On Breaking Truss-based and Core-based Communities

HUIPING CHEN, University of Birmingham, Birmingham, UK
ALESSIO CONTE, University of Pisa, Pisa, Italy
ROBERTO GROSSI, University of Pisa, Pisa, Italy
GRIGORIOS LOUKIDES, King's College London, London, UK
SOLON P. PISSIS, CWI, Amsterdam, The Netherlands and Vrije Universiteit, Amsterdam, The Netherlands
MICHELLE SWEERING, CWI, Amsterdam, The Netherlands

We introduce the general problem of identifying a smallest *edge* subset of a given graph whose deletion makes the graph community-free. We consider this problem under two community notions that have attracted significant attention: k-truss and k-core. We also introduce a problem variant where the identified subset contains edges incident to a given set of nodes and ensures that these nodes are not contained in any community: k-truss or k-core, in our case. These problems are directly applicable in social networks: The identified edges can be *hidden* by users or *sanitized* from the output graph; or in communication networks: the identified edges correspond to *vital* network connections. We present a series of theoretical and practical results. On the theoretical side, we show through non-trivial reductions that the problems we introduce are NP-hard and, in fact, hard to approximate. For the k-truss-based problems, we also show exact exponential-time algorithms, as well as a non-trivial lower bound on the size of an optimal solution. On the practical side, we develop a series of heuristics that are sped up by efficient data structures that we propose for updating the truss or core decomposition under edge deletions. In addition, we develop an algorithm to compute the lower bound. Extensive experiments on 11 real-world and synthetic graphs show that our heuristics are effective, outperforming natural baselines, and also efficient (up to two orders of magnitude faster than a natural baseline), thanks to our data structures. Furthermore, we present a case study on a co-authorship network and experiments showing that the removal of edges identified by our heuristics does not substantially affect the clustering structure of the input graph.

This work extends a KDD 2021 paper, providing new theoretical results as well as introducing core-based problems and algorithms.

CCS Concepts: • **Mathematics of computing → Graph algorithms**; • **Information systems → Data mining;**

Additional Key Words and Phrases: Graph algorithms, k-truss, k-core, community detection

Michelle Sweering is supported by the Netherlands Organisation for Scientific Research (NWO) under the Gravitation-grant NETWORKS-024.002.003. Alessio Conte and Roberto Grossi are supported, in part, by MUR of Italy, under PRIN Project n. 2022TS4Y3N - EXPAND: scalable algorithms for EXPloratory Analyses of heterogeneous and dynamic Networked Data. Roberto Grossi is partially supported by NextGeneration EU programme PNRR ECS00000017 Tuscany Health Ecosystem.
Authors' addresses: H. Chen, University of Birmingham, B15 2TT, Birmingham, UK; e-mail: h.chen.13@bham.ac.uk; A. Conte and R. Grossi, University of Pisa, 6126 Pisa, Italy; e-mails: alessio.conte@unipi.it, roberto.grossi@unipi.it; G. Loukides, King's College London, Department of Informatics, WC2R 2LS, London, UK; e-mail: grigorios.loukides@kcl.ac.uk; S. P. Pissis, CWI, 1098 XG Amsterdam, The Netherlands and Vrije Universiteit, 1081 HV Amsterdam, The Netherland; e-mail: solon.pissis@cwi.nl; M. Sweering, CWI, 1098 XG Amsterdam, The Netherlands; e-mail: michelle.sweering@cwi.nl.

ACM 1556-4681/2024/04-ART135
https://doi.org/10.1145/3644077

ACM Reference Format:
Huiping Chen, Alessio Conte, Roberto Grossi, Grigorios Loukides, Solon P. Pissis, and Michelle Sweering. 2024. On Breaking Truss-based and Core-based Communities. *ACM Trans. Knowl. Discov. Data.* 18, 6, Article 135 (April 2024), 43 pages. https://doi.org/10.1145/3644077

1 INTRODUCTION

Graphs naturally model relationships between entities in a multitude of domains such as social networks, communication networks, or the web. A fundamental data analysis task in these domains is *community detection* (i.e., the identification of cohesive or dense subgraphs of a given graph). In the community detection process, different notions of graph communities, including the notions of k-plex, n-clan, n-club, k-core, k-ECC, or k-truss, are employed [21, 23]. These notions essentially relax the classic notion of clique [28] (i.e., the ideal situation, where all nodes are pairwise connected), either to capture practical application considerations [21] or to enable more efficient enumeration [15, 17].

1.1 The Community Breaking Problem

Irrespective of the community notion, the community structure is a key property of a graph. It is therefore essential to study how such a structure can be maintained or broken [20, 33, 40, 49, 55]. Here, we investigate the following general problem:

***Community Breaking* (CB)** problem: Given an undirected graph $G(V, E)$, a set of nodes $U \subseteq V$, and a notion of community, identify a smallest subset E' of E, so no community in $G' = G(V, E \setminus E')$ contains a node in U.

The CB problem is motivated by the following real-world applications:

A1. Maintaining communities in social networks [55]. The edges identified in the output of CB correspond to critical edges for maintaining user engagement in communities.

A2. Assessing resilience to attacks or errors in communication networks [33]. The edges identified in the output of CB correspond to vital connections in the network.

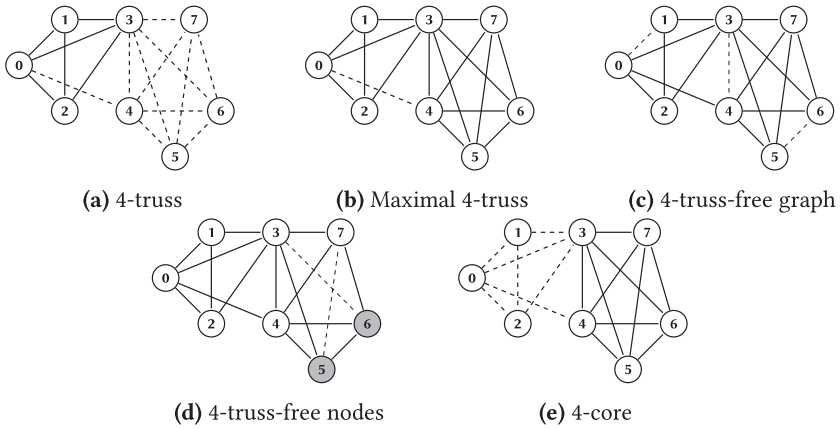
A3. Enabling social network users to hide friendships, so they are not seen as belonging to communities that could lead to their discrimination or unwanted targeted advertisement (e.g., through friend-based profile attribute inference attacks [7]). The edges identified in the output of the CB problem correspond to friendships users could opt to hide [20].

A4. Preventing the detection of confidential communities by sanitizing a graph prior to its dissemination, in the spirit of sanitization works on transaction [41] or sequential data [8, 9]. The edges identified in the output of the CB problem must be deleted to hide these communities in the sanitized graph.

Identifying a small edge subset is natural yet crucial. In A1 and A2, it allows estimating the number of social network unfriendings or communication connection failures that could trigger the loss of community structure, respectively. In A3 and A4, it allows less effort from users and more accurate analysis of the sanitized graph, respectively.

We focus on two well-established community notions: k-truss [15] and k-core [42].

A *k-truss* is a subgraph of a graph such that each edge is contained in at least $k-2$ triangles of the subgraph (see Figure 1(a)). A *maximal k-truss* is the largest *k-truss* of the graph (see Figure 1(b)). The notion of k-truss has attracted significant interest because: (1) k-trusses are less expensive to enumerate than cliques, k-plexes, n-clans, and n-clubs, as well as more cohesive than k-cores and k-ECCs [21]; and (2) k-trusses model meaningful cohesive subgraphs in communication [55], social [26], or collaboration [26] networks, thanks to good structural properties such as bounded diameter or strong decomposability [21].



(a) 4-truss **(b)** Maximal 4-truss **(c)** 4-truss-free graph

(d) 4-truss-free nodes **(e)** 4-core

Fig. 1. (a) The subgraph induced by the solid edges is a 4-truss, because every edge of the subgraph is contained in at least $4 - 2 = 2$ triangles of the subgraph. (b) The subgraph induced by all edges except the (dashed) edge $(0, 4)$ is the maximal 4-truss of the graph. (c) The graph obtained after deleting the set $\{(0, 1), (3, 4), (5, 6)\}$ of (dashed) edges contains no 4-truss. (d) The graph obtained after deleting the set $\{(3, 6), (5, 7)\}$ of (dashed) edges is a graph in which the (gray) nodes 5 and 6 are not contained in any 4-truss. (e) The subgraph induced by the solid edges is a 4-core.

A *k-core* is a subgraph of a graph such that each node has degree at least k in the subgraph. A *maximal* k-core is the largest k-core of the graph. As mentioned above, k-cores are less cohesive than k-trusses. They are also generally larger (e.g., a k-truss is a $(k - 1)$-core but not vice versa). k-cores have attracted much theoretical interest [31], and they have been used in a range of applications, such as computer network structure analysis [4], social network user engagement [10], peer-to-peer network content sharing [53], and communication network resilience optimization [33], partly because they can be retrieved in just linear time [6].

1.2 Truss-based and Core-based Community Breaking Problems

We define combinatorial optimization problems of practical importance that are based on the CB problem. We first define problems using the k-truss community notion.

***MIN-k-TBS* (Minimum k-Truss Breaking Set)**: Given an undirected graph $G(V, E)$ and a parameter k, find a smallest subset E' of E such that $G(V, E \setminus E')$ contains no k-truss. MIN-k-TBS is obtained from the CB problem by considering communities based on the notion of k-truss and $U = V$. MIN-k-TBS addresses applications A1 and A2 above.

Example 1.1. An optimal solution to MIN-k-TBS with $k = 4$ on Figure 1(c) is the set of (dashed) edges $E' = \{(0, 1), (3, 4), (5, 6)\}$. This is because deleting these edges leads to a graph with no 4-truss and because deleting any fewer edges leads to a graph that contains a 4-truss.

***MIN-k-TCBS* (Minimum k-Truss Communities Breaking Set):** Given an undirected graph $G(V, E)$, a parameter k, and a set $U \subseteq V$, find a smallest subset E' of E such that the edges in E' are incident to nodes in U and no k-truss in $G(V, E \setminus E')$ contains a node in U. MIN-k-TCBS is obtained from the CB problem by considering communities based on the notion of k-truss, having U as input, and further limiting what edges can be deleted. It addresses applications A3 and A4 above. In particular, the requirement for edges in E' to be incident to nodes in U is necessary for A3, where users can only hide their own edges. Waiving this requirement for A4 is trivial.

Example 1.2. An optimal solution to MIN-k-TCBS with $k = 4$ and $U = \{5, 6\}$ on the graph of Figure 1(d) is the set of edges $E' = \{(3, 6), (5, 7)\}$. This is because deleting these edges leads to a graph in which neither node 5 nor node 6 belong to a 4-truss, and because deleting any single edge does not prevent both nodes from belonging to a 4-truss.

We also define problems based on the k-core community notion. By simply using k-core as the community notion in the CB problem, or, equivalently, by replacing the notion of k-truss with that of k-core in MIN-k-TBS and in MIN-k-TCBS, we obtain the MIN-k-CBS (Minimum k-Core Breaking Set) and MIN-k-CCBS (Minimum k-Core Communities Breaking Set) problem, respectively.

Example 1.3. An optimal solution to MIN-k-CBS with $k = 4$ on the graph of Figure 1(e), which contains a 4-core, is the edge $\{(5, 6)\}$. This is because deleting this edge leads to a graph that does not contain a 4-core.

All problems we study are *intuitively* challenging: There are up to $2^{|E|}$ edge subsets that one may consider; and then both k-trusses and k-cores have a hierarchical structure. For example, a $(k + i)$ truss, for any k and $i \geq 0$, is also a k-truss and contains at least $\binom{k+i}{k}$ smaller k-trusses. Furthermore, the truss-based problems are fundamentally different from those based on cores. Intuitively, this is because trusses are based on a property of edges (number of triangles containing an edge), while cores are based on a property of nodes (number of edges incident to a node). Thus, these two classes of problems require different methods. For example, although a 5-truss is a 4-core, the graph in Figure 1(a) contains no 5-truss, but it does contain a 4-core. Thus, a method for breaking 5-trusses would not break the 4-core in this graph. Similarly, it is possible to break the 4-truss by breaking all 3-cores, but this requires deleting many *more* edges than the three edges needed to break the 4-truss (see Example 1.1 and Figure 1(c)).

1.3 Our Contributions and Article Organization

1. We define four problems: MIN-k-TBS, MIN-k-TCBS, MIN-k-CBS, and MIN-k-CCBS. See Section 3. We then present a series of highly non-trivial hardness results for these problems. For the truss-based problems, we show that: (1) MIN-k-TBS and MIN-k-TCBS are both NP-hard; (2) MIN-3-TBS and MIN-3-TCBS can be 3-approximated in polynomial time; and (3) $(k - 2 - \epsilon)$-approximating MIN-k-TBS and MIN-k-TCBS for $k > 3, \epsilon > 0$ is hard (under the unique games conjecture [29]). For the core-based problems, we show that: (1) MIN-2-CBS and MIN-2-CCBS can be solved in polynomial time; and (2) $(k - \epsilon)$-approximating MIN-k-CBS and MIN-k-CCBS for $k \geq 3, \epsilon > 0$ is hard (under the unique games conjecture). See Section 4.

2. We show a data structure for maintaining the *truss decomposition* of a graph (i.e., the maximal k-truss for each k) under edge deletions with theoretical guarantees. We also show a data structure for maintaining the *core decomposition* of a graph (i.e., the maximal k-core for each k) under edge deletions with theoretical guarantees. These data structures form the backbone of our heuristics. In addition, we provide the necessary set of combinatorial tools for designing our algorithms. See Section 3.

3. We develop exact algorithms for both MIN-k-TBS and MIN-k-TCBS. These algorithms are useful for evaluating our heuristics on *small-scale* graphs. See Section 5.

4. We develop three heuristics for MIN-k-TBS, namely, MBH_S, MBH_C, and SNH, based on different theoretical insights. MBH_S and MBH_C break a k-truss by deleting a single edge, based on different criteria regarding the triangles that contain the edge, while SNH aims to limit the unnecessary subsequent deletion of neighboring edges of a deleted edge. Our heuristics always return a feasible solution and can be trivially modified to solve MIN-k-TCBS. See Section 6.

5. We show a non-trivial lower bound on the size of an optimal solution to MIN-k-TBS and develop an efficient algorithm to compute it. The lower bound is useful for rigorously evaluating

our heuristics on *large-scale* graphs and also for quickly assessing the quality of any solution on a graph of interest before executing any heuristic. This is because an optimal solution always lies in between the lower bound and the solutions produced by heuristics. See Section 7.

6. We develop two heuristics for MIN-k-CBS, namely, CPH and SDNH, based on different theoretical insights. CPH aims to delete edges leading to the removal of many nodes from the *max-core* (i.e., a k-core of a graph that has no k'-core, for any $k' > k$), while SDNH aims to delete edges that minimize the total number of deleted edges. Our heuristics always return a feasible solution and can be trivially modified to solve MIN-k-CCBS. See Section 6.

7. We perform an extensive experimental evaluation using 11 real-world datasets, with up to millions of edges, as well as using 1,000 small-scale synthetic datasets. The evaluation shows that our heuristics are very effective, outperforming natural baselines, and at the same time efficient, thanks to the deployment of our data structures. It also shows that core-based heuristics are not effective for dealing with the truss-based problems and that truss-based heuristics are not effective for dealing with the core-based problems. This underlines the need to study both truss-based and core-based problems. We conclude the experimental evaluation with a case study on a co-authorship network and experiments suggesting that the removal of edges identified by our heuristics does not substantially affect the clustering structure of the input graph. See Section 9.

The rest of the article is structured as follows: In Section 2, we highlight the applicability of our algorithms via analyzing *a real social network*. Related work is discussed in Section 8. We conclude this article in Section 10.

A preliminary version of our work without the approximability/inapproximability results for MIN-k-TBS and MIN-k-TCBS and without the study of core-based problems appeared in KDD 2021 [13].

2 ANALYZING REAL SOCIAL NETWORKS

In this section, we showcase the applicability of our methods in a real-world setting. We used a dataset [47] that contains friendship and location information from 114,324 Twitter users who recorded their check-ins in 3,820,891 Foursquare venues. We refer to this dataset as FL (Friendship-Location). FL can be viewed as a node-labeled graph with 114,324 nodes and 607,327 edges. Each node corresponds to a user, each node label contains the check-ins of the user, and each edge corresponds to a friendship between two users.

FL contains check-ins to venues that may indicate users' sexual orientation, religious beliefs, or gambling habits. Moreover, the k-trusses in FL contain a large number of users who checked in to such venues, or are friends with users who checked in to such venues. For example, an 18-truss contained: (1) $^{115}/_{280} = 41.1\%$ of users who checked in to `gay bars`; and (2) $^{7,786}/_{11,732} = 66.4\%$ of pairs of friends, at least one of whom checked in to `gay bars`. Similarly, a 24-truss contained: (1) $^{57}/_{131} = 43.5\%$ of users who checked in to `casinos` or `strip clubs`; and (2) $^{2,973}/_{4,385} = 67.8\%$ of pairs of friends, at least one of whom checked in either to a `casino` or to a `strip club`.

Some users may not want to be seen as belonging to such k-trusses to avoid being discriminated based on the *homophily* theory; friends are more likely to share attributes [7]. These users can achieve this goal by employing our algorithms for MIN-k-TCBS to hide a small number of their friendships (e.g., by setting them to "private"), so that they are not seen as belonging to these k-trusses [20]. In fact, when applied with $k = 18$ and U containing 65 randomly selected nodes among those in the aforementioned 18-truss, our algorithms ensured that the 18-truss contains no nodes in U. Thus, the users corresponding to nodes in U are not seen as belonging to that truss. Specifically, MBH_S, MBH_C, and SNH achieved this by deleting only 0.057%, 0.112%, and 0.056% of the edges of the graph, in 1.6, 1.7, and 13.7 seconds, respectively. Similarly, when applied with $k = 24$ and U containing 13 randomly selected nodes among those in the aforementioned 24-truss,

MBH_S, MBH_C, and SNH ensured that no 24-truss containing nodes in U exists by deleting only 0.012%, 0.016%, and 0.008% of the edges of the graph, in 0.5, 0.6, and 1.2 seconds, respectively.

A k-truss is at least a $(k-1)$-core (see Lemma 3.1) and has size no larger than that of the k-core. Thus, the k-cores in FL also contain users who checked in to venues that may be used to discriminate them. For example, the aforementioned 24-truss is a subgraph of a 23-core that contains: (1) $644/2,541 = 25.3\%$ of users who checked in to `casinos` or `strip clubs`; and (2) $42,402/87,622 = 48.4\%$ of pairs of friends, at least one of whom checked in to a `casino` or `strip club`. Note that the percentages are lower, but the 23-core contains many more users that may not want to be seen as belonging to it, compared to the 24-truss. Recall that U in the case of 24-truss contained 13 nodes. To ensure that the 23-core does not contain any of these nodes, CPH and SDNH deleted 0.29% and 0.29% of the edges of the graph, in 9 and 46 seconds, respectively.

These results highlight the applicability of our methods in realistic settings.

3 PRELIMINARIES AND TECHNIQUES

We fix an undirected graph $G(V, E)$, with no multiple edges or self-loops. By $N_G(u)$, we denote the set of neighbors of a node $u \in V$ and by $|N_G(u)|$ its degree. A subgraph of G is defined by a set $S \subseteq E$; we use S to represent the set of edges or the subgraph of G induced by S. A triangle in G is a subgraph of three edges $\{e, f, g\}$ connecting three distinct nodes in V.

Given a subgraph S of G, the *support* of an edge e in S is the number of triangles of S that contain e and is denoted by $\sup_S(e)$. For an integer $k \geq 3$, a *k-truss* of G is a subgraph S of G such that every edge e in S has $\sup_S(e) \geq k-2$. The largest such subgraph (not necessarily connected) is called the *maximal k-truss* of G. A *minimal k-truss* of G is a subgraph S of G such that S is a k-truss, but no subset $S' \subset S$ is a k-truss. The trussness of G, denoted by $t(G)$, is the largest k such that there exists a k-truss in G, and the maximal k-truss with $k = t(G)$ is called the *max-truss* of G.

The *edge trussnness* $t(e)$ of an edge e is the largest k for which e belongs to a k-truss. The *truss decomposition* of G associates $t(e)$ to each e; equivalently, it is the set comprised of the maximal k-truss of G, for each k. It can be computed in $O(|E|^{\frac{3}{2}})$ time, e.g., using the algorithm of Reference [16].

The number of triangles of G is denoted by $\mathcal{T}_G$. All triangles of G can be computed in $O(|E|^{\frac{3}{2}})$ time [3], which also implies that $\mathcal{T}_G = O(|E|^{\frac{3}{2}})$. The *trussness of a triangle* is the minimum among the trussness of the edges of this triangle.

A *k-Truss Breaking Set* (k-TBS) of G is a set $E' \subseteq E$ such that the graph $G(V, E \setminus E')$ contains no k-truss. MIN-k-TBS asks for a *smallest* k-TBS:

Problem 1 (Minimum k-Truss Breaking Set (MIN-k-TBS)). *Given a graph $G(V, E)$ and an integer $k \geq 3$, find a smallest k-truss breaking set of G.*

Problem 2 (Minimum k-Truss Communities Breaking Set (MIN-k-TCBS)). *Given a graph $G(V, E)$, an integer $k \geq 3$, and a set $U \subseteq V$, find a smallest set $E' \subseteq E$ comprised of edges incident to nodes in U, so that no k-truss in the graph $G(V, E \setminus E')$ contains a node in U.*

For an integer $k \geq 1$, a *k-core* of G is a subgraph of G such that for every node u in G, we have that $|N_G(u)| \geq k$. The largest such subgraph (not necessarily connected) is the *maximal k-core* of G. The coreness of G (a.k.a. *graph degeneracy*), denoted by $c(G)$, is the largest k such that there exists a k-core in G. The maximal k-core with $k = c(G)$ is called the *max-core* of the graph G.

Given a node u, its *node coreness* $c(u)$ is the largest k for which u belongs to a k-core. We also define the *edge coreness* of an edge e analogously and denote it by $c(e)$. Clearly, for any edge (u, v), it holds that $c((u, v)) = \min(c(u), c(v))$.

The *core decomposition* of G associates to each node (or edge) its coreness. It can be computed in $O(|V| + |E|) = O(|E|)$ time, e.g., using the algorithms of [6, 34].

The k-Core Breaking Set of G is a set $E' \subseteq E$ such that $G(V, E \setminus E')$ contains no k-core. It leads to the Minimum k-Core Breaking Set (MIN-k-CBS) and the Minimum k-Core Communities Breaking Set (MIN-k-CCBS) problems. These problems are analogous to MIN-k-TBS and MIN-k-TCBS, respectively, and require an integer $k \geq 2$ in their input.

3.1 Combinatorial Properties

We show some combinatorial properties employed by our algorithms.

Lemma 3.1. *All nodes in a k-truss have degree at least $k-1$.*

Proof. An edge e of the k-truss has, by definition, support $\geq k-2$. Thus, the edge e is adjacent to at least $k-2$ other edges of the k-truss on each of its endpoints, and each endpoint is incident to at least $k-2$ edges plus e itself. □

Corollary 3.2. *A k-truss with v nodes satisfies $v \geq k$ and has at least $v \cdot (k-1)/2$ edges.*

Proof. By Lemma 1, each vertex in the k-truss has degree $\geq k-1$. As each edge e has support at least equal to $k-2$ by definition, the k-truss contains at least k nodes (2 incident to e and $k-2$ as the third vertex of each triangle). The claim is completed by the "hand-shaking lemma": a graph has as many edges as the sum of degrees of its vertices divided by 2. □

Lemma 3.3. *If a k-truss S contains no k-truss after deleting any of its edges, then S is a minimal k-truss.*

Proof. Every subgraph of S is contained in at least a graph created by removing an edge of S. If no such graph contains a k-truss, then no subgraph of S contains a k-truss, which implies that S is a minimal k-truss. □

Lemma 3.4. *Let S be a k-truss but not a $(k+1)$-truss. Then there exists an edge e in S such that $S \setminus e$ is not a k-truss.*

Proof. Let e' be an edge of minimum support in S, which must be exactly $k-2$ (or S would be a $k+1$-truss), and let e be one of the edges in S forming a triangle of X with e'. The support of e' in $S \setminus e$ is $k-3$ and the claim follows: □

Lemma 3.5. *Let S be a k-core but not a $(k+1)$-core. Then there exists an edge e in S such that $S \setminus e$ is not a k-core.*

Proof. Let $V(S)$ be all nodes incident to S, and let H be the induced subgraph on $V(S)$. Let v be a node of smallest degree in H: Its degree is exactly k, since S is not a $(k+1)$-core. Finally, let e be an edge incident to v; if we delete e, then v has degree at most $k-1$ and thus S is not a k-core. □

Note that although $S \setminus e$ in Lemma 3.4 is not a k-truss, it may still *contain* a smaller k-truss. Finally, we recall from Reference [16] a useful property that lets us bound the trussness of a graph to the number of triangles and edges it contains.

Theorem 3.6 ([16]). *Given a graph G with m edges, $\mathcal{T}_G$ triangles, and trussness $t(G)$, it holds that $t(G) \geq \frac{\mathcal{T}_G}{m} + 2$ and $t(G) = O(\sqrt{m})$.*

3.2 Triangles Update

We describe `Triangles Update`, a simple data structure to maintain the triangles of G under edge deletions. We construct a perfect hash table H in which the set of keys is the set of edges in G that are contained in at least one triangle. The value $H[e]$ for key (edge) e is the doubly linked list of triangles $\{e, f, g\}$ containing e. Triangle $\{e, f, g\}$ exists also in $H[f]$ and $H[g]$. Element $\{e, f, g\}$ in

$H[e]$ has two pointers to the elements in $H[f]$ and $H[g]$ representing $\{e, f, g\}$. The size of this data structure is in $O(\mathcal{T}_G)$. Constructing it takes $O(|E|^{\frac{3}{2}})$ time, the time to compute all triangles. Upon deleting edge e, we can:

(1) Find the list of triangles $\{e, f, g\}$, for all f, g, in time linear in the size of this list. This is precisely $H[e]$.
(2) Remove elements $\{e, f, g\}$, for all f, g, from $H[e]$, $H[f]$, and $H[g]$ in time linear in the size of the elements we remove. We do this by traversing $H[e]$ and updating the pointers.

3.3 Truss Decomposition Update

A practical algorithm for updating the truss decomposition of a graph under edge insertions was proposed in Reference [26]. The authors mention that a similar algorithm for deletion could be derived, however, no theoretical guarantee is given for either. Here, we present a new data structure under edge deletions with theoretical guarantees. In particular, our data structure maintains the trussness $t(e)$ of all edges $e \in E$ under edge deletions. To update $t(e)$, it also maintains the number of triangles of different trussness that e belongs to. Indeed, if e belongs to at least $k-2$ triangles of trussness at least k, then e belongs to a k-truss. We also analyze the construction time, the size, and the update time for our data structure.

The deletion of e breaks all triangles hinging on e, and $t(e)$ trivially becomes zero. In addition, when the trussness of an edge decreases, the trussness of any edge that shares a triangle with that edge may decrease too. Hence, the effect of deleting a single edge can propagate through the graph.

Based on these observations, we design a data structure, called `Truss Update`, for efficiently maintaining the truss decomposition under edge deletions. Our data structure requires:

(1) The trussness $t(G)$ of the input graph.
(2) An array T_{cur} of size $|E|$ containing the trussness $t_{\text{cur}}(e)$ of each edge e of the current graph.
(3) The data structure TRI implemented by `Triangles Update` (Section 3.2) for maintaining all triangles containing each edge of the current graph.
(4) A 2D array T of size $|E| \times t(G)$, whose element $T[e][i]$ corresponds to the number of triangles of trussness i in TRI(e), for every edge e in the current graph and every $i \in [3, t(G)]$.
(5) A stack L of edges whose trussness is to be updated.

When e is deleted, we assign $T[e][i]$ to 0, for each $i \in [3, t(G)]$. This is because e is no longer contained in any triangles. We also push e to L for its trussness to be updated and propagated. Then, while L is nonempty, we repeatedly pop an edge f from L and perform the following update operations (see Appendix A for details):

O_1: Update the trussness $t_{\text{cur}}(f)$ of edge f, if f is in fewer than $t_{\text{cur}}(f) - 2$ triangles of trussness $t_{\text{cur}}(f)$.

O_2: For each triangle $\{f, g, h\} \in \text{TRI}(f)$ whose trussness changes, update its entries in T and push g and h into L.

Complexity Analysis. The data structure occupies $O(|E| \cdot t(G) + \mathcal{T}_G) = O(|E| \cdot t(G))$ space by Theorem 3.6. Constructing it takes $O(|E|^{\frac{3}{2}} + |E| \cdot t(G)) = O(|E|^{\frac{3}{2}})$ time. For the updates, observe that deleting an edge can only decrease the trussness of any other edge by at most 1. Say we delete edge e that had trussness x. Operation O_1 is implemented in $O(x)$ time for the deleted edge e and $O(1)$ time for all subsequent edges (as their trussness can only decrease by 1). Operation O_2 takes $O(\sup_G(f))$ time for each edge f whose trussness decreases. As we only consider each edge once on each deletion (and thus each triangle at most three times), the worst-case time for a single deletion is $O(\mathcal{T}_G)$. The advantage is the *amortized* time complexity of updating. We only add edges

to L whose trussness is decreased. When we delete all edges, we decrease their trussness by at most $t(G)$ and hence update the trussness of each triangle at most $3t(G)$ times. Therefore, deleting all edges takes only $O(t(G) \cdot \mathcal{T}_G) = O(|E|^2)$ time. However, recomputing the truss decomposition of the graph *after every deletion* would need $O(|E|^{\frac{5}{2}})$ time for deleting all edges; that is, $O(|E|^{\frac{3}{2}})$ time per edge deletion.

3.4 Coreness Update

We describe `Coreness Update`, a simple data structure to maintain the coreness of edges under edge deletions. We construct a perfect hash table R in which the set of keys is the set of non-isolated nodes in G. The value $R[u]$ for key (node) u is another perfect hash table M_u, where $M_u[\rho]$ (for edge coreness ρ) is a doubly linked list comprised of edges incident to u that have coreness ρ. Element (u, v) with coreness ρ exists both in $M_u[\rho]$ and $M_v[\rho]$, and there is a pointer from (u, v) in $M_u[\rho]$ to (u, v) in $M_v[\rho]$, and vice versa. The size of this data structure is $O(|V| + |E|)$. Constructing it takes $O(|V| + |E|)$ time. Upon deleting edge (u, v) with coreness ρ, we can perform the following operations:

(1) Find the list of edges that are incident to u and have coreness ρ in $O(1)$ time. This is precisely $M_u[\rho]$.
(2) Update the coreness of the edges in $M_u[\rho]$ and the pointers, in time linear in the number of the edges we update. This is done by appending these edges into $M_u[\rho - 1]$, since an edge deletion can decrease the coreness of an edge by at most 1.
(3) Remove the element (u, v) from $M_u[\rho]$ and from $M_v[\rho]$, in time linear in the size of the largest of these linked lists. We do this by finding $R[u] = M_u$ and traversing $M_u[\rho]$ until encountering (u, v); removing (u, v) from $M_u[\rho]$ and from $M_v[\rho]$ utilizing the pointer from (u, v) in $M_u[\rho]$ to (u, v) in $M_v[\rho]$; and finally updating the pointer.

3.5 Core Decomposition Update

We present `Core Update`, a data structure for efficiently maintaining the core decomposition under edge deletions, since the effect of deleting an edge can propagate through the graph. Our data structure maintains the coreness $c(e)$ of all edges $e \in E$ under edge deletions. To update $c(e)$, we also maintain the number of edges of different coreness that e belongs to. `Core Update` requires:

(1) The coreness $c(G)$ of the input graph.
(2) An array C_{cur} of size $|E|$ containing the coreness $c_{cur}(e)$ of each edge e of the current graph.
(3) The hash table $R[u] = M_u$, for each node u of the current graph, implemented by `Coreness Update`.
(4) A 2D array C of size $|V| \times c(G)$, whose element $C[u][\rho]$ corresponds to the number of edges $(u, v) \in M_u$ of coreness ρ, for every node u of the current graph and every $\rho \in [1, c(G)]$.
(5) A stack L of edges whose coreness is to be updated.
(6) A perfect hash table K for storing visited edges.

When e is deleted, we push e to L for its coreness to be updated and propagated. Then, while L is nonempty, we pop an edge (u, v) from L. Let the coreness of (u, v) be ρ. We perform the following operations (see Appendix B for details):

O_1: Update the coreness of (u, v) in C_{cur}.

O_2: Update the entries for v in C. If the coreness of v changes, then we push the edges of $M_v[\rho]$ that are not in K into L, and we update K and $R[v]$.

O_3 : If (u, v) is deleted, then we perform O_2 for u.

Complexity Analysis. The data structure occupies $O(|V| \cdot c(G) + |E|) = O(|V| \cdot c(G))$ space, as $|E| \leq |V| \cdot c(G)$ by the definition of $c(G)$. Constructing it takes $O(|V| \cdot c(G))$ time. As for the

edge deletions, the overall cost is dominated by the updates of the hash tables, which is in turn proportional to the number of times an edge has its coreness decreased (i.e., $O(c(G))$). Following an argument similar to that in Section 3.3, the total cost is $O(c(G) \cdot |E|)$ time in the worst case (if all edges are deleted).

4 HARDNESS AND INAPPROXIMABILITY RESULTS

4.1 MIN-k-TBS Is NP-hard

Finding a minimum set of edges to delete to make a graph triangle-free is known to be NP-hard [48], and that is exactly the MIN-3-TBS problem. Assuming the Exponential Time Hypothesis (ETH), we cannot even solve this problem in $2^{o(|E'|)} \cdot n^{O(1)}$ time [5]. We will now prove that MIN-k-TBS is also NP-hard for $k > 3$ using a reduction from MIN-3-TBS.

Theorem 4.1. *For every $k \geq 3$, MIN-k-TBS is NP-hard.*

Proof. Let $G = (V, E)$ be a graph for which we want to solve MIN-3-TBS. Let T be the set of triangles in G. We consider a new graph $G_k = (V_k, E_k)$, which is constructed as follows: For each triangle $t \in T$, let $S_t := t \cup [k-3] \times \{t\}$ denote the 3 nodes in t and $k-3$ new nodes $(1, t), \ldots, (k-3, t)$. The new graph consists of the union of the cliques $\binom{S_t}{2}$ over all triangles $t \in T$. Formally,

$$G_k := \left(V \cup [k-3] \times T, \bigcup_{t \in T} \{\{u, v\} \mid u, v \in S_t, u \neq v\} \right),$$

where we observe that every edge of E is still in E_k as long as it participated in at least one triangle. We will now show that solving MIN-3-TBS for G is equivalent to solving MIN-k-TBS for G_k.

Suppose $G' = (V, E \setminus E')$ does not contain any triangles. Then for each triangle $t \in T$, there must be an edge $\{t_1, t_2\} \subseteq t$ with $\{t_1, t_2\} \in E'$. Note that for all $i \in [k-3]$ the edges $\{(i, t), t_1\}$ and $\{(i, t), t_2\}$ are contained in at most $k-3$ triangles in $(V_k, E_k \setminus E')$, therefore, their trussness is below k. Thus, no edge in $e \in \binom{S_t}{2} \setminus \binom{t}{2}$ can be in $k-2$ triangles of trussness k. It follows that the existence of k-trusses in $(V_k, E_k \setminus E')$ implies that of triangles in $(V_k, E \setminus E')$. However, G' and hence $(V_k, E \setminus E')$ are triangle-free. Therefore, $(V_k, E_k \setminus E')$ does not contain any k-trusses.

Suppose $G'_k = (V_k, E_k \setminus E')$ does not contain any k-trusses. Let $f : E_k \to E$ be any function such that

- $f(e) = e$ for all $e \in E$, and
- $f(e) \subseteq t$ for all $t \in T$ and $e \in \binom{S_t}{2} \setminus \binom{t}{2}$.

For each $t \in T$, the induced subgraph $G'_k[S_t]$ is not a k-truss. Hence, it is not a k-clique and there must be an edge $e_t \in E' \cap \binom{S_t}{2}$. Since $f(e_t) \subseteq t$, the triangle t does not appear in $(V, E \setminus f(E'))$. Therefore, $(V, E \setminus f(E'))$ is triangle-free.

It follows that solving MIN-3-TBS for G is equivalent to solving MIN-k-TBS for G_k. Therefore, the problem MIN-k-TBS is NP-hard. □

MIN-k-TBS is the special case of MIN-k-TCBS with $U = V$, i.e., all k-trusses are sensitive and all users do not want to belong to them. Thus, the following corollary holds:

Corollary 4.2. *For every $k \geq 3$, MIN-k-TCBS is NP-hard.*

4.2 MIN-3-TBS Is Approximable

Despite being NP-hard, we show that MIN-3-TBS admits an approximation algorithm. Towards this goal, we first present an approximation preserving reduction from MIN-3-TBS to the Ek-Vertex Cover problem, defined below, with $k = 3$.

Problem 3 (Ek-Vertex Cover (Ek-VC) [19]). *Given a k-uniform hypergraph[1] $\mathcal{H}(\mathcal{V}, \mathcal{E})$, find a smallest subset $\mathcal{V}' \subseteq \mathcal{V}$ such that, for each hyperedge $e \in \mathcal{E}$, $\mathcal{V}' \cap e \neq \emptyset$.*

Theorem 4.3. *MIN-3-TBS can be reduced to E3-VC in polynomial time.*

Proof. Given any instance $\mathcal{I}_{3-TR}$ of 3-TR, we construct an instance $\mathcal{I}_{E3-VC}$ of E3-VC as follows: For every edge $e \in E$ of G in 3-TR, we create a node $v \in \mathcal{V}$ of $\mathcal{H}$. For every 3-truss T in G, we create a hyperedge $\epsilon \in \mathcal{E}$ in H comprised of all nodes in $\mathcal{V}$ that correspond to the edges in T. This can be done in polynomial time. In the following, we will show that, given a solution $\mathcal{S}$ to $\mathcal{I}_{E3-VC}$, we can obtain a solution $\mathcal{S}'$ to $\mathcal{I}_{3-TR}$ in time polynomial in the size of $\mathcal{I}_{E3-VC}$:

($\Rightarrow$) If $\mathcal{S}$ is a solution to $\mathcal{I}_{E3-VC}$, then we can construct a set of edges $E' \subseteq E$ in polynomial time. We simply add into E' each edge corresponding to a node in $\mathcal{S}$. Since $\mathcal{S}$ is a solution of $\mathcal{I}_{E3-VC}$ the following hold: (I) Each hyperedge $\epsilon \in \mathcal{E}$ contains at least one node in $\mathcal{S}$. Thus, each 3-truss in G contains at least one edge in E'. (II) The size of $\mathcal{S}$ is minimal. Thus, the size of E' is also minimal. Due to I, II, and the observation that deleting all edges from E' removes all 3-trusses in G (since deletion of a single edge from a 3-truss breaks the 3-truss), we have that E' is a solution to $\mathcal{I}_{3-TR}$.

($\Leftarrow$) If E' is a solution to $\mathcal{I}_{3-TR}$, then we can construct a set of nodes $\mathcal{S} \subseteq \mathcal{V}$ in polynomial time, by simply adding into $\mathcal{S}$ each node in $\mathcal{V}$ corresponding to an edge in E'. Since E' is a solution to $\mathcal{I}_{3-TR}$, the following hold: (I) Each 3-truss in G contains at least one edge in E' (otherwise, the truss would not be removed by deleting all edges in E'). Thus, each hyperedge $\epsilon \in \mathcal{E}$ contains at least one node in $\mathcal{S}$. (II) The size of E' is minimal. Thus, the size of $\mathcal{S}$ is also minimal. Due to I, II, $\mathcal{S}$ is a solution to $\mathcal{I}_{E3-VC}$. □

Using the reduction and a k-approximation algorithm for Ek-VC with $k = 3$ [19], we obtain a 3-approximation for MIN-3-TBS and MIN-3-TCBS.

Corollary 4.4. *There exists a 3-approximation of MIN-3-TCBS.*

Proof. We can apply the approximation algorithm described for MIN-3-TBS to the graph (V, E_U), where E_U is the set of edges incident to U. □

4.3 MIN-k-TBS for $k > 3$ Is Inapproximable

For any fixed k, the graph property of not containing a k-truss is monotone, i.e., if G does not contain any k-truss, then no subgraph of G contains a k-truss. Moreover, no bipartite graph contains a k-truss, since bipartite graphs do not contain triangles. From Reference [2], we obtain the following additive inapproximability result, which clearly does not contradict the *multiplicative* approximability result for $k = 3$:

Theorem 4.5. *For all $\delta > 0$, it is NP-hard to approximate MIN-k-TBS up to an additive term of $|V|^{2-\delta}$.*

We will now use a reduction from Ek-VC to prove the following multiplicative inapproximability result for $k > 3$:

Theorem 4.6. *For all $\epsilon > 0, k > 3$, there is no $(k - 2 - \epsilon)$-approximation for MIN-k-TBS unless the unique games conjecture is false.*

Proof. Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a $(k-2)$-uniform hypergraph of which we want to find a vertex cover. We construct a new graph $G = (V, E)$ as follows: Let $V = \{(v, i) \mid v \in \mathcal{V}, i \in \{0, 1\}\} \cup \{(e, j) \mid e \in \mathcal{E}, j \in [k-2]\}$ and $E = \{\{(u, 0), (v, 1)\} \mid u, v \in e \in \mathcal{E}\} \cup \{\{(v, i), (e, j)\} \mid v \in e \in \mathcal{E}, i \in \{0, 1\}, j \in$

[1] A k-uniform hypergraph $\mathcal{H}(\mathcal{V}, \mathcal{E})$ consists of a set of vertices $\mathcal{V}$ and a collection $\mathcal{E}$ of k-element subsets of $\mathcal{V}$ called hyperedges.

$[k-2]\}$. Less formally, we construct an edge $\{(v,0),(v,1)\}$ for each vertex v of the hypergraph. We connect two vertices if their first coordinate shares a hyperedge and their second coordinate is different. Note that the graph so far is bipartite. We now add $k-2$ vertices for each hyperedge and connect them with all vertices corresponding to the vertices in those hyperedges. Note that we have a k-truss for each hyperedge, so we need to delete at least one edge corresponding to each hyperedge or its vertices. Thus, the number of edges to delete to break all k-trusses in G is at least the number of vertices needed to cover $\mathcal{H}$. Also note that if we delete one edge from such a truss, then the trussness of all edges of the form (e,j) in that truss drops below k. Hence, deleting the edges corresponding to a vertex cover of $\mathcal{H}$ results in a graph where all edges of the form (e,j) have trussness below k. Since the edges of the form (v,i) form a bipartite graph, the remaining graph has trussness lower than k.

Since E$(k-2)$-VC cannot be approximated within a factor $(k-2-\epsilon)$ in polynomial time, assuming the unique games conjecture [30], the same holds for MIN-k-TBS. □

Again, the inapproximability holds for the MIN-k-TCBS problem, too.

4.4 Min-2-CBS Is Solvable in Polynomial Time

We can find a minimum 2-core breaking set in polynomial time, since it is the complement of a spanning forest, which is a maximum acyclic subgraph. This is because each cycle is a 2-core and each 2-core contains a cycle.

Lemma 4.7. *MIN-2-CBS can be solved in polynomial time.*

We can solve MIN-2-CCBS analogously by contracting all components of $G[V \setminus U]$ into single vertices. This way, we break all cycles containing vertices in U by deleting only edges incident to U.

4.5 Min-k-CBS for $k > 2$ Is Inapproximable

We will now show that it is hard to even approximate a smallest k-core breaking set for $k > 2$. This also implies that MIN-k-CBS is NP-hard.

Theorem 4.8. *For all $k > 2$, there exists no polynomial time $(k-\epsilon)$-approximation algorithm for MIN-k-CBS, assuming the unique games conjecture.*

Proof. We will prove the statement using a reduction from Ek-Vertex Cover (Problem 3). Consider an instance $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ of Ek-Vertex Cover. We will construct a graph for which x is the minimum number of edges to delete to break all k-cores, if and only if x is the size of the minimum vertex cover of $\mathcal{H}$. See Figure 2.

In this new graph, we create gadgets $G(v)$ for all $v \in \mathcal{V}$ and vertices for all $e \in \mathcal{E}$. We connect $G(v)$ and the vertex in the graph corresponding to e with an edge if and only if $v \in e$ in $\mathcal{H}$. We say that a vertex/edge/subgraph is *destroyed* if it is not part of a k-core, otherwise it is *intact*. Observe that each vertex corresponding to $e \in \mathcal{E}$ has degree k in the graph, so the vertex will be destroyed if any of its neighbors is destroyed. Hence, it suffices to construct gadgets each of which satisfies the following properties:

- $G(v)$ belongs to a k-core.
- $G(v)$ can be destroyed by removing one edge in $G(v)$.
- $G(v)$ is destroyed if all its incident edges are destroyed.
- $G(v)$ is intact if none of its edges are removed and it has an intact incident edge.

Each $G(v)$ has the form of the internal nodes of a binary tree. The leaves of the binary tree correspond to the hyperedges that contain v. Each internal node of the binary tree (except the root)

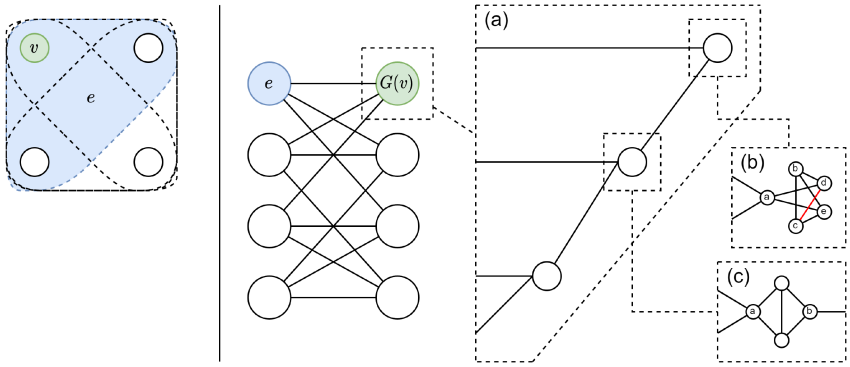


Fig. 2. On the left, an instance of $E3$-Vertex Cover. On the right, the corresponding constructed instance of 3-core breaking set: (a) corresponds to $G(v)$; (b) is a modified $(k+2)$-clique corresponding to the root (the edge (c,d) whose deletion can destroy the tree marked in red); and (c) is a modified $(k+1)$-clique corresponding to an internal node.

consists of a $(k+1)$-clique. The children of this binary tree node are attached to an arbitrary vertex a in the clique and the parent to a different vertex b in the clique. We delete the edge between a and b. The root node of the binary tree is a $(k+2)$-clique with edges (a,b), (a,c), and (d,e) deleted, where a,b,c,d,e are distinct arbitrary nodes of the clique (this is possible, because $k \geq 3$), and again a is the node connected to the children. Note that by deleting a single edge—for example, (c,d)—we can destroy the root node (see Figure 2*(b)*). Moreover, destroying a parent node destroys the children nodes, but a parent node gets destroyed only if both children are destroyed. It follows that gadget $G(v)$ has all desired properties described above.

Suppose there exists a vertex cover C of size x. Note that one can destroy the graph by deleting edge (c,d) in all gadgets $G(v)$ corresponding to each $v \in C$. Conversely, if there is a core-breaking set S of size x, then take the set of the at most x vertices of $\mathcal{H}$ whose binary trees intersect S: this must be a vertex cover, as otherwise any uncovered edge and its vertices would make a k-core.

Note that this is an approximation-preserving reduction. Moreover, there is no $(k-\epsilon)$-approximation algorithm of Ek-vertex cover by [30] assuming the unique games conjecture. It follows that there is no $(k-\epsilon)$-approximation algorithm of MIN-k-CBS either assuming the unique games conjecture. □

The inapproximability result can be easily adapted to the more general MIN-k-CCBS problem, and it implies that the problem is NP-hard.

5 EXACT ALGORITHM

An exact algorithm for MIN-k-TBS can be designed based on the following fact: A graph has a k-truss if and only if it has a minimal k-truss. As a minimal k-truss is broken by deleting any of its edges (see Lemma 3.3), we observe that an edge subset of G is a feasible solution to MIN-k-TBS if and only if it intersects *all* minimal k-trusses of G, and it is an optimal solution when it is a smallest such subset.

The above observation draws a connection between MIN-k-TBS and the *hypergraph transversal problem* [25, 35], which seeks to find a minimum transversal of a hypergraph (i.e., a smallest set of nodes of a hypergraph that intersects all its hyperedges). Specifically, let H be the hypergraph whose nodes are in one-to-one correspondence to the edges of the graph G; the hyperedges of H are all and only the minimal k-trusses of G. Clearly, an optimal solution to MIN-k-TBS is a *minimum transversal* of H.

ALGORITHM 1: MTL

Input : A graph $G = (V, E)$ and an integer $k \geq 3$.
Output : All minimal k-trusses of G.

```
1  List-min (G, ∅, ∅,k)
2  Function List-min(G, sol, x, k)
3      if Prune (G,sol,x,k) then return
4      if sol contains a k-truss then                          // using Corollary 3.2
5          if sol is a minimal k-truss then output sol
6          return
7      else
8          v ← node of minimum degree in G \ x, among those incident to sol and with at least one edge
             in E \ (sol ∪ x)
9          foreach edge e = {v, w} in E \ (sol ∪ x) do
10             List-min (G, sol ∪ {e}, x, k)
11             x ← x ∪ {e}
12 Function Prune(G, sol, x, k)
13     foreach v incident to sol do
14         if |N_{G\x}(v)| < k − 1 then return true
15     foreach e ∈ sol do
16         if |TRI_{G\x}(e)| < k − 2 then return true
17         if t_{G\x}(e) < k then return true
18     return false
```

A minimum hypergraph transversal can be found by the algorithm in Reference [35]. However, before finding such a transversal, one needs to construct H, which in turn requires listing all minimal k-trusses of G. For this task, we devised the MTL (Minimal k-Truss Listing) algorithm, presented below.

Thus, our exact algorithm for MIN-k-TBS first executes MTL on G, to construct H, and then finds a minimum transversal of H, which is an optimal solution to MIN-k-TBS. In line with the hardness of MIN-k-TBS, our exact algorithm takes in general exponential time in the number of edges $|E|$.

MTL Algorithm. Listing all minimal k-trusses is significantly harder than computing the truss decomposition, since minimal k-trusses can be exponential in number. For example, every k-clique is also a minimal k-truss.

To address this task, we thus based our MTL algorithm on the classic *binary partition* method and equipped it with pruning criteria to prevent unnecessary recursive branches and save computation time. The completeness of our algorithm easily follows from the fact that the binary partition method fully explores the space of possible solutions.

As can be seen in Algorithm 1, MTL uses a function `List-min` that is applied recursively to extend a partial solution sol and backtracks when no extension is possible. The set x keeps tracks of the elements that were tried already and should not be added to sol to prevent duplication.

`List-min` halts whenever sol *contains* a k-truss (Line 4), as sol surely does not need further extension to be a minimal k-truss, and it outputs sol only if sol *is* a minimal k-truss (Line 5). Thanks to Corollary 3.2, we can also avoid the check in Line 4 entirely when the number of edges in sol is too small to possibly create a k-truss. The check in Line 5 is performed by computing

the trussness of SOL, after the deletion of each single edge; a task made more efficient by `Truss Update`.

The `List-min` algorithm also employs the following pruning conditions:

First, SOL is extended with edges connected to it, so that it remains connected. Indeed, a minimal k-truss is necessarily connected; otherwise, its connected components would be smaller k-trusses. Furthermore, the edges that are used to extend SOL are incident to nodes of minimum degree available (Line 8). This does not affect the correctness of the binary partition method; it is a heuristic choice that increases the effectiveness of the `Prune` function described below by generating graphs with nodes of small degree whenever these edges are added to X.

Second, `List-min` uses the `Prune` function to detect recursive branches that surely cannot extend SOL to a k-truss. `Prune` checks three properties, ordered from the most efficiently computable to the most powerful in terms of pruning power:

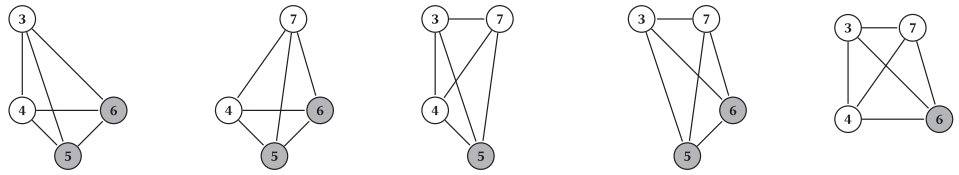
(1) There is a node v incident to SOL with degree less than $k-1$ in $G \setminus X$ (i.e., the graph obtained by deleting the edges in X) for any node incident to SOL.
(2) There is an edge $e \in$ SOL contained in less than $k-2$ triangles in $G \setminus X$ for any edge in SOL.
(3) There is an edge $e \in$ SOL with trussness less than k in $G \setminus X$ for any edge in SOL.

If any of these properties is satisfied in $G \setminus X$, then SOL cannot be extended to a k-truss, since elements of X cannot be added to SOL. In this case, `Prune` returns `true`; otherwise, it returns `false`. Furthermore, `Prune` can utilize `Truss Update` for finding the edge trussness in the current subgraph (Line 17) without explicitly computing the truss decomposition of the subgraph.

Modifications for MIN-k-TCBS. The following trivial modifications (referred to as steps) are needed to exactly solve MIN-k-TCBS: (1) Algorithm 1 is modified to produce only minimal k-trusses containing nodes in U. (2) The hypergraph H is constructed using the minimal k-trusses output by the modified Algorithm 1. (3) The algorithm in Reference [35] is modified to output a minimum hypergraph transversal whose nodes correspond only to edges incident to nodes in U. This is precisely an optimal solution to MIN-k-TCBS.

Step 1 is necessary, because *only* minimal k-trusses with nodes in U must be part of Step 2. To see this, assume that a minimal k-truss with *no* node in U is produced in Step 1. This k-truss corresponds to a hyperedge of H that is constructed in Step 2 and contains at least one node in the transversal that is output by Step 3. However, this node corresponds to an edge of G that is *not* incident to any node in U and hence cannot be contained in any optimal solution to MIN-k-TCBS. Therefore, to solve MIN-k-TCBS exactly, we only produce minimal k-trusses containing nodes in U in Step 1. Step 2 is similar to that in the MIN-k-TBS problem but only uses the minimal k-trusses containing nodes in U. This step is clearly necessary to produce the transversal. Step 3 is necessary, because *only* nodes of the hypergraph corresponding to edges that are incident to nodes in U must be contained in the transversal. To see this, assume that the transversal contains a node corresponding to an edge that is *not* incident to a node in U. This edge cannot be part of any optimal solution to the MIN-k-TCBS problem, as per the definition of the problem.

Example 5.1. Consider applying the exact algorithm for MIN-k-TCBS on the graph G of Figure 1(a) when $U = \{5, 6\}$ and $k = 4$. The minimal 4-trusses $T_1, \ldots, T_5$ containing nodes in U are shown in Figure 3, and the hypergraph H, constructed using T_1 to T_5, is shown in Figure 4. For ease of presentation, H is represented as a bipartite graph where each node in the right part: (I) corresponds to one of the minimal 4-trusses T_1 to T_5 and (II) is connected to the nodes that correspond to the edges of this minimal k-truss (the numbers above a node in the left part represent the



(a) T_1 **(b)** T_2 **(c)** T_3 **(d)** T_4 **(e)** T_5

Fig. 3. Minimal 4-trusses $T_1, \ldots, T_5$ in the graph of Figure 1(a) that contain at least one node in $U = \{5, 6\}$.

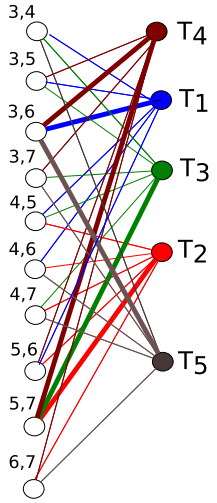


Fig. 4. Hypergraph H in Example 5.1 as a bipartite graph and a minimum transversal composed of the nodes "3, 6" and "5, 7."

endpoints of its corresponding edge).[2] For example, T_1 in Figure 3(a) is shown as the blue node in Figure 4, which is connected to the nodes labeled "3, 4," "3, 5," "3, 6," "4, 5," "4, 6," and "5, 6." These nodes correspond to the edges of T_1. The nodes "3, 6" and "5, 7" in Figure 4 correspond to edges incident in U and constitute a minimal transversal of H. Specifically, these nodes correspond to the edges $(3, 6)$ and $(5, 7)$ of G, which in turn constitute an optimal solution to MIN-4-TCBS.

6 HEURISTIC ALGORITHMS

We describe heuristic algorithms, which are based on the theoretical insights presented in Section 3. In Sections 6.1 and 6.2, we present heuristics for MIN-k-TBS that can also be trivially modified to solve MIN-k-TCBS. In Sections 6.3 and 6.4, we present heuristics for MIN-k-CBS that can also be trivially modified to solve the MIN-k-CCBS problem.

6.1 Max-truss Breaking Heuristics

Any k'-truss with $k' > k$ is also a k-truss, by definition. Thus, we need to delete edges that break every k'-truss for all $k' > k$ as well to obtain a solution to MIN-k-TBS. However, if a k-truss is not a k'-truss for any $k' > k$, then we can delete a single edge to break it, as implied by Lemma 3.4.

[2]We do not represent the other edges of G as nodes in the left part to avoid cluttering the figure, as these nodes are not contained in the output transversal.

ALGORITHM 2: MBH_S

Input : A graph $G = (V, E)$ and an integer $k \geq 3$
Output: A feasible solution E' to MIN-k-TBS

1 $G' \leftarrow G$ and $\tilde{E} \leftarrow E$
2 **while** $t(G') \geq k$ **do**
3 Let T be the max-truss of G'
4 Select an arbitrary edge e' in T with support $t(G') - 2$ in T
5 Compute the set $\text{TRI}(T, e')$ of triangles in T containing e'
6 Select an edge e: $\{e, f, g\} \in \text{TRI}(T, e')$ and e has maximum support in T
7 $\tilde{E} \leftarrow \tilde{E} \setminus \{e\}$ and $G' \leftarrow G(V, \tilde{E})$
8 **return** $E' \leftarrow E \setminus \tilde{E}$

The max-truss T of G satisfies the condition of Lemma 3.4 with $k = t(G)$. Thus, there exists *a single edge* e whose deletion breaks T. The process can be repeated until the trussness $t(G')$ of the residual graph G' falls below k, at which point we obtain a feasible solution to MIN-k-TBS. Our Max-Truss Breaking Heuristics are based on this idea. Specifically, Lemma 3.4 confirms the existence of an edge e among the edges that form triangles with an edge e' with support $t(G') - 2$ in the max-truss (see the proof of Lemma 3.4). However, Lemma 3.4 does not specify how such e may be selected. We thus explore two strategies to select an edge e for the current graph G':

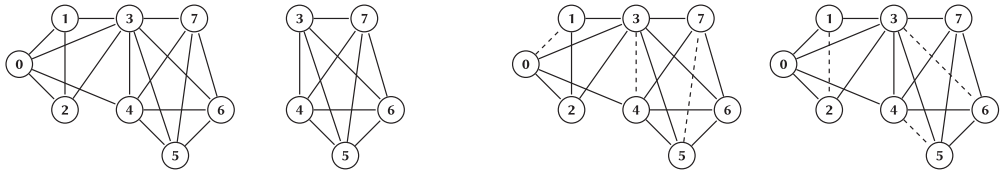
1. We select as e the edge with maximum support in the max-truss T of G', breaking ties arbitrarily. This way to select e intuitively preserves the graph size (i.e., reduces the total number of deleted edges), because the deletion of e breaks a large number of triangles that no longer appear in the max-truss of G' in subsequent iterations. We refer to this heuristic as MBH_S, where S stands for size preservation. Algorithm 2 implements this idea.

2. Let us denote by $\text{TRI}_{\geq k}(G', e)$ (respectively, $\text{TRI}_{<k}(G', e)$) the set of all triangles in G' of trussness at least k (respectively, below k) containing edge e. We select as e an edge from T with largest ratio $\frac{|\text{TRI}_{\geq k}(G', e)|}{|\text{TRI}_{<k}(G', e)|}$, breaking ties arbitrarily. This deletes edges that, on one hand, are in many triangles that inevitably have to be broken, and on the other hand, are in few triangles that do not. The former triangles are those with trussness at least k, as their existence would imply a k-truss. The latter triangles are those with trussness below k. Note that, by preserving the latter triangles, this strategy helps maintain the global clustering coefficient of the graph.[3] We refer to the heuristic employing this strategy as MBH_C, where C stands for cluster coefficient preservation. The MBH_C pseudocode is the same as that of MBH_S (Algorithm 2) except for Line 6, which is replaced by:

Select an edge e: $\{e, f, g\} \in \text{TRI}(T, e')$ and e has maximum $\frac{|\text{TRI}_{\geq k}(G', e)|}{|\text{TRI}_{<k}(G', e)|}$.[4]

Example 6.1. Consider applying MBH_S on the graph G of Figure 1(a), which is copied below for ease of reference, using $k = 4$. Observe that $t(G) = 5$. Thus, in the first iteration, the algorithm first selects the edge $(3, 7)$ as e', arbitrarily among the edges with support $5-2 = 3$ in the max-truss of G, shown in Figure 5(b). Then, it selects the edge $(3, 4)$ as e, as this edge: (I) forms a triangle with $(3, 7)$, and (II) has the maximum support in the max-truss of G (ties are broken arbitrarily). However, the deletion of the edge $(3, 4)$ from G is not enough to construct a feasible solution to MIN-4-TBS, since the residual graph G' still contains a 4-truss. Therefore, in the second iteration, MBH_S computes

[3]The global clustering coefficient quantifies the tendency of the nodes of a graph G to cluster together [37] and is defined as 3 times the ratio between number of triangles in G and number of all triplets (triangles and wedges) in G.
[4]If $|\text{TRI}_{<k}(G', e)| = 0$, then we set $|\text{TRI}_{<k}(G', e)| = 1$. With this adjustment, MBH_C tends to prioritize the removal of edges that contribute to a higher count of triangles within the k-truss while participating in fewer triangles with trussness below k.



(a) G **(b)** max-truss of G **(c)** 4-truss-free graph **(d)** 4-truss-free graph

Fig. 5. (a) Input graph G. (b) Max-truss of G. (c) The graph obtained from G after deleting the (dashed) edges, output by MBH_S, when it is applied to G using $k = 4$. (d) The graph obtained from G after deleting the (dashed) edges, output by MBH_C when it is applied to G using $k = 4$.

$t(G') = 4$ and selects the edge $(0, 2)$ as e' arbitrarily among the edges with support $4 - 2 = 2$ in the max-truss of G' (this max-truss is the graph induced by all edges of G except $(3, 4)$ and $(0, 4)$). Next, the algorithm selects the edge $(0, 1)$ as e, which forms a triangle with $(0, 2)$ and has the maximum support in the max-truss of G' (ties are broken arbitrarily). Similarly, in the third iteration, again $t(G') = 4$ and the algorithm selects the edge $(5, 6)$ as e' and the edge $(5, 7)$ as e. Since the residual graph G', which is shown in Figure 5(c), has no 4-truss, MBH_S outputs $E' = \{(3, 4), (0, 1), (5, 7)\}$. Note that MBH_S found an optimal solution, as $|E'| = 3$ is also the size of an optimal solution; see Example 1.1. The 4-truss free graph obtained after deleting the edges in E' from G is shown in Figure 5(c).

Example 6.2. Consider applying MBH_C instead of MBH_S in the setting of Example 6.1. In the first iteration, the algorithm selects the edge $(3, 5)$ as e', since it breaks ties differently from MBH_S, and the edge $(4, 5)$ as e. The latter edge is selected because it forms a triangle with $(3, 5)$ in the max-truss of G (see Figure 5(b)) and has the largest ratio, namely, 3, among all edges forming a triangle with $(3, 5)$ in the max-truss (these edges are only contained in triangles with trussness at least 4, so the denominators in their ratios are set to 1, and $(4, 5)$ is selected arbitrarily among them). Since $t(G') = 4$, MBH_C proceeds in the second iteration. In this iteration, it selects the edge $(0, 2)$ as e' (breaking ties differently from MBH_S) and the edge $(1, 2)$ as e. The latter edge is selected because it forms a triangle with the edge $(0, 2)$ in the max-truss of G' (i.e., the graph induced by all edges of G except $(0, 4)$ and $(4, 5)$) and has the largest ratio, namely, 2, among all edges forming a triangle with the edge $(0, 2)$ in the max-truss. These edges have all ratio 2, and the edge $(1, 2)$ is selected arbitrarily among them. Similarly, in the third iteration, again $t(G') = 4$ and the algorithm selects the edge $(3, 7)$ as e' and the edge $(3, 6)$ as e. Since now $t(G') < 4$, MBH_C returns $E' = \{(4, 5), (1, 2), (3, 6)\}$. As in Example 6.1, $|E'| = 3$, so MBH_C found an optimal solution. The 4-truss free graph obtained after deleting the edges in E' from G is shown in Figure 5(d).

As there may be many edges e' with support $t(G') - 2$ in the max-truss (Line 4), a variation of MBH_S or MBH_C can select as e an edge that has the largest support in the max-truss T of G' among those that form a triangle with any of those edges e', in addition to satisfying the criteria of strategy 1 or 2.

Complexity Analysis. In each iteration, MBH_S computes the trussness $t(G')$ of G', the max-truss T of G', and the support of each edge e in T. These computations take $O(|E|^{\frac{3}{2}})$ time. All triangles in T containing edge $e' = (u, v)$ are computed in time $O(|V|) = O(|E|)$. Selecting an edge e (Line 6) is then performed by traversing $\text{TRI}(T, e')$ in $O(|E|)$ time. Since MBH_S performs $r \leq |E|$ iterations, where r is the total number of edges to be removed, the total time is $O(|E|^{\frac{3}{2}} r)$ in the worst case. The only difference in MBH_C is in the computation of Line 6 that computes the ratio $\frac{|\text{TRI}_{\geq k}(G', e)|}{|\text{TRI}_{< k}(G', e)|}$ for each e that forms a triangle in $\text{TRI}(T, e')$. This computation takes time $O(|E|^{\frac{3}{2}})$ per iteration

ALGORITHM 3: SNH

Input : A graph $G = (V, E)$ and an integer $k \geq 3$
Output: A feasible solution E' to MIN-k-TBS

$G' \leftarrow G$ and $\tilde{E} \leftarrow E$
while $t(G') \geq k$ **do**
  max $\leftarrow -\infty$
  Let M be the maximal k-truss of G'
  for *each* $e \in M$ **do**
    score $\leftarrow \sum_{\{e,f,g\} \in \mathrm{TRI}_{\geq k}(M,e)} \Gamma_k(M, e, f, g)$
    if score > max **then**
      max $\leftarrow$ score; selected $\leftarrow e$
  $\tilde{E} \leftarrow \tilde{E} \setminus \{\text{selected}\}$ and $G' \leftarrow G(V, \tilde{E})$
return $E' \leftarrow E \setminus \tilde{E}$

of the while loop. Thus, $\mathrm{MBH_C}$ also takes $O(|E|^{\frac{3}{2}} r)$ time in the worst case. Plugging in the `Truss Update` and `Triangles Update` data structures speeds up the computation and yields the *improved* worst-case time of $O(|E|^{\frac{3}{2}} + |E|r + t(G)\mathcal{T}_G)$ for both heuristics, which is always bounded by $O(|E|^2)$ (see Appendix C for the details).

Modifications for MIN-k-TCBS. We modify Line 6 in Algorithm 2 to consider only edges incident to nodes in U, as required by MIN-k-TCBS. If no such edge can be selected, then we delete an edge that is incident to a node in U and has a maximum support for $\mathrm{MBH_S}$ in the maximal k-truss (or maximum ratio $\frac{|\mathrm{TRI}_{\geq k}(G',e)|}{|\mathrm{TRI}_{<k}(G',e)|}$ for $\mathrm{MBH_C}$), among all edges that are incident to nodes in U, breaking ties arbitrarily. This guarantees that no node in U belongs to a k-truss in $G(V, E \setminus E')$.

6.2 "Save the Neighbors" Heuristic

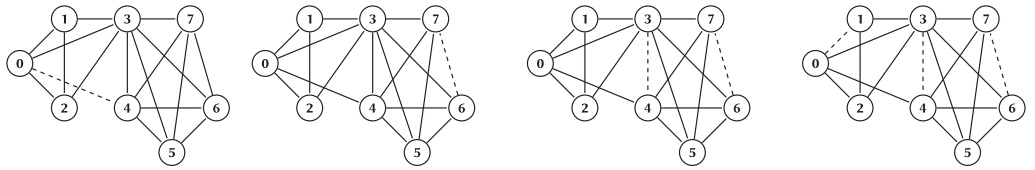
A straightforward way to construct a feasible solution to MIN-k-TBS is to iteratively delete an edge from the max-truss of the graph G until G has trussness below k. However, this heuristic may delete an unnecessarily large number of edges, whose trussness would have dropped below k by propagation anyway, in particular, by the deletion of neighboring edges. The main idea of our Save the Neighbors Heuristic (SNH) is to reduce the number of deleted edges by picking edges whose neighbors are less likely to be deleted.

Let M be a *maximal* k-truss of the current graph G', and consider a triangle $\{e, f, g\}$ in M. If e is contained in many triangles in M, then it is a good candidate for deletion. However, if f and g are contained in many more triangles in M (and are thus likely to be deleted), then the deletion of e is likely to be irrelevant in breaking the trussness of triangle $\{e, f, g\}$. In this case, f or g would be a better choice for breaking the triangle than e. We employ the utility function $\Gamma_k(M, e, f, g)$, defined as follows:

$$\frac{|\mathrm{TRI}_{\geq k}(M, e)|}{\max(|\mathrm{TRI}_{\geq k}(M, f)| - k + 2, 1)} + \frac{|\mathrm{TRI}_{\geq k}(M, e)|}{\max(|\mathrm{TRI}_{\geq k}(M, g)| - k + 2, 1)}$$

to assign a weight corresponding to our estimated importance of e in breaking triangle $\{e, f, g\}$. We then delete edges in order of maximum total weight summed over all triangles in which they are contained.

This idea aims at breaking the necessary amount of triangles in M by deleting few of its edges. Note that M is the maximal k-truss of G' (not the max-truss as in the Max-truss Breaking Heuristics). Algorithm 3 describes SNH.



(a) Maximal 4-truss of G **(b)** G' after the 1st iteration **(c)** G' after the 2nd iteration **(d)** G' after the 3rd iteration

Fig. 6. (a) The subgraph induced by all edges except the (dashed) edge $(0, 4)$ is the maximal 4-truss of G in Figure 5(a). (b) G' obtained from G in Figure 5(a) after deleting the (dashed) edge $(6, 7)$, which was selected in the 1st iteration of SNH. (c) G' obtained from the graph in Figure 6(b) after deleting the (dashed) edge $(3, 4)$, which was selected in the 2nd iteration of SNH. (d) G' obtained from the graph in Figure 6(c) after deleting the (dashed) edge $(0, 1)$, which was selected in the 3rd iteration of SNH.

Example 6.3. Consider applying SNH instead of MBH_S in the setting of Example 6.1. In the first iteration, SNH iterates over all edges of G and selects the edge $(6, 7)$ as e, since this edge has the largest score (ties are broken arbitrarily). To see how the score of the edge $(6, 7)$ is computed, observe the maximal 4-truss M of G in Figure 1(b), which is copied in Figure 6(a) for ease of reference. Clearly, each edge that forms a triangle with the edge $(6, 7)$ is contained in 3 triangles in M that have trussness at least 4. Thus, $\Gamma_4(M, (6, 7), e', e'') = \frac{3}{\max(3-4+2,1)} + \frac{3}{\max(3-4+2,1)} = 6$, for any edges e' and e'' in a triangle $\{(6, 7), e', e''\}$ in M, and the score of the edge $(6, 7)$ is equal to $\Gamma_4(M, (6, 7), (7, 4), (4, 6)) + \Gamma_4(M, (6, 7), (7, 3), (3, 6)) + \Gamma_4(M, (6, 7), (7, 5), (5, 6)) = 18$. Since the residual graph G', which is produced from G after the deletion of the edge $(6, 7)$, has trussness at least 4 (see Figure 6(b)), SNH proceeds in the second iteration. In this iteration, it selects the edge $(3, 4)$, as this edge has the largest score (ties are broken arbitrarily). Again, G' has trussness at least 4 (see Figure 6(c)), so the algorithm selects the edge $(0, 1)$ in the third iteration. Now, G' has trussness less than 4 (see Figure 6(d)), so SNH outputs $E' = \{(6, 7), (3, 4), (0, 1)\}$. Note that SNH found an optimal solution, since $|E'| = 3$ is also the size of an optimal solution; see Example 1.1.

Complexity Analysis. In each iteration, SNH computes the trussness $t(G')$ of G' and M (the maximal k-truss of G'). These computations take $O(|E|^{\frac{3}{2}})$ time. Computing all triangles (of trussness at least k) in M takes $O(|E|^{\frac{3}{2}})$ time. Based on these and the utility function $\Gamma_k(M, e, f, g)$, SNH then evaluates the formula in Line 6. The evaluation of the formula over all edges in M takes $O(|E| + \mathcal{T}_G)$ time, because each triangle contains $3 = O(1)$ edges (thus, it is evaluated three times in the sum of Line 6) and there are $\mathcal{T}_G$ triangles. Since SNH performs $r \leq |E|$ iterations, where r is the total number of edges to be removed, and $\mathcal{T}_G = O(|E|^{\frac{3}{2}})$, SNH takes $O(|E|^{\frac{3}{2}} r)$ time in the worst case. Plugging in the `Truss Update` and `Triangles Update` data structures constructed on M speeds up the computation and yields the worst-case time of $O(|E|^{\frac{3}{2}} + |E|r + t(G)\mathcal{T}_G + \mathcal{T}_G \cdot r)$. This bound is larger than the improved bound of MBH_S and MBH_C by an additive term $\mathcal{T}_G \cdot r$, because SNH considers up to $\mathcal{T}_G$ triangles in each of the r iterations.

Modifications for MIN-k-TCBS. The selected edge e must also be incident to a node in U.

6.3 Coreness Propagation Heuristic

Our Coreness Propagation Heuristic (CPH) is based on two ideas: (1) The max-core of G can be broken by deleting a single edge, as it satisfies the condition of Lemma 3.5. Thus, we can obtain a feasible solution to MIN-k-CBS by first deleting an edge e from the max-core of G and then repeating this process until the coreness $c(G')$ of the residual graph G' falls below k. (2) The deletion of e can reduce the coreness of other edges whose nodes will no longer belong in the core (see

ALGORITHM 4: CPH

Input : A graph $G = (V, E)$ and an integer $k \geq 2$
Output: A feasible solution E' to MIN-k-CBS

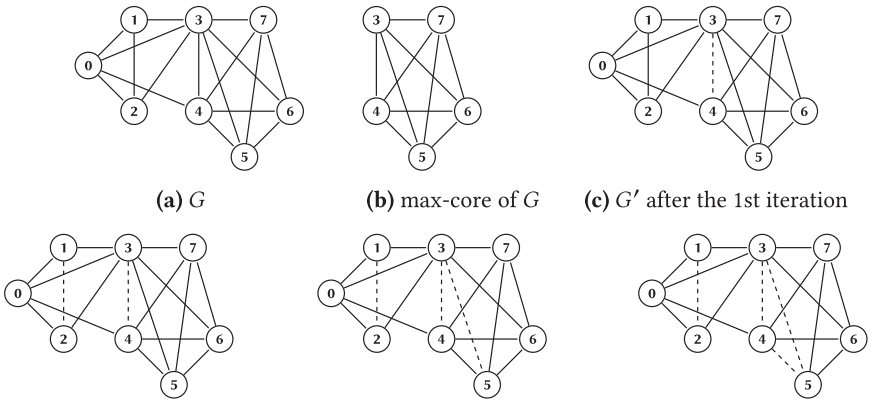
1 $G' \leftarrow G$ and $\tilde{E} \leftarrow E$
2 **while** $c(G') \geq k$ **do**
3 max$\leftarrow -\infty$
4 $\rho \leftarrow c(G')$
5 Let C be the max-core of G'
6 **for** *each* $e = (u, v) \in C$ *s.t.* $\mathbb{I}_C(u) + \mathbb{I}_C(v) > 0$ **do**
7 score $\leftarrow \mathbb{I}_C(u) + \mathbb{I}_C(v)$
8 score $\leftarrow$ score $+ |\{w \in N_C(u) \wedge \mathbb{I}_C(w) = 1 \wedge \mathbb{I}_C(u) = 1\} \cup \{w \in N_C(v) \wedge \mathbb{I}_C(w) = 1 \wedge \mathbb{I}_C(v) = 1\}|$
9 **if** score $>$ max **then**
10 max $\leftarrow$ score; selected $\leftarrow e$
11 $\tilde{E} \leftarrow \tilde{E} \setminus \{\text{selected}\}$ and $G' \leftarrow G(V, \tilde{E})$
12 **return** $E' \leftarrow E \setminus \tilde{E}$

Section 3.5). Thus, to reduce the total number of deleted edges, we should delete edges leading to the deletion of many nodes from the max-core of G'.

Specifically, we select an edge e that will be deleted, as follows: Let C be the max-core of G', $\rho = c(G')$, and $\mathbb{I}_C(u)$ be an indicator function that returns 1 if u has degree ρ in C (i.e., $|N_C(u)| = \rho$), and 0 otherwise. We give each edge (u, v) in the max-core of G' with $\mathbb{I}_C(u) + \mathbb{I}_C(v) > 0$ a score $\mathbb{I}_C(u) + \mathbb{I}_C(v)$. Then, we go over each $w \in N_C(u) \cup N_C(v)$ with $\mathbb{I}_C(w) = 1$ and add 1 to the score of (u, v) only if w has a neighbor u (or v) with $\mathbb{I}_C(u) = 1$ (or $\mathbb{I}_C(v) = 1$).[5] Thus, the score of (u, v) corresponds to $|A| + |B|$, where $A \subseteq \{u, v\}$ is the set of nodes whose degree in the max-core after deleting edge (u, v) drops below ρ and B is the set of nodes that are neighbors of the node(s) in A and whose degree ρ in the max-core drops when leaving out some node in A (e.g., if the degree of u drops below ρ by deleting (u, v), then we want to count which nodes have their degree dropped below ρ when removing all edges incident to u). We thus select as e the edge with the largest score, breaking ties arbitrarily. Note that Lemma 3.5 guarantees that an edge can be deleted in each iteration, since at least one node in the max-core of G' will have degree ρ in the max-core. Algorithm 4 implements this idea.

Example 6.4. Consider applying CPH on the graph G of Figure 1(a), which is copied in Figure 7(a) below for ease of reference, using $k = 3$. Since G has a 4-core, CPH computes the score of each edge in the max-core C of G, shown in Figure 7(b). Then, it selects the edge $(3, 4)$, which has the largest score (ties are broken arbitrarily). The score of this edge is 7. This is because $\mathbb{I}_C(3) = 1$, $\mathbb{I}_C(4) = 1$, $N_C(3) = \{4, 5, 6, 7\}$, $N_C(4) = \{3, 5, 6, 7\}$, and $\mathbb{I}_C(w) = 1$ for each node $w \in \{3, 4, 5, 6, 7\}$, which implies that the score is $\mathbb{I}_C(3) + \mathbb{I}_C(4) = 1 + 1 = 2$ the first time Line 7 of Algorithm 4 is executed and increases by 5 the first time Line 8 of Algorithm 4 is executed. Since the residual graph G', which is produced from G after the deletion of the edge $(3, 4)$, has coreness at least 3 (see Figure 7(c)), CPH performs a second iteration. In this iteration, it selects the edge $(1, 2)$, whose score is the maximum among all edges in the max-core of G' (which is G' itself, as all nodes have degree at least 3), breaking ties arbitrarily. Similarly, in the third iteration, CPH selects the edge $(3, 5)$ from the max-core of the graph G', which is shown in Figure 7(d); the max-core is induced

[5] An alternative is to add $|\{w \in N_C(u) \wedge \mathbb{I}_C(w) = 1 \wedge \mathbb{I}_C(u) = 1\}| + |\{w \in N_C(v) \wedge \mathbb{I}_C(w) = 1 \wedge \mathbb{I}_C(v) = 1\}|$. This is faster to compute but performed poorly in practice.



(a) G **(b)** max-core of G **(c)** G' after the 1st iteration

(d) G' after the 2nd iteration **(e)** G' after the 3rd iteration **(f)** G' after the 4th iteration

Fig. 7. (a) Input graph G. (b) Max-core of G. (c) G' obtained from G after deleting the (dashed) edge $(3, 4)$, which was selected in the 1st iteration of CPH. (d) G' obtained from the graph in Figure 7(c) after deleting the (dashed) edge $(1, 2)$, which was selected in the 2nd iteration of CPH. (e) G' obtained from the graph in Figure 7(d) after deleting the (dashed) edge $(3, 5)$, which was selected in the 3rd iteration of CPH. (f) G' obtained from the graph in Figure 7(e) after deleting the (dashed) edge $(4, 5)$, which was selected in the 4th iteration of CPH.

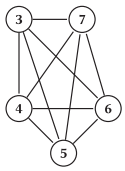
by the edges $(3, 5), (3, 6), (3, 7), (4, 5), (4, 6), (4, 7), (5, 6), (5, 7)$, and $(6, 7)$. Since the residual graph G', produced by deleting the edge $(1, 2)$, has still a 3-core, in the fourth iteration, CPH selects the edge $(4, 5)$ from the max-core of the graph G', which is shown in Figure 7(e). Since the deletion of this edge leads to G' in Figure 7(f), which has coreness below 3, CPH outputs the set of edges $E' = \{(3, 4), (1, 2), (3, 5), (4, 5)\}$ that have been selected.

Complexity Analysis. In each iteration, CPH computes the coreness $c(G')$ of G', the max-core of G', and the coreness of each edge in G'. These computations take $O(|V| + |E|) = O(|E|)$ time. Selecting an edge from the max-core of G' takes $O(|E|c(G'))$ time. With a breadth-first search (BFS) on G', we can identify each node u with $\mathbb{I}_C(u) > 0$ (since we know $c(u)$) and then iterate over the edges of G'. Since we consider only nodes with degree $\rho = c(G')$ in G' and computing score takes $O(c(G'))$ time, we can compute all edge scores in $O(|E|c(G'))$ time. CPH performs $r \leq |E|$ iterations, where r is the total number of edges to be removed and $c(G') \leq c(G) \leq |V|$. Thus, it takes $O(|E|c(G)r) = O(|E|^2|V|)$ time in the worst case.

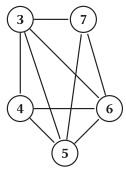
Plugging in the `Coreness Update` and `Core Decomposition Update` data structures yields the worst-case time of $O(|E| + |E|c(G)r + |E|c(G)) = O(|E|c(G)r)$. Specifically, the CPH algorithm needs $O(|V| + |E|) = O(|E|)$ time to construct the data structures, $O(|E|c(G))$ time to compute the score of edges in each iteration, using `Coreness Update`, and amortized $O(|E|c(G))$ time across all edge deletions to update G' by maintaining the data structures.

Modifications for MIN-k-CCBS. We modify Line 6 in Algorithm 4 to consider only edges incident to nodes in U and Line 8. If no such edge can be selected, then we find a node u with: (1) the largest coreness among all nodes incident to U and (2) the smallest number of incident edges among these nodes. Then, we delete an arbitrarily selected edge of u. This is to make it easier for the node to be taken out of the core in which the node is contained[6] in subsequent iterations.

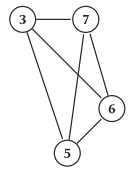
[6]Note that u may not be contained in the max-core of G', but it has to be contained in a k'-core with $k' > k$.



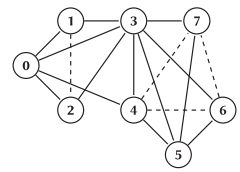
(a) Maximal 3-core of G' after the 1st iteration



(b) Maximal 3-core of G' after the 2nd iteration



(c) Maximal 3-core of G' after the 3rd iteration



(d) G' after the 3rd iteration

Fig. 8. (a) Maximal 3-core obtained from G', which is in turn obtained from G in Figure 5(a), after deleting the edge $(1, 2)$. (b) Maximal 3-core of G', which is in turn obtained by deleting the edge $(4, 7)$ from the graph in Figure 8(a). (c) Maximal 3-core of G', which is in turn obtained by deleting the edge $(4, 6)$ from the graph in Figure 8(b). (d) G' obtained from the graph G in Figure 5(a) after deleting the (dashed) edges that are output by SNH.

6.4 Small Degree Neighborhood Heuristic

We can construct a feasible solution to MIN-k-TCBS by iteratively deleting an edge from the maximal k-core of G until the coreness of G falls below k. We select an edge to delete based on two criteria that aim to minimize the total number of deleted edges: (1) We favor the deletion of an edge (u, v) whose endpoints u and v have small degrees in the maximal k-core. This is because the nodes of such an edge are easier to be removed from the maximal k-core in the current or subsequent iterations. Specifically, if the degree of u in the maximal k-core C is the smallest possible (i.e., $|N_C(u)| = k$) prior to the edge deletion, then u will be removed from C and also from *any* other k-core. Otherwise, u will still be contained in a maximal k-core, but its new degree will be reduced by 1. Hence, if u has small $|N_C(u)|$ prior to the edge deletion, then it will be closer to being removed from the maximal k-core in subsequent iterations. (2) We favor the deletion of an edge (u, v) where many neighbors of u and/or v in the maximal k-core C (i.e., nodes in $N_C(u) \cup N_C(v)$) have themselves a small degree in C. The reason is similar to criterion 1: Any $w \in N_C(u) \cup N_C(v)$ with $N_C(w) = k$ prior to the deletion will be removed from C and any other k-core, while any other w will be more easily removed from the maximal k-core in subsequent iterations.

These two criteria form the basis of our "Small Degree Neighborhood" Heuristic (SDNH), which iteratively deletes the edge (u, v) in the maximal k-core C having a smallest score

$$s((u, v)) = |N_C(u)| \cdot |N_C(v)| \cdot \frac{\sum_{w \in N_C(u) \cup N_C(v)} |N_C(w)|}{|N_C(u) \cup N_C(v)|}, \tag{1}$$

until there is no k-core left in the residual graph.[7]

Example 6.5. Consider applying SDNH instead of CPH in the setting of Example 6.4. Since G has a 3-core, SDNH computes the score for each edge in the maximal 3-core of G (which is G itself, since every node in G has degree at least 3), and it selects the edge $(1, 2)$. This is because this edge has the smallest score (ties are broken arbitrarily). Specifically, $N_C(0) = \{1, 2, 3, 4\}$, $N_C(1) = \{0, 2, 3\}$, $N_C(2) = \{0, 1, 3\}$, and $N_C(3) = \{0, 1, 2, 4, 5, 6, 7\}$ and thus $s((1, 2)) = |N_C(1)| \cdot |N_C(2)| \cdot \frac{\sum_{w \in N_C(1) \cup N_C(2)} |N_C(w)|}{|N_C(1) \cup N_C(2)|} = |\{0, 2, 3\}| \cdot |\{0, 1, 3\}| \cdot \frac{\sum_{w \in \{0,1,2,3\}} |N_C(w)|}{|\{0,1,2,3\}|} = 3 \cdot 3 \cdot \frac{4+3+3+7}{4} = 38.25$. Since there is still a 3-core left in the graph G', produced from G after deleting the edge $(1, 2)$, SDNH performs a second iteration. In this iteration, it selects the edge $(4, 7)$, whose score is the smallest among all

[7] An alternative is to use $|N_C(u)| + |N_C(v)|$ in the denominator. This is faster to compute but performed poorly in practice.

edges in the maximal 3-core of G', shown in Figure 8(a) (ties are broken arbitrarily). Similarly, in the third (respectively, fourth) iteration, SDNH selects the edge $(4, 6)$ from the maximal 3-core of G' in Figure 8(b) (respectively, the edge $(6, 7)$ from the maximal 3-core of G' in Figure 8(c)). Since at this point G' in Figure 8(d) has no 3-core, SDNH outputs the set of edges $E' = \{(1, 2), (4, 7), (4, 6), (6, 7)\}$ that have been selected.

Thus, SDNH differs from CPH along three important dimensions. First, it is applied to the maximal k-core and not the max-core of a graph. Second, it considers also nodes with degree larger than the level of coreness we aim to break. Third, it selects edges for deletion based on the actual number of neighbors of a node in the k-core and not just based on whether this number exceeds a threshold.

Complexity Analysis. In each iteration, SDNH computes the maximal k-core C of G', and it also performs a BFS on G' in $O(|V| + |E|) = O(|E|)$ time to compute for each u in C: (I) $|N_C(u)|$ and (II) $\sum_{w \in N_C(u)} |N_C(w)|$. In addition, it lists the triangles in C in $O(|E|\, c(G))$ time to compute for each edge (u, v): (III) the number $|N_C(u) \cap N_C(v)|$ of triangles in C containing (u, v), and (IV) the value of $\sum_{w \in N_C(u) \cap N_C(v)} |N_C(w)|$. The triangles are found as described in Reference [16] by examining the nodes u in degeneracy order and listing the triangles $\{u, u', u''\}$ where $u', u'' \in \{x \in N_C(u) \mid x > u\}$ are linked by an edge in C. We observe that (I)–(IV) allow for computing the numerator and the denominator in Equation (1) for (u, v) in $O(1)$ time. In particular, the numerator can be rewritten as $\sum_{w \in N_C(u)} |N_C(w)| + \sum_{w \in N_C(v)} |N_C(w)| - \sum_{w \in N_C(u) \cap N_C(v)} |N_C(w)|$ based on the inclusion/exclusion principle. Similarly, the denominator can be rewritten as $|N_C(u)| + |N_C(v)| - |N_C(u) \cap N_C(v)|$ by the inclusion/exclusion principle. Thus, computing the score $s((u, v))$ for each edge (u, v) and selecting an edge with smallest score takes $O(|E|\, c(G))$ time. Since SDNH performs $r \leq |E|$ iterations, where r is the total number of edges to be removed, it takes $O(|E|\, c(G)\, r)$ time in the worst case (recall that $c(G) = O(E^{1/2})$ is typically small in real-world graphs).

Let us note that plugging in the `Coreness Update` and `Core Decomposition Update` data structures does not improve the worst-case bound but saves time in practice.

Modifications for MIN-k-TCBS. The selected edge e must also be incident to a node in U.

7 LOWER BOUND ON THE SIZE OF OPT

Let OPT be an optimal solution to MIN-k-TBS. Due to the exponential time-complexity of our exact algorithm (Section 5), computing OPT is a heavy task even for small graphs with few hundreds of nodes. We design an algorithm for computing a lower bound on the size |OPT| of OPT. Clearly, the bound also extends trivially to MIN-k-TCBS when $U = V$.

Our main idea is to use cliques as a "proxy" for trusses. Since a k-clique is a k-truss, we must at the very least make the input graph G free from k-cliques to solve MIN-k-TBS.

A first idea is to apply Turan's theorem [11]: A graph with n nodes and *no* clique of size k or more cannot have more than $\frac{k-2}{k-1}\frac{n^2}{2}$ edges, thus it must be "missing" at least $\binom{n}{2} - \frac{k-2}{k-1}\frac{n^2}{2}$ edges. Turan's theorem is unlikely to be useful if applied directly to G, but it will always give us a positive lower bound of edges to delete if applied to a clique of size *at least* k.

We thus devise an algorithm, called LB (for Lower Bound), which works in three phases:

(1) Computes an *edge clique partition* of G, defined below, to obtain a collection of edge-wise disjoint cliques.
(2) Applies the best available lower bound on each clique.
(3) Outputs a lower bound on |OPT| by summing the bounds of the cliques. This is possible, because the cliques are all edge-wise disjoint.

Although LB does not provide a tight lower bound, it provides a bound that is close to |OPT| (see Section 9) and hence serves as a good reference point for evaluating the effectiveness of our heuristics. Below, we detail the phases of LB.

Computing an Edge Clique Partition. An edge clique partition (ECP) of a graph G is a collection of cliques of G such that any two cliques do not share edges (they may share a single node), and each edge of G is contained in one of the cliques. A trivial ECP is given by the set of edges of G, but to get a good lower bound, we want an ECP with few large cliques rather than many small ones. While minimizing the number of cliques is famously NP-complete [28], the authors of Reference [18] recently introduced a fast and flexible framework for the related *edge clique cover* problem (where cliques are allowed to overlap): one algorithm from this framework, called "pivoting" (see Table 2 in Reference [18]), is aimed precisely at finding covers with large cliques. We take this algorithm and adapt it to our needs by simply deleting each clique from G as soon as it is found. Since the deleted edges in this clique cannot be placed in other cliques by the algorithm, we obtain an ECP.

Lower Bounding the Edges to Remove from Each Clique. Turan's theorem, as mentioned above, immediately provides a lower bound. This is, however, far from tight, as graphs without large cliques may still have high trussness (e.g., a complete 3-partite graph has no 4-cliques but can have trussness up to $n/3 + 2$). To get a finer bound, we can employ Theorem 3.6, which implies a graph with m edges and T triangles has trussness at least $\frac{T}{m} + 2$. We combine this with known lower bounds on triangles from References [22, 38] and Reference [11, Corollary 6.1.8], for a graph with n nodes and m edges, which are synthesized below:

1. If $m \leq n^2/4$, then $T \geq 0$ [11].
2. If $n^2/4 \leq m \leq n^2/3$, then $T \geq \frac{9mn - 2n^2 - 2(n^3 - 3m)^{3/2}}{27}$ [22].
3. If $n^2/4 \leq m \leq \lfloor n^2/4 \rfloor + \lfloor n/2 \rfloor$, then $T \geq (m - \lfloor n^2/4 \rfloor)\lfloor n^2/2 \rfloor$ [38].
4. If $m \geq n^2/3$, then a lower bound for T is obtained by building the piece-wise linear function interpolating the points given by integer $y = 2, 3, \ldots$ in $m = (y-1)n^2/2y$, $T = (4m - n^2)m/3n$, and computing the interpolated value of T corresponding to the specific required m [11].

Given a graph with n nodes and m edges, we use the above formulas to get a lower bound on the number T of triangles (if more than one formula applies, then we get the largest lower bound among those of all these formulas). Then, Theorem 3.6 implies that the trussness of this graph is at least $\frac{T}{m} + 2$: By taking the *highest* number $m_{\max}$ of edges for which the resulting trussness is less than k, we know that a graph of n nodes must have *no more than* $m_{\max}$ edges (but possibly fewer) if its trussness is less than k. If we have a clique with n nodes and $\binom{n}{2}$ edges, then this means we must remove from it at least $\binom{n}{2} - m_{\max}$ edges.

Given a clique of size at least k from the ECP, we use as lower bound the minimum of the lower bound computed by Turan's theorem and that computed by Theorem 3.6.

Computing the Lower Bound on |OPT|. As the cliques are all edge-wise disjoint, we sum the bounds obtained in the previous phase and output the sum as a lower bound on |OPT|.

Complexity Analysis. The time complexity of LB is dominated by the time of the "pivoting" algorithm in Reference [18]. A straightforward analysis of the latter algorithm yields an $O(q\Delta^2|E|)$ time bound, where q, Δ, and $|E|$ is the size of the largest clique, the highest degree, and the number of edges in G, respectively.

8 RELATED WORK

The notions of k-truss [15] and k-core [42] are appealing both theoretically and practically (see Section 1). In fact, there are many works that aim to detect a maximal k-truss for each k (e.g.,

Reference [16]) or a k-truss containing certain nodes and/or attributes (e.g., References [26, 27]). Similarly, there are many works on detecting a maximal k-core for each k (e.g., References [34, 46]) or a k-core containing certain nodes and/or attributes (e.g., Reference [51]). There is also much work on extending the notion of k-truss to capture application-specific requirements [14, 20, 27, 44]. For example, Reference [44] proposed a variant of k-truss in which some specified nodes must not be contained. The notion of k-core has also been extended to capture application-specific requirements [12, 24, 43, 45, 51]. For example, References [12] and [45] proposed notions of k-core that consider distance between nodes and information diffusion, respectively.

Several recent works studied how to modify the community structure of a graph based on the concept of k-truss [49, 52, 55] or k-core [10, 32, 33, 49, 50, 53, 54]. All these works consider fixed-budget problems, where the goal is to modify the maximal k-truss or k-core of a graph by adding or deleting a *fixed* number of edges or nodes, according to some criterion relevant to the maximal k-truss or k-core of the input graph. Among k-truss-based works, Reference [52] seeks to find a fixed number of nodes to retain in a specific type of maximal k-truss so as to maximize its number of nodes; Reference [49] seeks to delete a fixed number of nodes from the maximal k-truss so as to minimize its number of nodes; and Reference [55] seeks to delete a fixed number of edges from the maximal k-truss so as to minimize its number of edges. Among k-core-based works, References [10, 32] seek to add a fixed number of nodes into a specific type of maximal k-core so as to maximize its number of nodes; Reference [53] seeks to add a fixed number of edges into the maximal k-core so as to maximize the number of its nodes; and Reference [50] (respectively, Reference [54]) seeks to delete a fixed number of nodes (respectively, edges) from the maximal k-core so as to minimize its number of nodes.

Importantly and unlike these works, we consider problems that are not specific to the maximal k-truss or k-core, but rather consider *all* k-trusses or k-cores, or alternatively all those containing pre-specified nodes (i.e., nodes in U). This task is inherently more difficult due to the nested structure of k-trusses or k-cores. That is, a k-truss may contain exponentially many smaller k-trusses, and the same applies to a k-core. Furthermore, we consider problems that seek to find a global-optimum solution and not a fixed-budget solution like the aforementioned works.

9 EXPERIMENTS

We experimentally evaluate our methods for breaking k-truss- and k-core-based communities in Sections 9.1 and 9.2, respectively. Then, in Section 9.3, we present a case study on a co-authorship network in which we use our methods for breaking k-trusses and those for breaking k-cores. Last, we examine how well our methods preserve modularity in Section 9.4. Our source code is available at https://bitbucket.org/breakingtruss/tkdd24/

9.1 k-Truss-based Communities

We compared our heuristics to our exact algorithm and the lower bound, as well as to two natural baselines, in terms of effectiveness and efficiency. We focus on the MIN-k-TBS problem. Our algorithms for the MIN-k-TCBS problem performed similarly, as MIN-k-TBS is a special case of MIN-k-TCBS with $U = V$ and thus much more challenging computationally. Recall that, in Section 2, we showed results for MIN-k-TCBS using a real dataset.

Experimental Datasets and Setup. We used 11 real-world datasets of differing characteristics and from different domains (see Table 1). The first 5 datasets are available from http://networkrepository.com; FL from https://sites.google.com/site/yangdingqi/home/foursquare-dataset; and all other datasets from https://snap.stanford.edu/. We also used 1,000 synthetic datasets with 30 nodes and 84 edges each, generated using the Albert-Barabasi model.

Table 1. Characteristics of Real Datasets

Dataset	Domain	# Edges	# Nodes	$t(G)$	$c(G)$	Max degree	Avg degree
TRIBES	Social	58	16	5	5	10	7
KARATE	Social	78	34	5	4	17	4
DOLPHINS	Social	159	62	5	4	12	5
NETSCIENCE	Collab.	914	379	9	8	34	4
JAZZ	Collab.	2,724	198	30	29	100	27
WIKI	Web	100,761	8,298	23	53	1,065	24
EPINIONS	Social	405,739	75,888	33	67	3,044	10
FL	Social	607,327	114,324	30	55	1,755	10
M14B	Simulation	1,679,018	214,765	7	9	40	15
DBLP	Collab.	1,871,070	511,163	115	114	576	11
AMAZON	E-comm.	2,439,436	410,236	11	10	2,760	7

We compared our heuristics to two natural baselines:

ATk (for All Trussness $\geq k$): It deletes all edges of trussness at least k. Clearly, ATk finds a feasible solution to MIN-k-TBS, since it suffices to delete all edges identified by ATk to solve MIN-k-TBS, but it does not consider the impact of an edge deletion on the trussness of other edges. Thus, we compared against ATk to show how many edges are "saved" by our heuristics. ATk is very fast. It takes $O(|E|^{\frac{3}{2}})$ time, as it only requires computing the trussness decomposition of G once.

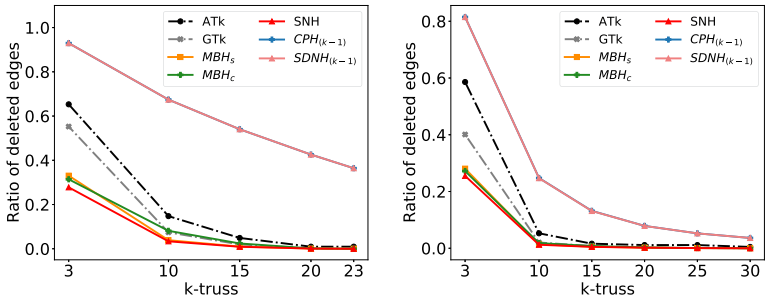
GTk (for Greedy Trussness $\geq k$): GTk is the baseline that motivated SNH (see Section 6.2). That is, GTk iteratively deletes the edge with the highest trussness, breaking ties arbitrarily, until there is no k-truss in the graph. GTk requires $O(|E|^{\frac{3}{2}}r)$ time to delete r edges, since it computes the truss decomposition after every iteration. Thus, it is expected to be much slower than ATk. However, GTk identifies substantially fewer edges to delete than ATk, because it considers the impact of deleting an edge to the trussness of other edges. We compared against GTk to show the effectiveness of our heuristics and the efficiency impact of our data structures.

We also compared our heuristics to the exact algorithm denoted by OPT (see Section 5) and the lower bound algorithm denoted by LB (see Section 7) to rigorously assess the effectiveness/efficiency trade-offs offered by our heuristics. We do not report the runtime of OPT and LB, because the former is impractical for the size of real-world graphs and the latter is much faster than our methods; it should be clear that LB does not construct feasible solutions to our problems.

To measure effectiveness, we used: (1) the ratio of deleted edges $\frac{|E'|}{|E|}$; and (2) the relative error $\text{RE} = \frac{C_G - C_{G'}}{C_G}$, where C_G (respectively, $C_{G'}$) is the global clustering co-efficient of G (respectively, G') [37]. Measures 1 and 2 capture damage to the structure of the output graph caused by edge deletion and by the loss of clusters, respectively.

We implemented all evaluated methods and executed them on an AMD EPYC 7282 @ 2.8 GHz with 252 GB RAM. We omit the results of the variations of MBH discussed in Section 6.1, as they performed similarly to MBH_S and MBH_C but were much slower. We also omit the versions of our heuristics that do not employ `Truss Update`, as they were more than one order of magnitude slower. In our implementations, we used the algorithm of Reference [16] to compute the truss decomposition.

Why We Need k-Truss-based Heuristics. Lemma 3.1 implies that a graph with no $(k-1)$-core (and hence no node with degree at least $k-1$) has no k-truss. Thus, a feasible solution to MIN-$(k-1)$-CBS is also a feasible solution to MIN-k-TBS. However, breaking the k-trusses of a graph



(a) WIKI **(b)** FL

Fig. 9. Ratio of deleted edges, for varying k, on large graphs. Note that applying the core-based heuristics, CPH and SDNH, with $k-1$ deleted a much larger ratio of edges to break the k-trusses, compared to all truss-based methods.

Table 2. Statistics for the Ratio Between the Number of Deleted Edges by a Heuristic and by the Exact Algorithm (i.e., in an Optimal Solution) on 1,000 Synthetic Graphs

	MBH_S	MBH_C	SNH
min	1	1	1
mean	1.07	1.064	1.043
median	1.062	1.059	1.055
max	1.267	1.278	1.25
st. dev.	0.054	0.052	0.044

(a) $k = 3$

	MBH_S	MBH_C	SNH
min	1	1	1
mean	1.15	1.251	1.018
median	1	1	1
max	2	3	1.5
st. dev.	0.265	0.361	0.085

(b) $k = 4$

by breaking all its $(k-1)$-cores requires deleting many more edges in practice, as can be seen in Figure 9: For example, applying either of our core breaking heuristics with $k-1$ led to the deletion of up to 45 times more edges compared to ATk, the worst-performing baseline for MIN-k-TBS in terms of effectiveness. Similar results were obtained from the other datasets (omitted). Therefore, it is highly preferable to deal with MIN-k-TBS using dedicated truss-based heuristics, rather than methods that are designed for breaking $(k-1)$-cores.

Effectiveness on Small Graphs. We show that our heuristics find near-optimal solutions (close to OPT) and also that the lower bound computed by our LB algorithm is not far from OPT. This can be seen in Table 2 and Figure 10, which show statistics for the ratio of deleted edges for synthetic and real graphs, respectively. On synthetic graphs, our heuristics deleted at most 7% more edges than the optimal, on average (see Table 2a). On real graphs, the results are similar (see Figure 10). SNH is the best-performing heuristic, which shows the effectiveness of its strategy for avoiding unnecessary edge deletion. MBH_S and MBH_C also performed very well, with the former being able to delete fewer edges, as it considers solely the support of edges in the max-truss. As expected by its design that considers triangles of all trussness values, MBH_C outperformed the other heuristics in terms of RE (see Appendix D).

Effectiveness on Large Graphs. We show that our heuristics are fairly close to the lower bound, which implies that they are even closer to the optimal solution. In addition, our heuristics substantially outperform both baselines, particularly for small k values (see Figure 11). Again, SNH outperformed MBH_S and MBH_C, with MBH_S being slightly better than MBH_C, as before. As expected, our exact (exponential-time) algorithm did not terminate in 24 hours in these experiments, and so its results are omitted.

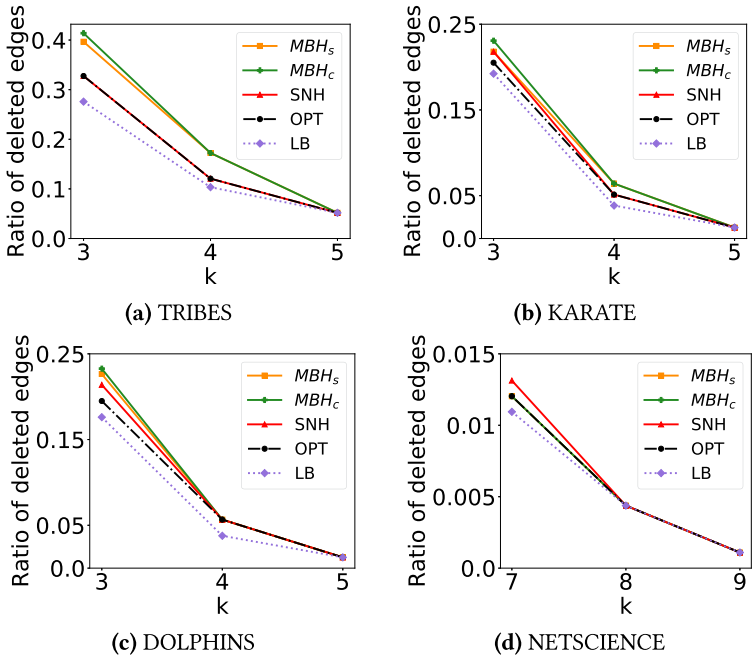


Fig. 10. Ratio of deleted edges, for varying k, on small graphs. Note that SNH essentially coincides with OPT (optimal solution) and that SNH is close to our LB (lower bound). Further note that LB is not far from OPT.

We also show that MBH_C preserves the global clustering coefficient better than the other heuristics and baselines in Table 3. These results suggest that, when performed carefully, edge deletion does not substantially affect the clustering structure of the graph. This is useful when one wants the output graph to be published for analysis (e.g., in A4 in Introduction).

Efficiency. We show that our heuristics are one to two orders of magnitude faster than GTk (see Figure 12), despite outperforming it in terms of quality (see Figure 11). For example, we were unable to run GTk on M14B or AMAZON within 24 hours. The reason is that our heuristics employ `Truss Update` and the `Triangles Update` data structure, instead of the expensive truss decomposition procedure employed by GTk. MBH_S is faster than MBH_C, since MBH_C also considers triangles with trussness below k, as well as than SNH, as the Γ_k function considers all triangles (of trussness at least k) in the maximal k-truss. As expected, ATk is the fastest method, because it does not need to recompute the trussness of edges after edge deletion; recall that it is by far the worst in terms of effectiveness (Figure 11). Of note, LB took less than 10 seconds in any case, thus providing a quick assessment tool for the user, as noted in Introduction.

9.2 k-Core-based Communities

We compared our heuristics to two natural baselines. We focus on the MIN-k-CBS problem; recall that, in Section 2, we showed results of MIN-k-CCBS on a real dataset. Our algorithms for the MIN-k-CCBS problem performed similarly, as MIN-k-CBS is a special case of MIN-k-CCBS with $U = V$ and thus much more challenging computationally.

Experimental Datasets and Setup. We used the large graphs (i.e., the last seven datasets of Table 1) and compared our heuristics to two natural baselines, along the lines of ATk and GTk, the two baselines we have developed for k-trusses:

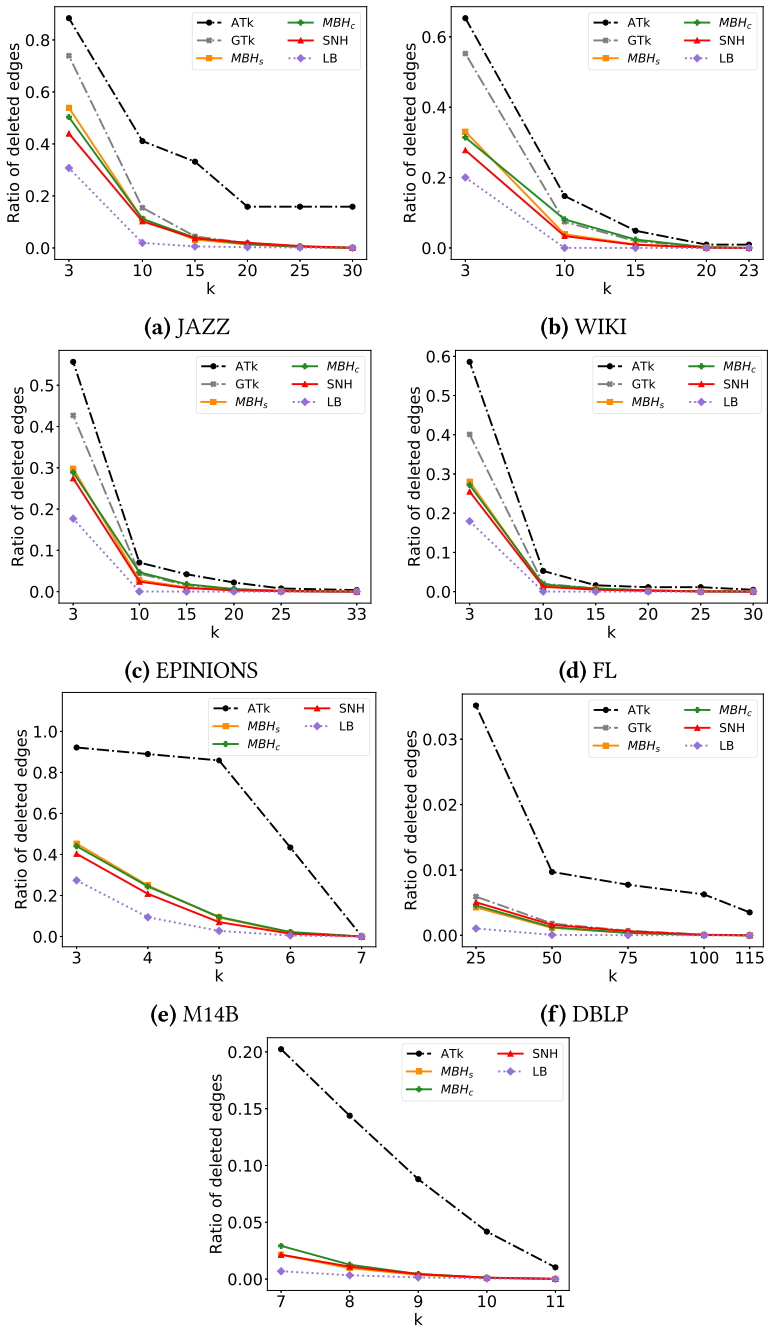


Fig. 11. Ratio of deleted edges, for varying k, on large graphs. SNH is close to our LB (lower bound). Thus, SNH is even closer to OPT (optimal solution), which is not computable on large graphs. GTk results are omitted from Figure 11(g), as it did not terminate in 24 hours.

Table 3. RE% in Terms of Global Clustering Coefficient for Varying k on Large Graphs. The best-performing method is in bold. As expected by its design, MBH_C is the clear winner. The results for $k = 3$ are omitted, as RE = 100% for all methods by definition; the results for GTk are also omitted, as GTk did not terminate in 24 hours

k	ATk	GTk	MBH_S	MBH_C	SNH
30	8.87	0.10	**0.08**	0.09	0.22
25	8.87	1.17	1.57	**0.8**	1.73
20	8.87	4.14	4.14	**2.43**	4.56
15	27	8.55	8.12	**6.19**	9.95
10	40.01	24.02	24.67	**23.37**	28.25
5	73.44	**59.74**	64.12	63.71	66.33

(a) JAZZ

k	ATk	GTk	MBH_S	MBH_C	SNH
23	7.6	0.044	0.034	**0.002**	0.003
20	7.6	1.47	1.37	**0.66**	0.83
15	28.65	10.93	11.83	**7.44**	10.7
10	55.22	29.38	32.142	**23.66**	32.4
5	81.11	65.98	70.22	**64.88**	70.68

(b) WIKI

k	ATk	GTk	MBH_S	MBH_C	SNH
33	6.517	0.040	0.006	**0.003**	0.008
25	9.07	3.08	2.61	**1.99**	3.20
20	23.78	7.90	7.19	**6.06**	8.31
15	35.90	17.28	17.00	**14.67**	17.90
10	49.74	34.07	34.33	**30.79**	35.09
5	81.85	68.57	68.96	**64.02**	67.95

(c) EPINIONS

k	ATk	GTk	MBH_S	MBH_C	SNH
30	6.922	0.025	0.011	**0.006**	0.02
25	16.37	1.81	1.41	**1.19**	1.87
20	16.37	6.04	5.29	**4.53**	6.26
15	20.36	11.90	11.38	**10.12**	11.83
10	36.94	21.62	21.35	**19.57**	21.78
5	71.37	54.80	54.80	**51.72**	54.45

(d) FL

k	ATk	MBH_S	MBH_C	SNH
7	0.08	0.03	**0.007**	0.017
6	8.07	5.26	**3**	4.63
5	48.86	22.82	**20.25**	21.4
4	74.72	51.98	**50.73**	51.61

(e) M14B

k	ATk	GTk	MBH_S	MBH_C	SNH
115	4.65	0.0024	0.0023	**0.0022**	0.0023
100	7.82	0.48	0.17	**0.16**	0.31
75	9.18	2.6	1.36	**1.35**	2.35
50	10.97	5.61	**3.84**	4.03	5.37
25	20.53	11.89	**10**	10.3	12.11

(f) DBLP

k	ATk	MBH_S	MBH_C	SNH
11	2.3	0.13	**0.08**	0.1
10	9.94	0.83	**0.52**	0.83
9	21.23	2.75	**1.81**	2.87
8	34.38	6.83	**4.74**	7.12
7	47	13.7	**10.23**	13.81

(g) AMAZON

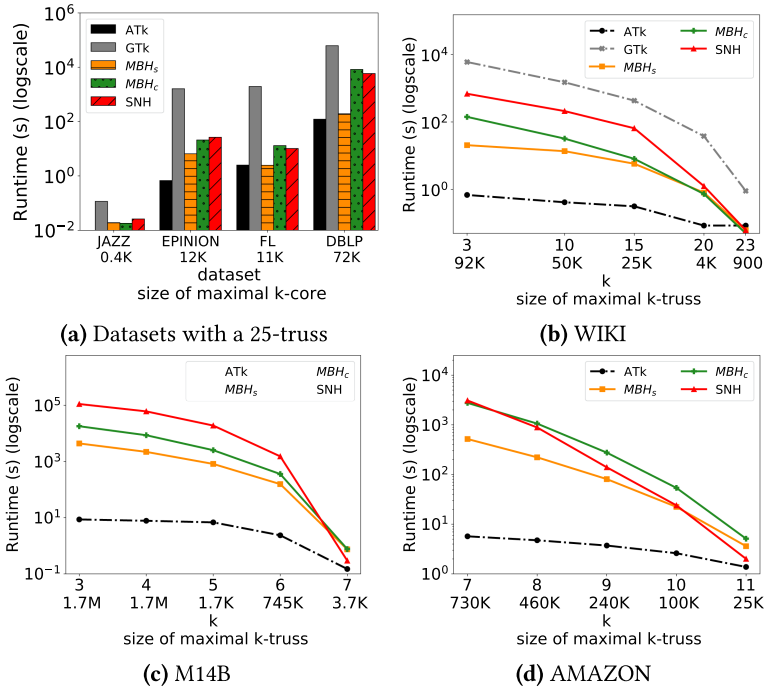
ACk (for All Coreness $\geq k$): It deletes all edges of coreness at least k without considering the impact of an edge deletion on the coreness of other edges. It takes $O(|E|)$ time, as it computes the core decomposition of G once. We compared to ACk to show how many edges are "saved" by our heuristics.

GCk (for Greedy Coreness $\geq k$): It iteratively deletes the edge with highest coreness, breaking ties arbitrarily, until there is no k-core in the graph. It takes $O(|E|r)$ time to delete r edges, as it computes the core decomposition after every iteration. Thus, GCk is much slower than ACk but identifies substantially fewer edges to delete. We compared against GCk to show the effectiveness of our heuristics and the efficiency impact of our data structures.

To measure effectiveness, we used the relative error $RE = \frac{|E'_{GCk}|-|E'|}{|E'_{GCk}|}$, where E'_{GCk} (respectively, E') is a solution of GCk (respectively, of one of our heuristics). We used this measure to highlight the difference from GCk. This is because there is a large difference in the ratio of deleted edges between ACk and GCk and across different k values. In addition, we report the number of deleted edges for ACk, GCk, and our heuristics in Appendix E.

We implemented all evaluated methods and executed them on the system described in Section 9.1. We used the algorithm of Reference [34] for computing the core decomposition.

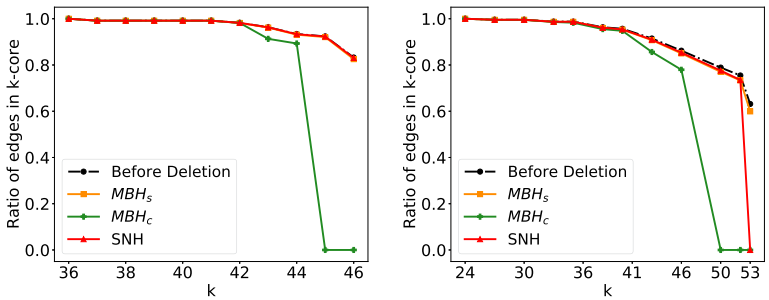
Why We Need k-Core-based Heuristics. We motivate the need for our k-core-based heuristics by showing that a feasible solution to MIN-k-TBS is not necessarily a feasible solution to MIN-k-CBS. Specifically, we applied our truss-based heuristics, MBH_S, MBH_C, and SNH on the 24-truss mentioned in Section 2 and measured the ratio of its edges that are in a 23-core after applying these heuristics. The ratio was 1. Thus, every edge is still in a 23-core. In fact, every remaining edge is still in a 36-core (see Figure 13(a) where the ratio for $k = 36$ is 1), since the 24-truss was also a 36-core. Moreover, at least 89% of the remaining edges are still in a 44-core. This shows that the developed truss-based heuristics were inappropriate for dealing with MIN-k-CBS and ineffective in



(a) Datasets with a 25-truss **(b)** WIKI

(c) M14B **(d)** AMAZON

Fig. 12. (a) Runtime when $k = 25$, for all datasets with a 25-truss. (b,c) Runtime for varying k. GTk results are omitted from Figures 12(c) and 12(d), as it did not terminate in 24 hours.



(a) FL **(b)** FL

Fig. 13. Ratio of edges with coreness k before and after applying MBH_S, MBH_C, and SNH on: (a) the 24-truss and (b) the 18-truss in Section 2.

reducing the coreness of edges. This is because there are many $(k'-1)$-cores that are not k-trusses, for some $k' \geq k$. Thus, removing all k-trusses did not remove all $(k-1)$-cores. Repeating the same experiment using the 18-truss mentioned in Section 2 yielded analogous results (see Figure 13(b)), since this 18-truss was also a 24-core.

Effectiveness. We show that our heuristics substantially outperform GCk for all tested k values (see Figure 14). ACk performed much worse than GCk and our heuristics (see Appendix E). Our heuristics deleted a much smaller percentage of edges compared to GCk, since they do not simply consider edge coreness as GCk does. The difference between GCk and our heuristics increases with

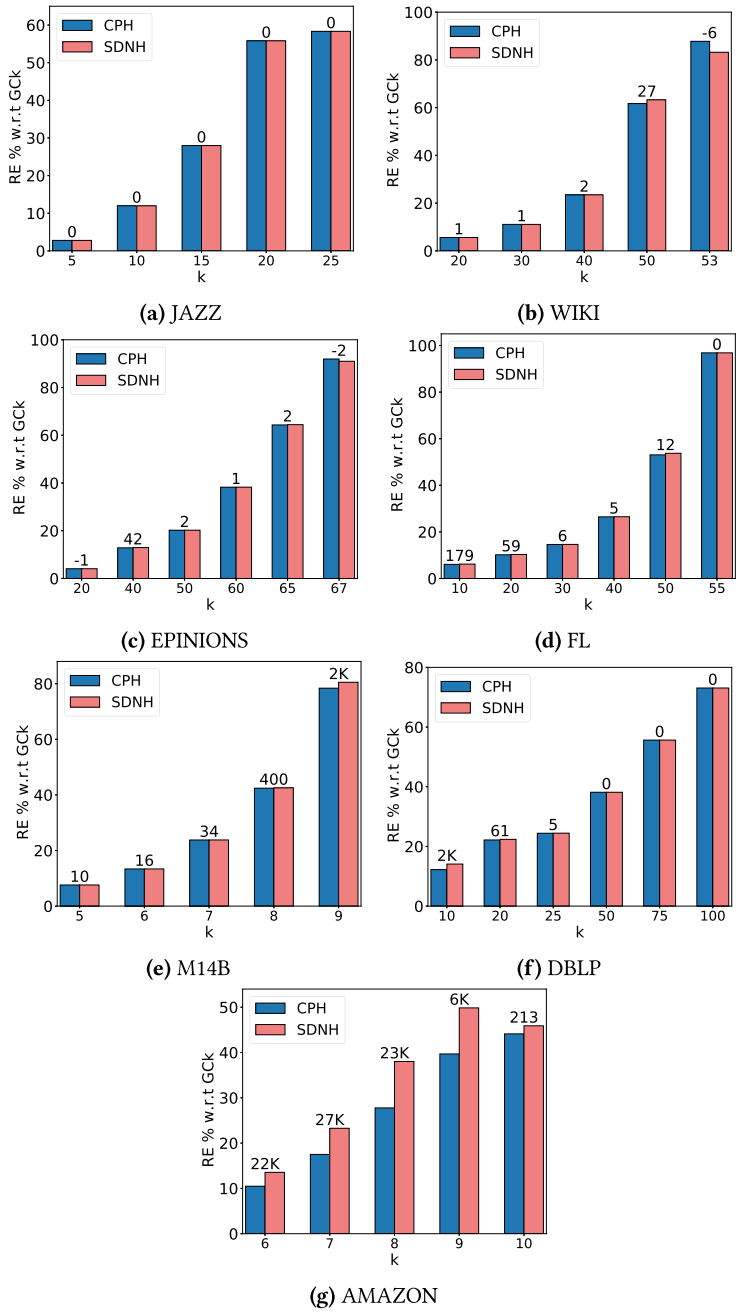
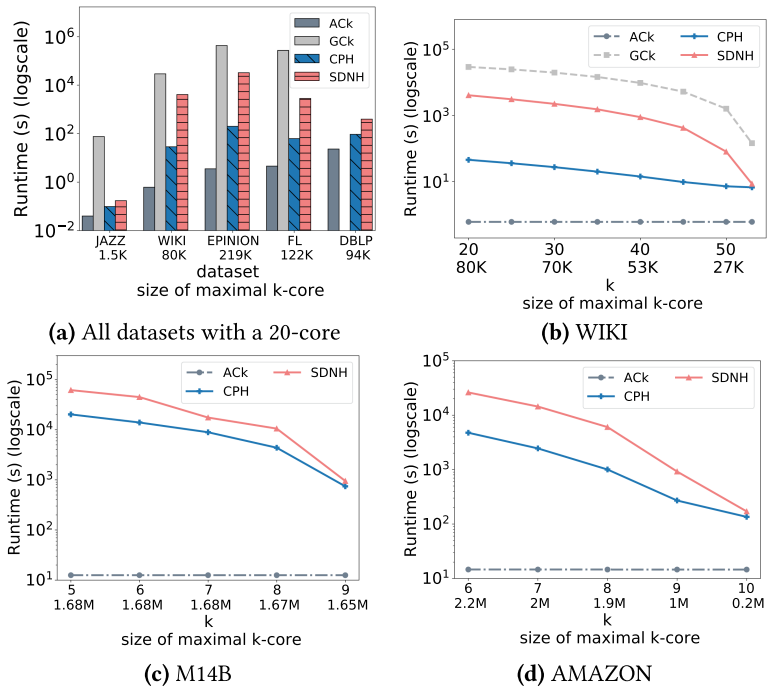


Fig. 14. $RE = \frac{|E'_{GCk}|-|E'|}{|E'_{GCk}|}\%$ improvement with respect to edge deletion of our heuristics compared to the GCk baseline for varying k (the higher the better) in all large datasets. The largest k corresponds to the coreness of G. The number on the top of each pair of bars indicates the difference $|E'_{\text{CPH}}| - |E'_{\text{SDNH}}|$ (i.e., how many more edges are deleted by CPH).



(a) All datasets with a 20-core

(b) WIKI

(c) M14B

(d) AMAZON

Fig. 15. (a) Runtime when $k = 20$ for all datasets with a 20-core. (b, c, d) Runtime for varying k. GCk results are omitted from Figure 15(c) and 15(d), as it did not terminate in 24 hours.

k, exceeding 80% in three of the datasets. This is because fewer edges need to be deleted when k is large, so the effect of careful edge deletion is larger. SDNH outperformed CPH in most cases, alike the max-truss-based SNH outperformed the maximal k-truss-based heuristics. This is due to the more fine-grained edge deletion strategy of SDNH, which considers the actual degree of nodes in the maximal k-core instead of just whether nodes have degree equal to a threshold in the max-core. The difference is in the order of several thousands of edges in AMAZON and is generally larger when k is small, since in this case more edges have to be deleted.

Efficiency. We show that our heuristics are at least one order of magnitude faster than GCk (see Figure 15), despite outperforming it significantly in terms of quality (Figure 14). For example, we were unable to run GCk on M14B or AMAZON within 24 hours. The reason is that our heuristics employ `Coreness Update` and `Core Decomposition Update`, instead of the expensive core decomposition procedure employed by GCk. CPH was faster than SNH, albeit generally less effective. As expected, ACk is the fastest method, because it does not need to recompute the coreness of edges after deletion; recall that ACk is by far the worst in terms of effectiveness, as mentioned above.

9.3 Case Study on a Co-authorship Network

We applied our heuristics to a co-authorship network, obtained from https://projects.csail.mit.edu/dnd/DBLP/. This dataset, which is referred to as COAUTH, contains an edge between every two authors who jointly wrote at least one paper that was retrieved from DBLP. The characteristics of COAUTH are shown in Table 4.

Our objective was to examine application A1 (see Section 1), i.e., to determine which edges are critical for maintaining the k-truss or k-core communities. We focused on the communities

Table 4. Characteristics of the COAUTH dataset

# Edges	# Nodes	$t(G)$	$c(G)$	Max degree	Avg degree
5,946,678	1,412,433	119	118	1,775	8

Table 5. Critical Edges (Output E') of MBH_S, MBH_C, and SNH when Applied to COAUTH. T_u denotes the max-truss of author u, $t(T_u)$ its trussness, and $|N_{T_u}(u)|$ the number of edges in T_u that are incident to u (i.e., the number of potential critical edges). The removal of critical edges breaks the max-trusses

Author u	$t(T_u)$	$\lvert N_{T_u}(u)\rvert$	Authors incident to u in the E' of MBH_S	Authors incident to u in the E' of MBH_C	Authors incident to u in the E' of SNH
Christos Faloutsos	34	33	William T. Young	William T. Young	Amanda Gentzel
Jeffrey D. Ullman	29	28	Michael Lesk	Michael J. Pazzani	Rakesh Agrawal
Sushil Jajodia	26	25	Richard T. Snodgrass	Nikos A. Lorentzos	Elisa Peressi
Andrew Y. Ng	12	12	Rajat Monga	Rajat Monga	Paul A. Tucker
Li Fei-Fei	12	11	Zhiheng Huang	Zhiheng Huang	Michael S. Bernstein
Dacheng Tao	11	11	Wanlin Zhu Dong Xu	Wanlin Zhu Ching Yu Cheng	Guangqiang Geng Ching Yu Cheng

of well-known authors. Specifically, we selected 6 authors from the top-200 computer science authors according to https://research.com/scientists-rankings/computer-science. We applied our heuristics for MIN-k-TCBS and those for MIN-k-CCBS to the COAUTH dataset, and the set U was comprised of the node corresponding to one of the 6 selected authors.

We first examined which are the critical edges for maintaining the max-truss of each of the 6 authors. As can be seen in Table 5, the node of each of these authors is contained in a very dense max-truss and has several edges in the max-truss. For example, the trussness of the max-truss of Christos Faloutsos is 34 and its node has 33 edges in the max-truss. However, our heuristics identified only one or two edges as critical for any author. This implies that maintaining the k-trusses of a graph is an important problem, since few edges can break them. This finding is in line with prior work [55]. Furthermore, each of our heuristics selected different edges than the others in many cases. This shows that the different goals of our heuristics are reflected in the solutions they construct, even though these solutions are comprised of a small number of edges.

Then, we examined which are the critical edges for maintaining the max-core of the 6 selected authors that were mentioned above. The results are in Table 6. As can be seen, the node of each of these authors is contained in a very dense max-core and has several edges in the max-core. Yet, our heuristics identified a small number of edges as critical for any author. Again, this shows the importance of maintaining k-cores [54], as they may be broken by the deletion of few edges. Furthermore, our heuristics selected different edges due to their different design goals.

By comparing Tables 5 and 6, we can see that the edges that are selected by our heuristics to break k-trusses are different from those that were selected to break k-cores. This justifies the need for designing different methods for each type of community. In addition, it can be observed that the critical edges tend to link the selected authors with authors who have much fewer co-authors (i.e., they are in fewer triangles in the max-truss or have much lower degree in the max-core). This is because the algorithms intuitively select nodes that can be removed from the max-truss or max-core after the deletion of few edges (e.g., when $u \in U$, MBH_S and MBH_C select an edge (u, u') where u' has an incident edge e' with the smallest possible support in the max-truss).

To further demonstrate the results of our heuristics, we visualized a subgraph of the k-truss of one of the selected authors, namely, Jeffrey D. Ullman, for $k = 29$ (this truss is also the max-truss) and for $k = 25$, in Figures 16(a) and 16(b), respectively. The subgraphs we visualized are composed of the ego-network of Jeffrey D. Ullman (i.e., the graph comprised of the node of Jeffrey D. Ullman,

Table 6. Critical Edges (Output E') of CPH and SDNH when Applied to COAUTH. C_u denotes the max-core of author u, $c(C_u)$ its coreness, and $|N_{C_u}(u)|$ the number of edges in C_u that are incident to u (i.e., the number of potential critical edges). The removal of critical edges breaks the max-cores

Author u	$c(C_u)$	$\lvert N_{C_u}(u)\rvert$	Authors incident to u in the E' of CPH	Authors incident to u in the E' of SDNH
Christos Faloutsos	33	44	Thomas G. Dietterich	Henry G. Goldberg
Jeffrey D. Ullman	28	29	David Maier Hans-Jörg Schek	Dieter Gawlick Michael J. Pazzani
Sushil Jajodia	25	43	Abdullah Uz Tansel	Tyrone Grandison
Andrew Y. Ng	22	23	Ian J. Goodfellow	Andrew McCallum
Li Fei-Fei	14	14	Percy Liang	Percy Liang
Dacheng Tao	28	65	Jimeng Sun Richang Hong	Gang Hua Richang Hong Linjun Yang Dong Xu Tat-Seng Chua Xuelong Li

the neighbors of this node, the edges from this node to its neighbors, and the edges between these neighbors). We also show the selected edges by each of our heuristics. Note that the heuristics selected few edges incident to the node representing Jeffrey D. Ullman to break the 29-truss or 25-truss and that these edges are generally different from one heuristic to another. Next, we performed a similar experiment but for the k-core heuristics. Specifically, we visualized the ego-network of Jeffrey D. Ullman contained in his 28-core in Figure 17(a) (this core is also the max-core) and that contained in his 25-core in Figure 17(b). Again, our heuristics selected few edges that were often different between heuristics.

These experiments illustrate the differences between the two types of communities we focus on: k-trusses and k-cores. The latter are less dense and larger in size (e.g., the 28-core for Jeffrey D. Ullman has about 59% lower density compared to the 29-truss for the same person, and it is about 2 times larger). For example, many of the co-authors of Jeffrey D. Ullman are part of his k-core, although they have not jointly published a paper (see, for example, the node of Chen Li in Figure 17(a) who has published a paper with only three co-authors of Jeffrey D. Ullman). Thus, the notion of k-truss captures how closely the co-authors of an author in the k-truss work together better than the notion of k-core. In general, since k-cores are less dense and larger than k-trusses, they require a larger number of critical edges to be maintained. However, the k-cores capture less subtle relationships.

9.4 Modularity Preservation

We examined how well our heuristics can preserve *modularity* [36], a well-known measure of graph clustering quality, computed for a given graph on its clustering (node partition). It can be expressed as the fraction of the edges of the graph that fall within the given clusters minus the expected fraction if the edges were distributed at random (see Reference [36] for a formal definition). Thus, a large modularity value corresponds to a "high quality" clustering, and the largest possible value is 1, whereas a random clustering is expected to have modularity close to 0.

We considered all large datasets that were used in Sections 9.1 and 9.2 (i.e., the last seven datasets in Table 9.1) except WIKI and EPINIONS, which do not have a meaningful clustering structure (their modularity was very close to 0).

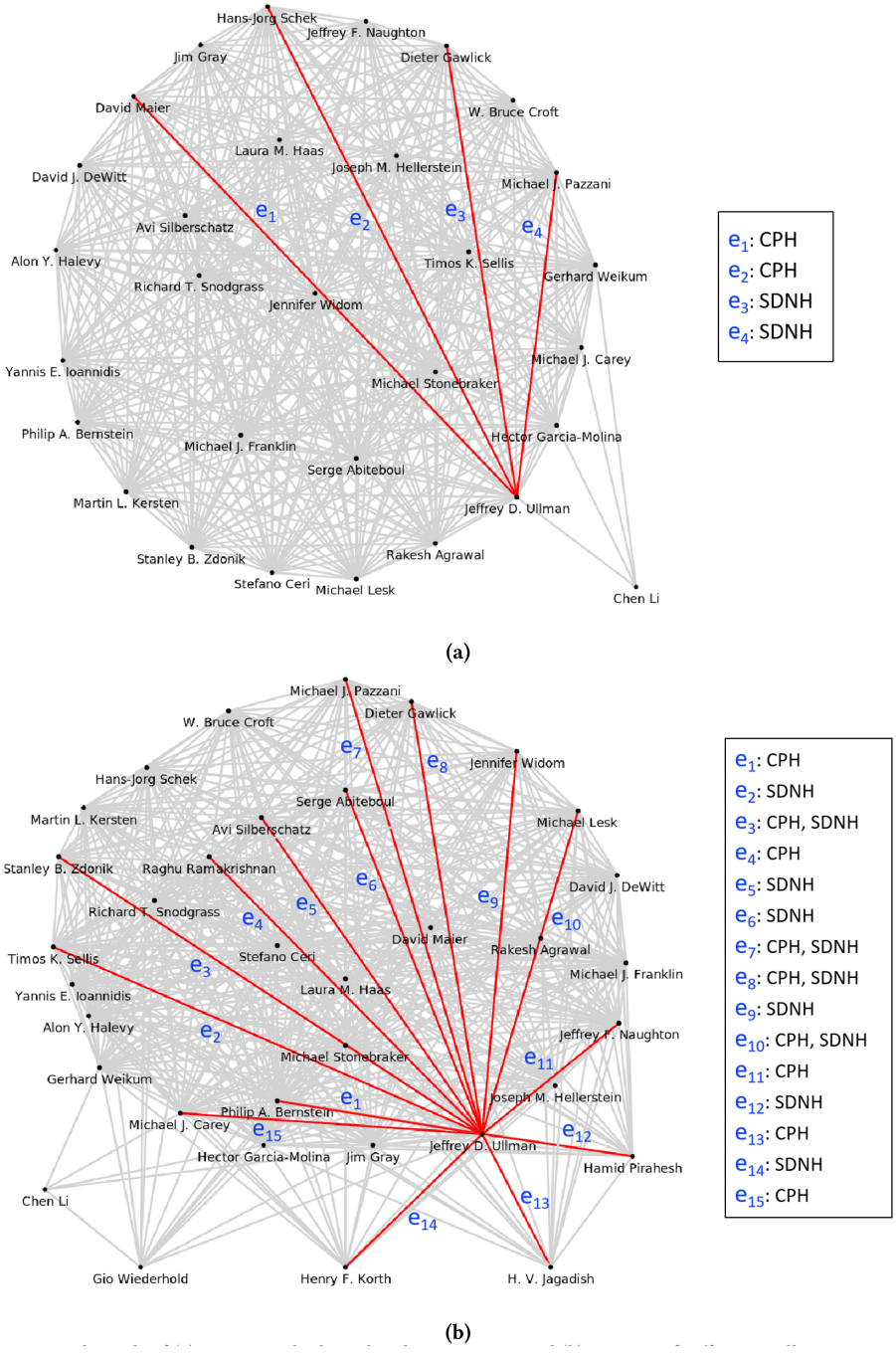


Fig. 16. Subgraph of (a) 29-truss, which is also the max-truss, and (b) 25-truss of Jeffrey D. Ullman containing his co-authors, the edges between Jeffrey D. Ullman and his coauthors, and the edges between these co-authors. The trusses are too large for visualization. The red edges on the graph are selected by our heuristic(s).

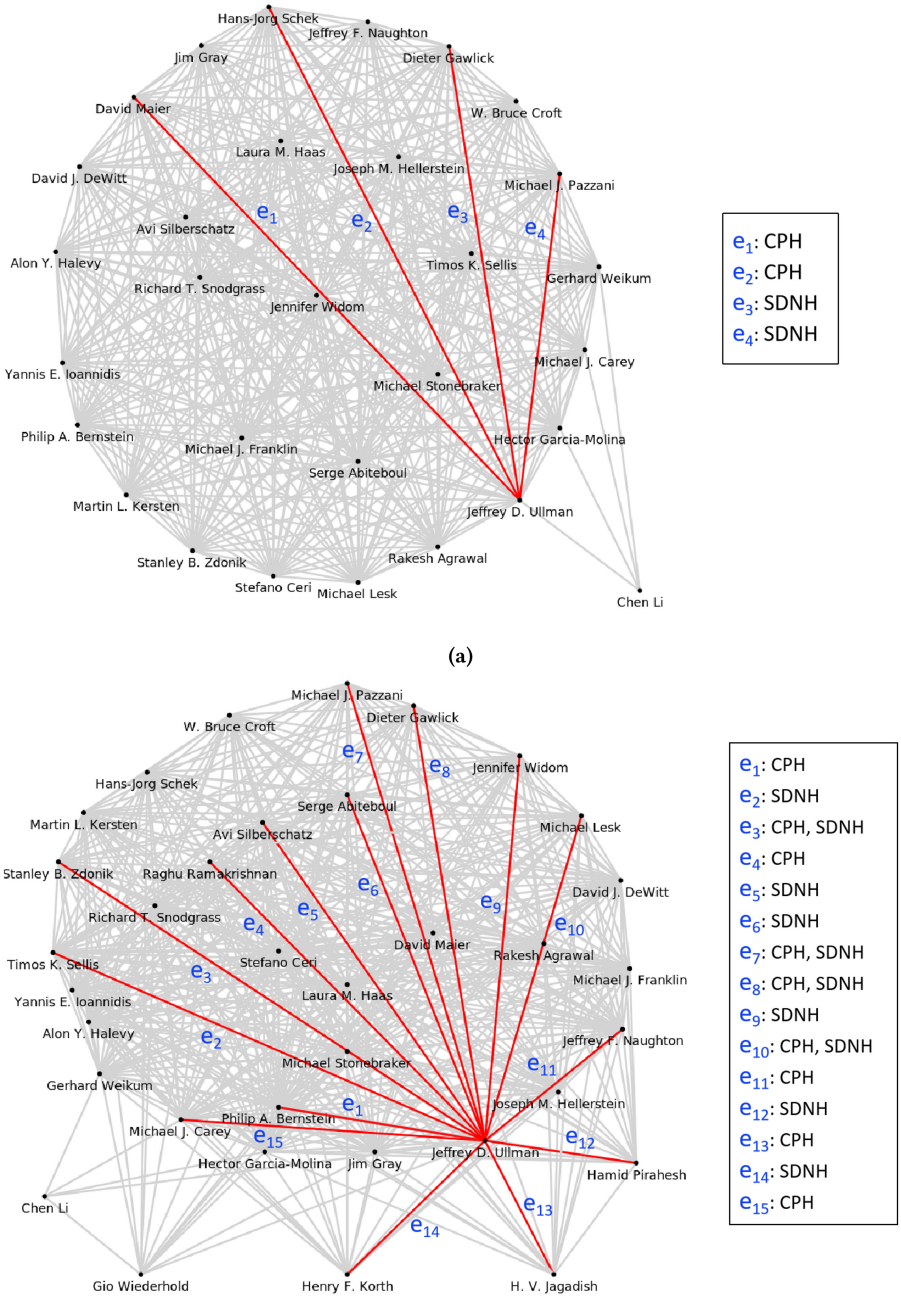


Fig. 17. Subgraph of (a) 28-core, which is also the max-core, and (b) 25-core of Jeffrey D. Ullman containing his co-authors, the edges between Jeffrey D. Ullman and his coauthors, and the edges between these co-authors. The cores are too large for visualization. The red edges on the graph are selected by our heuristic(s).

We first applied the well-established Asynchronous Label Propagation (ALP) algorithm [39] to obtain a clustering C of the graph $G(V, E)$ and then measured the modularity of C.[8] Next, we applied one of our heuristics for the MIN-k-TBS problem or for the MIN-k-CBS problem to $G(V, E)$ and produced the graph $G'(V, E \setminus E')$, where E' is the output of the applied heuristic. Then, we measured the modularity of G' based on C. We tested all our heuristics and all k values in Sections 9.1 and 9.2.

In this scenario, one might expect that removing edges within a community could reduce the modularity value, as such edges bring a positive contribution to its computation. However, our heuristics made mostly negligible changes: For the smallest values of k in Figure 11, the value of modularity computed on G' had an average relative error of 1.9% compared to that of G. Moreover, the relative error quickly became negligible as k increased. This is expected, as increasing k dramatically reduces the number of edges $|E'|$ that are selected by our truss-based or core-based heuristics (see, e.g., Figure 11). Similar observations were made for our k-core heuristics: For the smallest values of k in Figure 14, the relative error was 4.2%, on average. These results suggest that the proposed heuristics are able to effectively break the unwanted combinatorial structures, namely, k-trusses or k-cores, without significantly impacting the dynamics of networks.

10 CONCLUSION

In this article, we introduced the general community breaking problem and formalized two combinatorial optimization problems based on it: MIN-k-TBS, which employs the notion of k-truss; and MIN-k-TCBS, which employs the notion of k-core. We also formalized variants of MIN-k-TBS and of MIN-k-TCBS in which solutions are constrained to be comprised of edges incident to a given set of nodes.

We made both theoretical and practical contributions: On the theoretical side, we showed that the problems introduced are NP-hard and hard to approximate. In addition, for MIN-k-TBS and its variant, we provided exact exponential-time algorithms and a non-trivial lower bound on the size of the optimal solution. On the practical side, we developed a number of heuristics based on theoretical insights, which were sped up by efficient data structures we designed. Our experimental results confirm our complexity analyses and show that our heuristics outperform baselines and offer excellent tradeoffs between effectiveness and efficiency.

Two final remarks are in order: A lower bound for MIN-k-CBS is possible using similar ideas to those in Section 7, since a k-clique is also a k-core; this turned out to be unsatisfactory, as k-cores are often much larger than the k-cliques contained in them. Similarly, due to the large size of k-cores, an exact algorithm for MIN-k-CBS based on the binary partition method presented in Section 5 turns out to be computationally prohibitive even for small graphs. Designing new methods to obtain effective lower and upper bounds for MIN-k-CBS thus constitutes an interesting direction for future work.

APPENDIX

A TRUSS DECOMPOSITION UPDATE: DETAILS

Consider an edge f that is popped from L and is contained in fewer than $t_{cur}(f) - 2$ triangles of trussness $t_{cur}(f)$. We first provide details on how operation O_1 is performed for f:

While f is in fewer than $t_{cur}(f) - 2$ triangles of trussness $t_{cur}(f)$, we:

(1) Update the trussness of f in T by:

[8] We used the ALP implementation and the modularity computation function from Reference [1] with their default parameters.

— Adding the number of triangles of trussness $t_{cur}(f)$ to those of trussness $t_{cur}(f)-1$.
— Setting the number of triangles of trussness $t_{cur}(f)$ to 0.

(2) Reduce the trussness of f in T_{cur} by 1.

We now provide details on how operation O_2 is performed for f:
For each triangle $\{f,g,h\} \in \text{TRI}(e)$ whose trussness changes, we:

(1) Compute the original trussness of $\{f,g,h\}$. Let it be tt_1.
(2) Compute the new trussness of $\{f,g,h\}$. Let it be tt_2.
(3) If $tt_2 < tt_1$, then we update the entries for g and h in T to reflect the new number of triangles of trussness tt_1 and tt_2 in which g and h are contained.
(4) Push the edges g and h to L.

B CORE DECOMPOSITION UPDATE: DETAILS

We provide details on how operation O_2 is performed. Recall that, in this operation, we update of the coreness of $C[v]$. This is performed as follows:

(1) Update the entry in $C[v]$. Specifically, we distinguish two cases:
— $C[v][\rho] > \rho$: All edges except (u,v) remain in ρ-core. We subtract 1 from $C[v][\rho]$, and, if the updated coreness of (u,v) is not 0, then we also increase $C[v][\rho-1]$ by 1.
— $C[v][\rho] \leq \rho$: We add $C[v][\rho]$ to $C[v][\rho-1]$ and set $C[v][\rho]=0$, because the coreness of all edges in $M_v[\rho]$ will become $\rho-1$. If the updated coreness of (u,v) is 0, then we also subtract $C[v][\rho-1]$ by 1.
(2) If $C[v][\rho]=0$, then push all un-visited edges in $M_v[\rho]$ into L and mark these edges as visited, by adding them to hash table K, then update $R[v]=M_v$.

C IMPROVEMENTS TO MBH$_S$ AND MBH$_C$

We plug in the `Truss Update` and `Triangles Update` data structures, which can be constructed in $O(|E|^{3/2})$ time, into MBH$_S$ and MBH$_C$. Since `Truss Update` provides the trussness of each edge in $O(1)$ time, Lines 3 and 4 take $O(|E|)$ time. Since `Triangles Update` provides $O(1)$-time access to the list of triangles containing an edge e', Lines 5 and 6 take $O(\sup_T(e'))$ time. The update of G' is handled by the maintenance of `Truss Update` and `Triangles Update`, which amortizes to $O(t(G)\mathcal{T}_G + \mathcal{T}_G)$ time across *all* edge deletions. This gives the *improved* time bound of $O(|E|^{3/2} + |E|r + t(G)\mathcal{T}_G)$. For MBH$_C$, the only difference is that we consider all triangles in G' in Line 6. Since we delete a triangle after considering it, the cost amortizes to $O(\mathcal{T}_G)$. This gives the same *improved* time bound of $O(|E|^{3/2} + |E|r + t(G)\mathcal{T}_G)$.

D ADDITIONAL EXPERIMENTAL RESULTS: k-TRUSSES

Table 7 shows that MBH$_C$ outperforms the other heuristics in terms of RE.

Table 7. RE% in Terms of Global Clustering Coefficient for Varying k on Small Graphs. The best-performing method is in bold. MBH$_C$ is the clear winner. Results for $k=3$ are omitted, as $RE=100\%$ for all methods by definition

k	MBH$_S$	MBH$_C$	SNH
5	12.061	**6.811**	8.418
4	**30.541**	35.327	33.276

(a) TRIBES

k	MBH$_S$	MBH$_C$	SNH
5	6.878	6.878	**6.407**
4	25.216	**21.597**	36.861

(b) KARATE

k	MBH$_S$	MBH$_C$	SNH
5	4.364	**4.159**	5.767
4	28.759	**25.167**	28.669

(c) DOLPHINS

k	MBH$_S$	MBH$_C$	SNH
9	0.716	**0.138**	0.138
8	1.873	**1.156**	2.626
7	3.945	**3.148**	4.65

(d) NETSCIENCE

E ADDITIONAL EXPERIMENTAL RESULTS: k-CORES

Tables 8 and 9 show the number of deleted edges for ACk, GCk, and our heuristics.

Table 8. Number of deleted edges, for varying k, on the first four of the large datasets in Table 1 of the article

k	ACk	GCk	CPH	SDNH
5	2,687	2,046	1,989	1,989
10	2,610	1,319	1,161	1,161
15	2,377	629	453	453
20	1,465	172	76	77
25	435	36	15	15

(a) JAZZ

k	ACk	GCk	CPH	SDNH
20	80,460	43,216	40,793	40,792
30	69,653	24,925	22,160	22,161
40	53,340	11,061	8,466	8,464
50	26,713	1,732	663	636
53	14,117	131	16	22

(b) WIKI

k	ACk	GCk	CPH	SDNH
20	219,184	120,673	115,724	115,728
40	142,214	44,057	38,394	38,352
50	103,363	21,359	17,040	17,040
60	63,891	6,545	4,042	4,042
65	45,128	1,724	615	613
67	27,461	211	17	19

(c) EPINION

k	ACk	GCk	CPH	SDNH
10	287,185	139,945	131,474	131,295
20	122,401	48,657	43,701	43,642
30	70,354	23,769	20,299	20,293
40	46,281	9,503	6,988	6,983
50	22,215	1,809	849	840
55	8,978	95	3	6

(d) FL

Table 9. Number of deleted edges, for varying k, on the last three of the large datasets in Table 1 of the article

k	ACk	GCk	CPH	SDNH
5	1,678,858	887,417	820,008	819,998
6	1,678,410	698,702	605,328	605,312
7	1,677,257	512,901	390,914	390,880
8	1,672,520	314,029	180,835	180,435
9	1,653,731	97,210	21,010	18,943

(a) M14B

k	ACk	GCk	CPH	SDNH
10	405,960	115,858	101,695	99,535
20	93,741	29,455	22,927	22,871
25	70,449	20,574	15,551	15,546
50	20,476	6,530	4,039	4,039
75	11,707	2,700	1,198	1,198
100	11,707	457	123	123

(b) DBLP

k	ACk	GCk	CPH	SDNH
5	2,255,256	1,013,410	953,327	940,449
6	2,172,725	724,442	648,512	630,162
7	2,061,253	459,277	378,898	358,643
8	1,847,868	223,823	161,670	152,530
9	990,145	55,944	33,745	35,781
10	219,390	4,885	2,730	2,698

(c) AMAZON

REFERENCES

[1] A. A. Hagberg, D. A. Schult, and P. J. Swart. 2023. Network-X 3.1. Retrieved from https://networkx.org/documentation/stable/release/release_3.1.html

[2] N. Alon, A. Shapira, and B. Sudakov. 2005. Additive approximation for edge-deletion problems. In *FOCS*. 419–428. DOI: https://doi.org/10.1109/SFCS.2005.11

[3] N. Alon, R. Yuster, and U. Zwick. 1997. Finding and counting given length cycles. *Algorithmica* 17, 3 (1997), 209–223.
[4] J. Ignacio Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani. 2008. K-core decomposition of Internet graphs: hierarchies, self-similarity and measurement biases. *Netw. Heterogen. Media* 3, 2 (2008), 371–393.
[5] N. R. Aravind, R. B. Sandeep, and N. Sivadasan. 2017. Dichotomy results on the hardness of H-free edge modification problems. *SIAM J. Discr. Math.* 31, 1 (2017), 542–561. DOI: https://doi.org/10.1137/16M1055797
[6] V. Batagelj and M. Zaversnik. 2003. An O (m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049* (2003).
[7] G. Beigi and H. Liu. 2020. A survey on privacy in social media: Identification, mitigation, and applications. *ACM/IMS Trans. Data Sci.* 1, 1, Article 7 (2020), 38 pages.
[8] G. Bernardini, H. Chen, A. Conte, R. Grossi, G. Loukides, N. Pisanti, S. P. Pissis, G. Rosone, and M. Sweering. 2020. Combinatorial algorithms for string sanitization. *ACM Trans. Knowl. Discov. Data* 15, 1, Article 8 (2020), 34 pages.
[9] G. Bernardini, A. Conte, G. Gourdel, R. Grossi, G. Loukides, N. Pisanti, S. Pissis, G. Punzi, L. Stougie, and M. Sweering. 2023. Hide and mine in strings: Hardness, algorithms, and experiments. *IEEE Trans. Knowl. Data Eng.* 35, 6 (2023).
[10] K. Bhawalkar, J. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma. 2012. Preventing unraveling in social networks: The anchored k-core problem. In *ICALP*. 440–451.
[11] B. Bollobás. 2004. *Extremal Graph Theory*. Courier Corporation.
[12] F. Bonchi, A. Khan, and L. Severini. 2019. Distance-generalized core decomposition. In *SIGMOD*. 1006–1023.
[13] H. Chen, A. Conte, R. Grossi, G. Loukides, S. P. Pissis, and M. Sweering. 2021. On breaking truss-based communities. In *KDD*. 117–126.
[14] L. Chen, C. Liu, R. Zhou, J. Li, X. Yang, and B. Wang. 2018. Maximum co-located community search in large scale social networks. *Proc. VLDB Endow.* 11, 10 (2018), 1233–1246.
[15] J. Cohen. 2008. Trusses: Cohesive subgraphs for social network analysis. *Nat. Secur. Agency Tech. Rep.* 16 (2008), 3–29.
[16] A. Conte, D. De Sensi, R. Grossi, A. Marino, and L. Versari. 2020. Truly scalable k-truss and max-truss algorithms for community detection in graphs. *IEEE Access* 8 (2020), 139096–139109.
[17] A. Conte, D. Firmani, C. Mordente, M. Patrignani, and R. Torlone. 2017. Fast enumeration of large k-plexes. In *KDD*. 115–124.
[18] A. Conte, R. Grossi, and A. Marino. 2020. Large-scale clique cover of real-world networks. *Inf. Computat.* 270 (2020), 104464.
[19] I. Dinur, V. Guruswami, S. Khot, and O. Regev. 2003. A new multilayered PCP and the hardness of hypergraph vertex cover. In *STOC*. 595–601.
[20] S. Ebadian and X. Huang. 2019. Fast algorithm for k-truss discovery on public-private graphs. In *IJCAI*. 2258–2264.
[21] Y. Fang, X. Huang, L. Qin, Y. Zhang, W. Zhang, R. Cheng, and X. Lin. 2020. A survey of community search over big graphs. *VLDB J.* 29, 1 (2020), 353–392.
[22] D. C. Fisher. 1989. Lower bounds on the number of triangles in a graph. *J. Graph Theor.* 13, 4 (1989), 505–512.
[23] S. Fortunato. 2010. Community detection in graphs. *Phys. Rep.* 486, 3 (2010), 75–174.
[24] E. Galimberti, M. Ciaperoni, A. Barrat, F. Bonchi, C. Cattuto, and F. Gullo. 2020. Span-core decomposition for temporal networks: Algorithms and applications. *ACM Trans. Knowl. Discov. Data* 15, 1, Article 2 (Dec. 2020), 44 pages. DOI: https://doi.org/10.1145/3418226
[25] D. Gunopulos, H. Mannila, R. Khardon, and H. Toivonen. 1997. Data mining, hypergraph transversals, and machine learning. In *PODS*. 209–216.
[26] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. 2014. Querying k-truss community in large and dynamic graphs. In *SIGMOD*. 1311–1322.
[27] X. Huang and L. V. S. Lakshmanan. 2017. Attribute-driven community search. *Proc. VLDB Endow.* 10, 9 (2017), 949–960.
[28] R. M. Karp. 1972. Reducibility among combinatorial problems. In *Complexity of Computer Computations*. Springer, 85–103.
[29] S. Khot. 2002. On the power of unique 2-prover 1-round games. In *STOC*. 767–775.
[30] S. Khot and O. Regev. 2008. Vertex cover might be hard to approximate to within 2-ϵ. *J. Comput. Syst. Sci.* 74, 3 (2008), 335–349.
[31] Y.-X. Kong, G.-Y. Shi, R.-J. Wu, and Y.-C. Zhang. 2019. k-core: Theories and applications. *Phys. Rep.* 832 (2019), 1–32. DOI: https://doi.org/10.1016/j.physrep.2019.10.004
[32] R. Laishram, A. E. Sar, T. Eliassi-Rad, A. Pinar, and S. Soundarajan. 2020. Residual core maximization: An efficient algorithm for maximizing the size of the k-core. In *SDM*. 325–333.
[33] R. Laishram, A. E. Sariyüce, T. Eliassi-Rad, A. Pinar, and S. Soundarajan. 2018. Measuring and improving the core resilience of networks. In *WWW*. 609–618.
[34] D. W. Matula and L. L. Beck. 1983. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM* 30, 3 (July 1983), 417–427. DOI: https://doi.org/10.1145/2402.322385

[35] K. Murakami and T. Uno. 2014. Efficient algorithms for dualizing large-scale hypergraphs. *Discr. Appl. Math.* 170 (2014), 83 – 94.
[36] M. E. J. Newman. 2006. Modularity and community structure in networks. *Proc. Nat'l Acad. Sci.* 103, 23 (2006), 8577–8582.
[37] M. E. J. Newman, S. H. Strogatz, and D. J. Watts. 2001. Random graphs with arbitrary degree distributions and their applications. *Phys. Rev. E* 64, 2 (2001), 026118.
[38] V. S. Nikiforov and N. G. Khadzhiivanov. 1981. Solution of the problem of P. Erdos on the number of triangles in graphs with n vertices and [n2/4]+ l edges. *CR Acad. Bulgare Sci.* 34, 969-970 (1981), 2.
[39] U. N. Raghavan, R. Albert, and S. Kumara. 2007. Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E* 76, 3 (2007), 036106.
[40] F. Rousseau, J. Casas-Roma, and M. Vazirgiannis. 2018. Community-preserving anonymization of graphs. *Knowl. Inf. Syst.* 54, 2 (2018), 315–343.
[41] Y. Saygin, V. S. Verykios, and C. Clifton. 2001. Using unknowns to prevent discovery of association rules. *SIGMOD Rec.* 30, 4 (2001), 45–54.
[42] S. B. Seidman. 1983. Network structure and minimum degree. *Soc. Netw.* 5, 3 (1983), 269–287.
[43] N. Tatti. 2019. Density-friendly graph decomposition. *ACM Trans. Knowl. Discov. Data* (Sep. 2019) Article 54, 29 pages.
[44] C. Wang and J. Zhu. 2019. Forbidden nodes aware community search. *AAAI* 33, 01 (2019), 758–765.
[45] Z. Wang, C. Wang, W. Wang, X. Gu, B. Li, and D. Meng. 2020. Adaptive relation discovery from focusing seeds on large networks. In *ICDE*. 217–228.
[46] D. Wen, L. Qin, Y. Zhang, X. Lin, and J. X. Yu. 2019. I/O efficient core graph decomposition: Application to degeneracy ordering. *IEEE Trans. Knowl. Data Eng.* 31, 1 (2019), 75–90. DOI: https://doi.org/10.1109/TKDE.2018.2833070
[47] D. Yang, B. Qu, J. Yang, and P. Cudre-Mauroux. 2019. Revisiting user mobility and social relationships in LBSNs: A hypergraph embedding approach. In *WWW*. 2147–2157.
[48] M. Yannakakis. 1981. Edge-deletion problems. *SIAM J. Comput.* 10, 2 (1981), 297–309. DOI: https://doi.org/10.1137/0210021 arXiv:https://doi.org/10.1137/0210021
[49] F. Zhang, C. Li, Y. Zhang, L. Qin, and W. Zhang. 2020. Finding critical users in social communities: The collapsed core and truss problems. *IEEE Trans. Knowl. Data Eng.* 32, 1 (2020), 78–91.
[50] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. 2017. Finding critical users for social network engagement: The collapsed k-core problem. In *AAAI*, Satinder P. Singh and Shaul Markovitch (Eds.). 245–251.
[51] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. 2017. When engagement meets similarity: Efficient (k,r)-core computation on social networks. *Proc. VLDB Endow.* 10, 10 (June 2017), 998–1009.
[52] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. 2018. Efficiently reinforcing social networks over user engagement and tie strength. In *ICDE*. 557–568.
[53] Z. Zhou, F. Zhang, X. Lin, W. Zhang, and C. Chen. 2019. K-core maximization: An edge addition approach. In *IJCAI*. 4867–4873.
[54] W. Zhu, C. Chen, X. Wang, and X. Lin. 2018. K-core minimization: An edge manipulation approach. In *CIKM*. 1667–1670.
[55] W. Zhu, M. Zhang, C. Chen, X. Wang, F. Zhang, and X. Lin. 2019. Pivotal relationship identification: The k-truss minimization problem. In *IJCAI*. 4874–4880.

Received 16 December 2022; revised 30 October 2023; accepted 3 January 2024