 (2022), 899–926.

[13] Huiping Chen, Alessio Conte, Roberto Grossi, Grigorios Loukides, Solon P Pissis, and Michelle Sweering. 2021. On breaking truss-based communities. In *KDD*.

[14] Zi Chen, Long Yuan, Li Han, and Zhengping Qian. 2021. Higher-Order Truss Decomposition in Graphs. *IEEE Transactions on Knowledge and Data Engineering* (2021).

[15] Meenal Chhabra, Sanmay Das, and Boleslaw Szymanski. 2013. Team formation in social networks. *Computer and information sciences III* (2013), 291–299.

[16] Eunjoon Cho, Seth A Myers, and Jure Leskovec. 2011. Friendship and mobility: user movement in location-based social networks. In *KDD*.

[17] Jonathan Cohen. 2008. Trusses: Cohesive subgraphs for social network analysis. *National security agency technical report* (2008).

[18] CTtransit. 2010. https://www.cttransit.com/sites/default/files/PDF_files/Bus%20Stop%20Conslidation_NH.pdf

[19] Sybil Derrible and Christopher Kennedy. 2010. The complexity and robustness of metro networks. *Physica A: Statistical Mechanics and its Applications* 389, 17 (2010), 3678–3691.

[20] Issa Moussa Diop, Chantal Cherifi, Cherif Diallo, and Hocine Cherifi. 2020. On local and global components of the air transportation network. In *Conference on Complex Systems (CCS)*.

[21] Nurcan Durak, Ali Pinar, Tamara G Kolda, and C Seshadhri. 2012. Degree relations of triangles in real-world networks and graph models. In *CIKM*.

[22] Wendy Ellens and Robert E Kooij. 2013. Graph measures and network robustness. *arXiv preprint arXiv:1311.5064* (2013).

[23] Wendy Ellens, Floske M Spieksma, Piet Van Mieghem, Almerima Jamakovic, and Robert E Kooij. 2011. Effective graph resistance. *Linear algebra and its applications* 435, 10 (2011), 2491–2506.

[24] Paul Erdős, Alfréd Rényi, et al. 1960. On the evolution of random graphs. *Publication of the Mathematical Institute of the Hungarian Academy of Sciences* 5, 1 (1960), 17–60.

[25] Scott Freitas, Diyi Yang, Srijan Kumar, Hanghang Tong, and Duen Horng Chau. 2021. Evaluating graph vulnerability and robustness using tiger. In *CIKM*.

[26] Gene. 2013. *Efficient algorithm for finding all maximal subsets.* https://stackoverflow.com/questions/14106121

[27] Zakariya Ghalmane, Mohammed El Hassouni, Chantal Cherifi, and Hocine Cherifi. 2018. K-truss decomposition for modular centrality. In *International Symposium on Signal, Image, Video and Communications (ISIVC)*.

[28] Arpita Ghosh, Stephen Boyd, and Amin Saberi. 2008. Minimizing effective resistance of a graph. *SIAM review* 50, 1 (2008), 37–66.

[29] Jimmy H Gutiérrez, César A Astudillo, Pablo Ballesteros-Pérez, Daniel Mora-Melià, and Alfredo Candia-Véjar. 2016. The multiple team formation problem using sociometry. *Computers & Operations Research* 75 (2016), 150–162.

[30] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying k-truss community in large and dynamic graphs. In *SIGMOD*.

[31] Xin Huang, Wei Lu, and Laks VS Lakshmanan. 2016. Truss decomposition of probabilistic graphs: Semantics and algorithms. In *SIGMOD*.

[32] Jian Gang Jin, Loon Ching Tang, Lijun Sun, and Der-Horng Lee. 2014. Enhancing metro network resilience via localized integration with bus services. *Transportation Research Part E: Logistics and Transportation Review* 63 (2014), 17–30.

[33] Camille Jordan. 1869. Sur les assemblages de lignes. *Journal für die reine und angewandte Mathematik* 71 (1869), 185–190.

[34] Russell Jurney. 2013. *Efficient algorithm for finding all maximal subsets.* https://data.world/datasyndrome/relato-business-graph-database

[35] Steve WJ Kozlowski and Bradford S Bell. 2013. Work groups and teams in organizations. (2013).

[36] Ricky Laishram, Ahmet Erdem Sar, Tina Eliassi-Rad, Ali Pinar, and Sucheta Soundarajan. 2020. Residual core maximization: An efficient algorithm for maximizing the size of the k-core. In *SDM*.

[37] Naomi R Lamoreaux. 1988. *The great merger movement in American business, 1895-1904.* Cambridge University Press.

[38] Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. 2010. Predicting positive and negative links in online social networks. In *TheWebConf (fka. WWW)*.

[39] Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. 2010. Signed networks in social media. In *CHI*.

[40] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. In *KDD*.

[41] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2007. Graph evolution: Densification and shrinking diameters. *The ACM Transactions on Knowledge Discovery from Data* 1, 1 (2007), 2–es.

[42] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. 2009. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics* 6, 1 (2009), 29–123.

[43] Jure Leskovec and Julian Mcauley. 2012. Learning to discover social circles in ego networks. In *NeurIPS (fka. NIPS)*.

[44] Qingyuan Linghu, Fan Zhang, Xuemin Lin, Wenjie Zhang, and Ying Zhang. 2020. Global reinforcement of social networks: The anchored coreness problem. In *SIGMOD*.

[45] Qingyuan Linghu, Fan Zhang, Xuemin Lin, Wenjie Zhang, and Ying Zhang. 2022. Anchored coreness: efficient reinforcement of social networks. *The VLDB Journal* 31, 2 (2022), 227–252.

[46] Kaixin Liu, Sibo Wang, Yong Zhang, and Chunxiao Xing. 2021. An Efficient Algorithm for the Anchored k-Core Budget Minimization Problem. In *ICDE*.

[47] Qing Liu, Minjun Zhao, Xin Huang, Jianliang Xu, and Yunjun Gao. 2020. Truss-based community search over large directed graphs. In *SIGMOD*.

[48] R Duncan Luce. 1950. Connectivity and generalized cliques in sociometric group structure. *Psychometrika* 15, 2 (1950), 169–190.

[49] Qi Luo, Dongxiao Yu, Xiuzhen Cheng, Zhipeng Cai, Jiguo Yu, and Weifeng Lv. 2020. Batch processing for truss maintenance in large dynamic graphs. *IEEE Transactions on Computational Social Systems* 7, 6 (2020), 1435–1446.

[50] Fragkiskos D Malliaros, Vasileios Megalooikonomou, and Christos Faloutsos. 2012. Fast robustness estimation in large social graphs: Communities and anomaly detection. In *SDM*.

[51] David E Manolopoulos and Patrick W Fowler. 1992. Molecular graphs, point groups, and fullerenes. *The Journal of chemical physics* 96, 10 (1992), 7603–7614.

[52] Sourav Medya, Tianyi Ma, Arlei Silva, and Ambuj Singh. 2020. A Game Theoretic Approach For Core Resilience. In *IJCAI*.

[53] Robert J Mokken et al. 1979. Cliques, clubs and clans. *Quality & Quantity* 13, 2 (1979), 161–173.

[54] Richard L Moreland, Linda Argote, and Ranjani Krishnan. 2002. Training people to work in groups. In *Theory and research on small groups*. Springer, 37–60.

[55] James G Oxley. 2006. *Matroid theory*.

[56] Giulia Preti, Gianmarco De Francisci Morales, and Francesco Bonchi. 2021. STruD: Truss Decomposition of Simplicial Complexes. In *TheWebConf (fka. WWW)*.

[57] Matthew Richardson, Rakesh Agrawal, and Pedro Domingos. 2003. Trust management for the semantic web. In *ISWC*.

[58] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*. https://networkrepository.com

[59] John Scott. 1988. Social network analysis. *Sociology* 22, 1 (1988), 109–127.

[60] Stephen B Seidman. 1983. Network structure and minimum degree. *Social networks* 5, 3 (1983), 269–287.

[61] Stephen B Seidman and Brian L Foster. 1978. A graph-theoretic generalization of the clique concept. *Journal of Mathematical sociology* 6, 1 (1978), 139–154.

[62] Nitai B Silva, Ren Tsang, George DC Cavalcanti, and Jyh Tsang. 2010. A graph-based friend recommendation system using genetic algorithm. In *CEC*.

[63] Xin Sun, Xin Huang, and Di Jin. 2022. Fast algorithms for core maximization on large graphs. *Proceedings of the VLDB Endowment* 15, 7 (2022), 1350–1362.

[64] Xin Sun, Xin Huang, Zitan Sun, and Di Jin. 2021. Budget-constrained Truss Maximization over Large Graphs: A Component-based Approach. In *CIKM*.

[65] Zitan Sun, Xin Huang, Jianliang Xu, and Francesco Bonchi. 2021. Efficient probabilistic truss indexing on uncertain graphs. In *TheWebConf (fka. WWW)*.

[66] Jia Wang and James Cheng. 2012. Truss decomposition in massive networks. *Proceedings of the VLDB Endowment* 5, 9 (2012), 812–823.

[67] Sheng Wei, Lei Wang, Xiongwu Fu, and Tao Jia. 2020. Using open big data to build and analyze urban bus network models within and across administrations.

Complexity (2020).

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

[69] Zhibang Yang, Xiaoxue Li, Xu Zhang, Wensheng Luo, and Kenli Li. 2022. K-truss community most favorites query based on top-t. *World Wide Web* 25, 2 (2022), 949–969.

[70] Daniel M Yellin. 1992. Algorithms for subset testing and finding maximal sets. In *SODA*.

[71] Hao Yin, Austin R Benson, Jure Leskovec, and David F Gleich. 2017. Local higher-order graph clustering. In *KDD*.

[72] Fan Zhang, Conggai Li, Ying Zhang, Lu Qin, and Wenjie Zhang. 2018. Finding critical users in social communities: The collapsed core and truss problems. *IEEE Transactions on Knowledge and Data Engineering* 32, 1 (2018), 78–91.

[73] Fan Zhang, Conggai Li, Ying Zhang, Lu Qin, and Wenjie Zhang. 2020. Finding Critical Users in Social Communities: The Collapsed Core and Truss Problems. *IEEE Transactions on Knowledge and Data Engineering* 32, 1 (2020), 78–91.

[74] Fan Zhang, Wenjie Zhang, Ying Zhang, Lu Qin, and Xuemin Lin. 2017. OLAK: an efficient algorithm to prevent unraveling in social networks. *Proceedings of the VLDB Endowment* 10, 6 (2017), 649–660.

[75] Fan Zhang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2017. Finding critical users for social network engagement: The collapsed k-core problem. In *AAAI*.

[76] Fan Zhang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2018. Efficiently reinforcing social networks over user engagement and tie strength. In *ICDE*.

[77] Yikai Zhang and Jeffrey Xu Yu. 2019. Unboundedness and efficiency of truss maintenance in evolving graphs. In *SIGMOD*.

[78] Jun Zhao, Renjie Sun, Qiuyu Zhu, Xiaoyang Wang, and Chen Chen. 2020. Community identification in signed networks: a k-truss based model. In *CIKM*.

[79] Kangfei Zhao, Zhiwei Zhang, Yu Rong, Jeffrey Xu Yu, and Junzhou Huang. 2021. Finding critical users in social communities via graph convolutions. *IEEE Transactions on Knowledge and Data Engineering* 35, 1 (2021), 456–468.

[80] Zibin Zheng, Fanghua Ye, Rong-Hua Li, Guohui Ling, and Tan Jin. 2017. Finding weighted k-truss communities in large networks. *Information Sciences* 417 (2017), 344–360.

[81] Yaoming Zhou, Junwei Wang, and Hai Yang. 2019. Resilience of transportation systems: concepts and comprehensive review. *IEEE Transactions on Intelligent Transportation Systems]* 20, 12 (2019), 4262–4276.

[82] Zhongxin Zhou, Wenchao Zhang, Fan Zhang, Deming Chu, and Binghao Li. 2021. VEK: a vertex-oriented approach for edge k-core problem. In *TheWebConf (fka. WWW)*.

[83] Difeng Zhu, Guojiang Shen, Jingjing Chen, Wenfeng Zhou, and Xiangjie Kong. 2022. A higher-order motif-based spatiotemporal graph imputation approach for transportation networks. *Wireless Communications and Mobile Computing* 2022 (2022), 1–16.

[84] Weijie Zhu, Chen Chen, Xiaoyang Wang, and Xuemin Lin. 2018. K-core minimization: An edge manipulation approach. In *CIKM*.

[85] Weijie Zhu, Mengqi Zhang, Chen Chen, Xiaoyang Wang, Fan Zhang, and Xuemin Lin. 2019. Pivotal Relationship Identification: The K-Truss Minimization Problem.. In *IJCAI*.

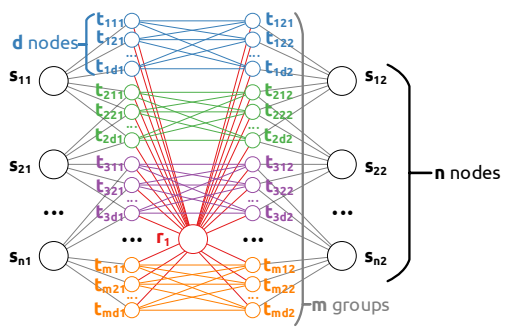


Figure 7: The constructed instance of the TIMBER problem corresponding to the maximum cover problem with $k = 4$, where $S_1 = \{t_1, t_2\}$, $S_2 = \{t_2, t_3\}$, and $S_n = \{t_3, t_m\}$.

Due to the space limit, additional supplementary materials are available online [1].

A PROOFS

In this section, we provide proofs for the theoretical claims in the main text.

A.1 Proof of Theorem 1

Proof of Theorem 1. We show the NP-hardness by reducing the NP-hard maximum coverage (MC) problem to the TIMBER problem. Consider the MC problem with the collection of n sets $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ and budget b. Let $T = \{t_1, t_2, \ldots, t_m\} = \bigcup_{i=1}^{n} S_i$. Consider the decision version where we shall answer whether there is a subset $\mathcal{S}' \subseteq \mathcal{S}$ with $|\mathcal{S}'| \leq b$ such that at least X elements in T are covered by $\mathcal{S}'$. We shall construct a corresponding instance of the TIMBER problem. We construct the graph G as follows. For each $t_j \in T$, we create $2d$ nodes t_{jp1} and $t_{jp2}, \forall 1 \leq p \leq d$, where d is sufficiently large ($d > 10kmn$), and add edges $(t_{jp1}, t_{jp'2})$ for all $p \neq p'$. For each $S_i \in \mathcal{S}$, we create two nodes s_{i1} and s_{i2}, and for each $t_j \in S_i$, we add edges (s_{i1}, t_{jp1}) and $(s_{i2}, t_{jp2}), \forall 1 \leq p \leq d$. Fix any $k \geq 3$, we create $k-3$ nodes $r_1, r_2, \ldots, r_{k-3}$, each of which is connected with all t-nodes (i.e., t_{jp1} and $t_{jp2}, \forall j, p$). See Figure 7 for an example of the construction. We also consider the decision version of the TIMBER problem where we shall answer whether there is a set P' of pairs of nodes with $|P'| \leq b$ such that $f(P') \geq Xd^2$.

$\Rightarrow$) Given a YES-instance $\mathcal{S}' = \{S_{i_1}, S_{i_2}, \ldots, S_{i_{b'}}\}$ with $|\mathcal{S}'| = b' \leq b$ for the MC problem, we claim that $P' = \{(s_{i_1 1}, s_{i_1 2})\}_{i=1}^{b'}$ is a YES-instance P' for the TIMBER problem. By our construction and $|\bigcup_{S' \in \mathcal{S}'} S'| \geq X$, merging all pairs in P' makes all the edges among the at least X corresponding groups of t-nodes enter the k-truss, and the total number is at least Xd^2.

$\Leftarrow$) Given a YES-instance P' with $|P'| = b' \leq b$ for the TIMBER problem, we claim that (1) those edges entering the k-truss are distributed in at least X groups of t-nodes corresponding to the elements in T, and (2) there exists $P'' \subseteq \{(s_{i1}, s_{i2})\}_{i=1}^{n}$ with $|P''| = b'$ that is also a YES-instance of the TIMBER problem. For (1), assume the opposite, i.e., less than X groups are involved, then the size of the new k-truss is at most $(X-1)d^2 + 2(k-3)md + 2mnd < Xd^2$, which contradicts the fact that P' is a YES-instance. For (2), it is easy to see that each non-(s_{i1}, s_{i2})-type pair can be replaced an (s_{i1}, s_{i2})-type pair without decreasing the size of the k-truss. Hence we can replace each element in P' by an (s_{i1}, s_{i2})-type pair without decreasing the number of groups of edges among t-nodes entering the k-truss. So we can find $P'' \subseteq \{(s_{i1}, s_{i2})\}_{i=1}^{n}$ with $|P''| = b'$ and $f(P'') \geq Xd^2$, completing the proof. □

A.2 Proof of Theorem 2

Proof of Theorem 2. Consider the example in Figure 7, but with $k = 5$ (there are r_1 and r_2 connected to all t-nodes). Let $X = \{(s_{11}, s_{12})\}$, $Y = \{(s_{11}, s_{12}), (s_{21}, s_{22})\} \supset X$, and $x = (s_{n1}, s_{n2})$, $f(X \cup \{x\}) - f(X) = 0 < f(Y \cup \{x\}) - f(Y)$, completing the proof. □

A.3 Proof of Theorem 3

Proof of Theorem 3. Truss decomposition algorithm takes $O(|E|^{1.5})$ time and $O(|V| + |E|)$ space [66]. Since we only consider connected graphs, $|E| = O(|V|)$ and thus $O(|V|+|E|) = O(|E|)$. Computing the size of the k-truss after each merger takes $O(|E|^1.5)$ time. Because there are $O(|V|^2)$ pairs and b iterations, the total time complexity is $O(b|V|^2|E|^{1.5})$. The space complexity is determined by that of storing the graphs and truss decomposition, which is $O(|E|)$. □

A.4 Proof of Lemma 1

Proof of Lemma 1. Let G' denote $PM(v_1, v_2; G)$. First, we show the decrease is limited. For each k, for each edge in the current k-truss, merging a pair of nodes can decrease the support by at most 1. Therefore, each current k-truss at least satisfies the condition of $(k-1)$-truss after the merger, completing the proof of the limited decrease. Regarding the increase, consider the inverse operation of merging two nodes, and we shall show the decrease is limited. Formally, we split v_1 in G' back into two nodes v_1 and v_2 in G, with $N(v_1; G') = N(v_1; G) \cup N(v_2; G)$. Regarding the trussness of each edge, this operation is no worse than deleting the node. Similarly, when we delete a node, for each k, for each edge in the current k-truss, the support decreases by at most 1, completing the proof. □

A.5 Proof of Corollary 1

Proof of Corollary 1. Recall that $T_{k-1}(G) \setminus \{v_1, v_2\}$ is defined as the subgraph obtained by removing v_1, v_2, and all their incident edges from $T_{k-1}(G)$. Since $G' \subseteq PM(v_1, v_2)$, $T_k(G') \subseteq T_k(PM(v_1, v_2))$. Hence, it suffices to show that $T_k(PM(v_1, v_2)) \subseteq T_k(G')$. First, by Lemma 1, for $e \in E(G \setminus \{v_1, v_2\})$, if $t(e; G) < k-1$, then $t(e; PM(v_1, v_2)) < k$ and thus $e \notin E(T_k(PM(v_1, v_2)))$, completing the proof for the first part $(T_{k-1}(G) \setminus \{v_1, v_2\})$. Second, for an edge (v_1, x), such an edge exists iff $x \in N(v_1) \cup N(v_2) \setminus \{v_1, v_2\}$; if $x \notin V(T_{k-1})$, then v_1 will be the only neighbor of x and thus (v_1, x) cannot be in the k-truss after the merger, completing the proof. □

A.6 Proof of Lemma 2

Proof of Lemma 2. If $t(v_2) < t(e)$, then $v_2 \notin V(T_{t(e)}(G))$. So merging v_1 and v_2 can only bring new edges into the $t(e)$-truss, and thus the trussness of e cannot decrease, completing the proof. □

A.7 Proof of Lemma 3

Proof of Lemma 3. $\Leftarrow$) Put $\{(v_1, x) : x$ is in the $(k-2)$-core of $T'_k[N^*]\}$ and $E(T'_k[N^*])$ together, each such (v_1, x) is in at least $k-2$ triangles $\triangle_{v_1 x x'}$ with $x' \in N^*$, completing the proof.

$\Rightarrow$) Let X denote $\{x : (v_1, x) \in T'_k\}$. For each $x \in X$, we have at least $k-2$ triangles $\triangle_{v_1 x x'}$ with all three constituent edges in T'_k. Hence $d(x; T'_k[N]) \geq k-2, \forall x \in X$, completing the proof. □

A.8 Proof of Lemma 4

Proof of Lemma 4. Given any G, by Lemmas 1 and 2, if $u \notin V(T_{k-1})$, then $T_k \subseteq T_k(PM(v, u)) \subseteq \tilde{T}_k \subseteq PM(v, u)$, where $\tilde{T}_k = \tilde{T}_k(v, u) = T_{k-1} \cup \{(v, x) : x \in N(v) \cup N(u) \setminus \{u, v\}\}, \forall v$. If $x \notin V(T_{k-1}) \cup$

Table 3: The basic statistics of the 14 real-world datasets. Notations: n denotes the number of nodes, n_k the number of nodes in T_k, m the number of edges, m_k the number of edges in T_k, and k_{max} the maximum k such that T_k is non-empty.

Dataset	n	m	k_{max}	n_5	m_5	n_{10}	m_{10}	n_{15}	m_{15}	n_{20}	m_{20}
email (EM) [41, 71]	986	16,064	23	743	14,771	492	10,494	257	5,308	73	1,622
facebook (FB) [43]	4,038	87,887	97	3,599	85,336	2,509	74,436	1,707	62,567	1,196	52,884
enron (ER) [42, 71]	33,696	180,811	22	13,983	139,351	2,159	53,913	769	21,837	192	4,441
brightkite (BK) [16]	56,739	212,945	43	8,009	74,498	1,454	27,742	544	15,950	353	12,274
relato (RL) [34]	54,007	251,370	44	6,897	144,787	2,386	89,041	1,282	60,093	781	41,808
epinions (EP) [57]	75,877	405,739	33	9,706	218,990	3,138	111,694	1,357	55,560	593	25,679
hepph (HP) [40]	34,401	420,784	25	22,760	298,416	5,011	75,343	864	14,065	124	2,109
slashdot (SD) [39]	77,360	469,180	35	4,048	72,554	638	19,174	372	13,036	237	9,554
syracuse (SC) [58]	13,640	543,975	59	12,274	484,914	8,696	301,374	5,446	185,365	3,672	128,992
gowalla (GW) [16]	196,591	950,327	29	42,860	434,483	7,163	140,993	2,060	52,009	531	16,381
twitter (TT) [43]	81,306	1,342,296	82	61,162	1,255,418	35,354	961,958	21,911	697,239	13,592	479,795
stanford (SF) [42]	255,265	1,941,926	62	151,955	1,569,406	49,199	934,901	33,980	694,205	16,157	383,159
youtube (YT) [68]	1,134,890	2,987,624	19	42,508	543,739	4,061	120,055	998	33,637	0	0
wikitalk (WT) [38, 39]	2,388,953	4,656,682	53	34,509	811,728	6,577	405,501	3,349	281,684	2,259	214,676

Algorithm 5: Prune outside nodes (based on [26])

Input : outside nodes V_o; inside neighbors $\tilde{N}_k(v), \forall v \in V_o$
Output: V_o^*: the outside nodes with maximal set of inside neighbors

1 $S, V_o' \leftarrow \emptyset$
2 **for** $v \in V_o$ **do**
3 **if** $\tilde{N}_k(v) \notin S$ **then** $\{S \leftarrow S \cup \{\tilde{N}_k(v)\}; V_o' \leftarrow V_0' \cup \{v\}\}$
4 $i \leftarrow 0; m(v) \leftarrow 0, \forall v \in V_o; V_o^* \leftarrow \emptyset$
5 **for** $v \in V_o'$ **do**
6 **for** $u \in \tilde{N}_k(v)$ **do** $m(u) \leftarrow \texttt{BitwiseOr}(m(u), 2^i)$
7 $i \leftarrow i + 1$
8 $r(v) \leftarrow \texttt{BitwiseAnd}(\{m(u) : u \in \tilde{N}_k(v)\}), \forall v \in V_o'$
9 $V_o^* \leftarrow \{v : r(v)$ is a power of $2\}$
10 **return** V_o^*

$\{v, u\}$, then $d(x; T_k(PM(v,u))) \le d(x; \tilde{T}_k(v,u)) = 0$. By Lemma 3, $x \notin V(T_k(PM(v,u)))$, and thus $T_k(PM(v,u)) = T_k(\tilde{T}_k(v,u)) = \hat{T}_k(v,u)$, where $\hat{T}_k(v,u) = T_{k-1} \cup \{(v,x) : x \in (N(v) \cup N(u) \setminus \{v,u\}) \cap V(T_{k-1})\}$. For $u_1, u_2 \notin V(T_{k-1})$, if $N(u_1) \cap V(T_{k-1}) \subseteq N(u_2) \cap V(T_{k-1})$, then $\hat{T}_k(v,u_1) \subseteq \hat{T}_k(v,u_2)$. If $\tilde{N}_k(u1) = \tilde{N}_k(u2)$, i.e., $\tilde{N}_k(u1) \subseteq \tilde{N}_k(u2) \wedge \tilde{N}_k(u2) \subseteq \tilde{N}_k(u1)$, then $T_k(PM(v,u_1)) \subseteq T_k(PM(v,u_2)) \wedge T_k(PM(v,u_2)) \subseteq T_k(PM(v,u_1))$, i.e., $T_k(PM(v,u_1)) = T_k(PM(v,u_2))$, completing the proof. □

A.9 Proof of Lemma 5

Proof of Lemma 5. It suffices to show that for any $u \in V_o \setminus \tilde{V}_o$, if a merger includes u, then there exists another merger consisting of two nodes in $V(T_{k-1}) \cup \tilde{V}_o$ with no worse performance. And this is an immediate corollary of Lemma 4. □

A.10 Proof of Lemma 6

Proof of Lemma 6. Lines 1 to 3 remove the outside nodes with duplicate inside neighborhood and take $O(|V_o|)$ times. Lines 4 to 7 build the membership function m where for a inside node u, the i-th bit of $m(u)$ indicates the membership relation between u and the i-th element of V_o', which takes $O(\sum_{v \in V_o} |\tilde{N}_k(v)|)$ time. Lines 8 to 9 use m to check the maximality of each unique inside neighborhood and take $O(\sum_{v \in V_o} |\tilde{N}_k(v)||V_o|)$. For the correctness, $r(v)$ consists of the nodes v' with $\tilde{N}_k(v') \supseteq \tilde{N}_k(v)$. If the final r is a power of 2, i.e., exactly a single bit of r is 1, then this bit represents v itself, which means that no other v' satisfies that $\tilde{N}_k(v') \supseteq \tilde{N}_k(v)$. Regarding the space complexity, the inputs and all the variables (S, V_o', and V_o^*) take $O(|E|)$ space, $m(v)$ for all $v \in V_o^*$ takes $O(\sum_{v \in V_o'} |\tilde{N}_k(v)|) = O(|E|)$ space if we represent the binary arrays in a sparse way [5]. □

A.11 Proof of Lemma 7

Proof of Lemma 7. If an edge e_0 is inserted into G such that the trussness of e_0 after the insertion is l, then all edges with original trussness at least l will not gain any trussness, and the remaining edges can gain at most 1 trussness [30]. Hence, it suffices to show that for each considered x, after inserting (v_1, x) into G, the trussness of (v_1, x) is at most $k-1$. Indeed, since $v_1 \notin T_{k-1}$, all edges incident to v_1 have original trussness at most $k-2$ and thus have trussness at most $k-1$ after the insertion. Therefore, all triangles containing (v_1, x) will not be in T_k and thus neither will (v_1, x). □

A.12 Proof of Lemma 8

Proof of Lemma 8. Finding the top-n_i inside nodes and top-n_o outside nodes (Lines 1 and 2) takes $O(|V_i| \log n_i + |V_o^*| \log n_o)$. For all inside nodes and all "new" neighbors, computing the incident PHSEs (Lines 3 to 4) takes $O(n_i |V(T_{k-1})|)$ time; and computing PHSEs for all pairs (Lines 5 to 9) takes $O(n_i n_o (|V(T_{k-1})| + |\hat{E}_k|))$ time. Maintaining the candidate set takes $O(n_i n_o \log n_c)$ time. Hence, the total time complexity is $O(|V_o^*| \log n_o + n_i n_o (|V_{k-1}| + |\hat{E}_k| + \log n_c))$. □

A.13 Proof of Lemma 9

Proof of Lemma 9. Finding the top-n_i inside nodes (Line 1) takes $O(|V_i| \log n_i)$ time. For all pairs among the chosen inside nodes, computing the scores (Lines 2 to 8) takes $O(n_i^2 |\hat{E}_k|)$ time. Maintaining the set of candidates IOMs takes $O(n_i^2 \log n_c)$ time. Therefore, the total time complexity is $O(|V_i| \log n_i + n_i^2 (|\hat{E}_k| + \log n_c))$. □

A.14 Proof of Theorem 4

Proof of Theorem 4. In each round, truss decomposition (Line 3) takes $O(|E|^{1.5})$ time. Collecting all the information (Lines 4 to 8) takes $O(|E|)$ time. By Lemmas 8 and 9, obtaining the candidate mergers (Lines 9 and 10) takes $O(|V_o^*| \log n_o + n_i n_o (|V_i| + |\hat{E}_k| + \log n_c) + n_i^2 (|\hat{E}_k| + \log n_c))$ time. Checking the results after all candidates (Line 11) takes $O(n_c |E(T_{k-1})|^{1.5})$ time. Updating the graph (Line 14) takes $O(|E|)$ time. Hence, it takes $O(b(|E|^{1.5} + n_c |E(T_{k-1})|^{1.5} + |V_o^*| \log n_o + n_i n_o (|V_i| + |\hat{E}_k| + \log n_c) + n_i^2 (|\hat{E}_k| + \log n_c)))$ time in total. All the inputs and variables take $O(|E| + n_c)$ space, including the intermediate ones in Algorithms 2 and 3 (note that we only maintain the set of best candidate nodes and pairs). By Lemma 6, Algorithm 5 takes $O(|E|)$ space. Hence, the total space complexity is $O(|E| + n_c)$. □

B DETAILS OF ALGORITHMS AND DATASETS

See Algorithm 5 and Table 3 for details omitted in the main text.