Skeletal cores and graph resilience

No Author Given

No Institute Given

Abstract. The structure of dense subgraphs within a network can provide important insights into the properties and resilience of the network; and, as such, such subgraphs have received much attention in the machine learning and network science literature. One of the most prominent types of dense subgraphs is the k-core: the maximal set of nodes such that each node in the k-core has at least k neighbors within the core. Recently, the notion of the *skeletal k-core*– a minimal subgraph that preserves the core structure of the graph– has attracted attention. However, the literature to date has contained only a biased greedy heuristic for sampling skeletal cores, which resulted in a skewed analysis of the network. In this work, we introduce a novel MCMC algorithm for sampling skeletal cores uniformly at random, as well a novel algorithm for estimation of the size of the space of skeletal k-cores, which, as we show, is important for the understanding the core resilience of the network. With these algorithms, we demonstrate the relationship between *resilience* of the network and the core structure of the graph and suggest fast heuristics for evaluating graph structure from a skeletal cores perspective. We show that the normalized number of skeletal cores in the graph correlates with the resilience of k-core towards edge deletion attacks.

Keywords: networks · robustness · k-cores.

1 Introduction

Within the machine learning and network science literature, the study of dense subgraphs has received a great deal of attention. Examples of such structures include cliques [18], k-clubs [21] and k-trusses [7]. Of particular interest is the k-core, which plays a role in applications such as community detection [23, 19], influence maximization [11], visualization [12, 3], anomaly detection [28] and understanding network topology [29, 3]. A network's k-core is defined as the maximal subgraph of the network such that every node in the subgraph has at least k neighbours also in the subgraph [26]. The core number of a node is the highest value of k for which that node belongs to a k-core.

k-cores have a history of being studied in the context of network resilience [17, 20]. However, an in-depth understanding of the impact that the *structure* of k-cores has on a network's resilience has been less thoroughly examined in the literature. For a better understanding of the structural properties of k-cores, the concept of the *skeletal core* was proposed [16]. The skeletal core of a graph is defined as a minimal subgraph that preserves the core number of each node. A

graph may have multiple skeletal cores. Skeletal cores can be seen as a "backbone" of the k-core structure of the graph, and so play an important role in the structure of the network. While useful for applications such as speeding up community detection, they are especially in the context of analyzing the network's *resilience* or *robustness* [16]. To this end, understanding the space of skeletal cores can provide valuable insight into the graph structure. In order to understand when a network's core structure will change, it is necessary to also understand when it remains the same.

However, because skeletal cores are a new concept, there are a number of important open problems surrounding them. For instance, the literature contains only a single greedy heuristic for identifying a single skeletal core, and it is not known how to sample skeletal cores of a network uniformly at random, which fundamentally limits the understanding of how skeletal cores are distributed and how they affect the core structure. Relatedly, it is also not known how to estimate the number of skeletal cores to which an edge belongs, making it difficult to gauge the importance of an edge with respect to the overall core structure of the network. Without solving these problems, one cannot properly understand the effect of a network's skeletal core structure on the resilience of the network.

In our work, we first introduce a novel MCMC algorithm for sampling skeletal k-cores uniformly at random and a corresponding algorithm for estimating the number of skeletal cores in the graph. We then use these algorithms to study the relationship between the space of skeletal cores and the robustness of complex networks. We suggest practical heuristics for the estimation of the probability of a given edge being part of a skeletal core and experimentally validate these heuristics with respect to the ground truth.

Our main contributions are as follows:

1. We provide a novel MCMC algorithm for sampling skeletal cores of the graph uniformly at random.
2. We suggest heuristics that can be used to estimate the likelihood of an edge being part of a skeletal core and evaluate them experimentally.
3. We provide a novel algorithm for the estimation of the size of the skeletal core space and experimentally demonstrate the relationship between the number of skeletal cores in the graph and the core resilience of the network.
4. We demonstrate how the proposed algorithms might be used to identify moments of fundamental change of the k-core structure during a process of edge deletion.

2 Related Work

First introduced by Seidman in 1983 [26], the k-core of a graph $G = (V, E)$ is defined as the maximal connected subgraph such that any node in the subgraph has a degree at least k. k-cores are an important part of network analysis, and have been used for many tasks, including community detection [19], speedups of the graph algorithms [23], influence maximization [11], anomaly detection [28], prediction of protein functions [2], and others. k-cores have also been used to

gain deeper insight into a graph structure, including through visualization [12]. Much attention has been paid to the process of finding k-cores in various settings, including in streaming [25] or distributed settings [22].

A skeletal core of a graph G is defined as a minimal subgraph H of G such that all nodes in H have the same core number as in G [16]. The literature contains a greedy algorithm (discussed in more detail in Section 4.2) for generating skeletal k-core for the given graph and suggests several applications for skeletal cores [16]. However, the greedy algorithm for finding skeletal cores does not sample uniformly at random from the *whole space* of skeletal cores, and so may lead to bias in analysis. Moreover, because graphs may have many skeletal cores, simply generating one such skeletal core may not provide sufficient insight into the graph structure. The estimation of the *expected* properties of skeletal cores (e.g. expected size of skeletal core or expected overlap between skeletal cores) is a more useful tool for graph analysis, but it requires the ability to sample cores uniformly at random.

Skeletal cores are particularly important to the robustness of networks. While network robustness/resilience can be defined in many ways [8, 10], as appropriate to different contexts, of relevance to this work is the resilience of a graph's k-core structure against changes in the graph structure. k-core numbers are used as a way to quantify connectedness and importance of the nodes in the network, and core structure can be used to study the local density of the graphs. As such, it is important to estimate how these properties may be affected if the graph is changed: for example, how do random communications failures in a communications network affect overall connectivity?

[17] suggests the core resilience measure, which measures the propensity of nodes to change core number when edges from the graph are dropped, and [1] studies the impact of noise and sampling process on the k-core of a graph. The related k-core minimization problem [31] attempts to "break" the core structure of the network, and the similar problem of k-core maximization [30] tries to forcibly increase k-core numbers of the nodes.

Understanding when changes in the graph structure happen during some processes (e.g. edge removal or edge addition) has been a topic of research in network science since the early days of the graph analysis. Examples include research on changes in random networks [9], including the emergence of k-cores in the random graphs [24] and dynamics of the k-core during edge removal in the random graphs [13].

3 Background

Here, we introduce concepts necessary for the presentation of our work.

3.1 K-core

The k-core is defined as a maximal subgraph s.t. any node is connected to at least k neighbours in subgraph. A node's core number is the maximum value

k such that the node belongs to a k-core. The core numbers of a graph can be obtained using a 'peeling' k-core decomposition technique, which runs in $O(|E|)$ time [4].

3.2 Core Strength

Core strength was introduced in [17]. The core strength of a node u estimates the number of edges to same or higher-shell neighbors that can be removed from the node without it decreasing its core number. The proposed MCMC algorithm (Section 4.1) uses core strength to identify transitions; and the proposed algorithm uses it for estimating the number of skeletal cores.

Formally the core strength of u is defined as $CS(u) = |N_{\geq}(u)| - K(u) + 1$, where $N_{\geq}$ is the set of neighbours of u with a core number greater than or equal to that of u (i.e., those that support u's core number) and $K(u)$ denotes the core number of u.[1]

3.3 Core Valid Subgraph and Skeletal K-core

A Core Valid Subgraph CVS of a graph $G = (V, E)$ is defined as a spanning subgraph of G such that all nodes in G have the same k-core number in CVS as they do in G [16]. In other words, CVS preserves the core structure of G.

A skeletal k-core is defined as a minimal CVS: i.e., one for which removal of any edge would lead to at least one node decreasing its core number [16].
Of interest in the context of our work is the greedy algorithm, suggested by [16]. This algorithm serves as base for the algorithm in Section 4.2 and can be summarized in the next steps:

1. Identify a set of edges of G s.t. any edge in the set can be removed without k-core numbers decreasing; Let's denote this set as R.
2. Select one of these edges $e \in R$ and remove it from the G.
3. Recompute R and repeat 1-3, until $R = \emptyset$.
4. Return G as a skeletal core.

3.4 Core resilience

Core Resilience was proposed in [17] to measure the robustness of a network's k-core structure against random edge removal. It provides a way to compare the estimated ability of the core structure of different networks to withstand changes (for example, due to failure of the communication channels between nodes or because of network evolution). Consider two graphs: $G = (V, E)$ and $G' = (V, E')$, where G' is a subgraph of G formed by randomly deleting $p\%$

[1] Note that this sometimes overestimates the desired value, as loss of an edge (u, v) can trigger reductions in core numbers of other nodes, and thus lower the core number of other neighbors of u; however, computing the exact value is more computationally intensive.

of the edges from G. Denote the top $r\%$ of the nodes with the highest k-core numbers in G as V_r. The resilience $R_r^{(p)}$ of G can then be defined as: $R_r^{(p)} = \tau_b(\{((K(u,G),(K(u,G')), u \in V_r\})$, where $K(u,G)$ denotes the core number of node u in graph G, and τ_b denotes the Kendall-Tau [14] rank correlation (other rank correlations may be used). $R_r^{(p_1,p_2)}$ is defined as the mean core resilience as the percentage of dropped edges changes from p_1 to p_2 [17]: $R_r^{(p_1,p_2)} = \frac{\int_{p_1}^{p_2} R_r^{(x)}(G)dx}{p_2-p_1}$. We use core resilience in our experimental analysis to show the relationship between skeletal cores and the robustness of the graph.

4 Algorithms to Explore the Space of Skeletal Cores

Here, we introduce an MCMC-based algorithm for sampling skeletal cores uniformly at random, and then demonstrate how to estimate the number of skeletal cores in a graph. Exploring the space of skeletal cores in unbiased way might be useful for different applications. Let's consider the scenario of the pandemic of a contagious virus which spreads through the network of personal interactions. As was shown by the research of COVID-19[27], k-cores of high complexity are known to sustain the outbreak even if the network became partially disconnected, thus, estimation of *which* edges are more important for the k-core could be useful to minimize the spread of the disease. While skeletal cores of the personal interaction network provide important insight regarding these edges, application of previously suggested greedy algorithm will provide a biased sample of the skeletal cores, leading to the non-optimal decision-making. Our MCMC algorithm doesn't have this drawback and is guaranteed to provide unbiased sampling.
Suggested MCMC algorithm uses the notion of *core strength* [17] (discussed in more details in the Section 3.2) to identify transitions between different skeletal k-cores.

In our experimental analysis, we show that the core resilience of a graph is strongly correlated with its skeletal core properties.

4.1 Sampling Skeletal Cores Uniformly at Random

In this Section, we describe an MCMC algorithm for sampling skeletal k-cores uniformly at random. At a high level, the proposed method is described as following: First, begin from any skeletal core T_0 (for example, one obtained by the greedy algorithm in [16]). This skeletal core is the initial state of a Markov Chain M. Next, randomly transform T_0 into another skeletal core T_1, or stay at T_0, with probabilities of D/D_{max} and $1 - D/D_{max}$ correspondingly, where D stands for the number of possible transformations from the current state. D_{max} needs to be bigger than any possible number of transitions from one state. This is equivalent to conducting a random walk over M. Repeat the procedure until the process converges to the stationary distribution. If transition probabilities are defined correctly, this stationary distribution will be an uniform distribution over the space of skeletal cores. Once the stationary distribution is reached, return the current skeletal core.

The key idea behind the proposed algorithm is to correctly transition between skeletal cores of the graph. When at a skeletal core T, no edge can be deleted from T without affecting core numbers (because T is skeletal), unless another edge (or edges) is added to compensate. When these replacement edges are added, this may necessitate further removal of edges to ensure that the new subgraph is still skeletal.

More formally, the proposed algorithm consists of the next steps:

1. Begin from some skeletal core T_0 of the original graph $G = (V, E)$. We can obtain this skeletal core from the original graph by using a greedy algorithm, like that proposed in [16], or in any alternative way.
2. Initialize the set of possible transitions $R_0 = \emptyset$.
3. To identify possible transitions, iterate over all edges $(u, v) \in T$ and generate all skeletal cores that can be obtained from T by removal of an edge (u, v), followed by the addition of an edge $(u, i) \in G \backslash T$ or an edge $(v, j) \in G \backslash T$ (or both of them) and corresponding removal of $(i, p) \in T$ and/or $(j, k) \in T$, if needed. Add these transitions to R.
4. Similarly, for all edges $(u, v) \in G \backslash T$, generate all skeletal cores, that can be obtained from T by addition of (u, v), the removal of $(u, i) \in T$ or $(v, j) \in T$ (or both), and corresponding addition of $(i, p) \in G \backslash T$ and/or $(j, k) \in G \backslash T$. Add these transitions to R.
5. Select one of the transitions from R with equal probability or stay at the current state with probability $1 - D/D_{max}$, where D_{max} needs to be bigger than any possible number of transitions from one state. It's possible to bound D_{max} in several ways - for example, it's trivial to show that $D_{max} \leq |E| * d_m^4$, where d_m is the highest degree in the graph [2].
6. Repeat steps 2-5 multiple times, until Markov Chain converges. There are different ways to diagnose the convergence of MCMC algorithms. [6]. One simple example could be running several instances of Markovian chains and comparing their outputs.
7. Return the current skeletal core.

We need to iterate over all edges and consider d_m^4 of possible transitions for every edge in the worst-case scenario, hence running time for one step is bounded by $O(|E| * d_m^4)$. The overall running time of the proposed algorithm depends primarily on the mixing time of the Markov chain M. While we do not propose proof that M is rapidly mixing, experiments suggest that the stationary distribution is reached relatively fast in most cases. Nonetheless, the main disadvantage of the algorithm is running time, which makes it prohibitively expensive to run on big graphs.

Theorem 1. *The described algorithm will sample skeletal cores u.a.r.*

Proof of the theorem can be found in the Section 5.

[2] Experimentally this bound proved to be very loose.

def $estimator(G = (V, E) : Graph, K : HashTable)$**:**
  $R = \emptyset$; $c = 0$; $d = 1$
  repeat
    $CS = getCoreStrength(G)$
    $R = \{(u, v) \in E : (CS[u] > 1 \wedge CS[v] > 1) \vee$
    $\vee (CS[u] > 1 \wedge K(u) < K(v))) \vee (CS[v] > 1 \wedge K(v) < K(u)))\}$
    $d* = |R|$; $c+ = 1$
  until $R! = \emptyset$
  return $d/factorial(c)$
def $estimate\_size(G = (V, E) : Graph, K : HashTable, N : int)$**:**
  $S = List()$; $W = List()$
  for $i = 0; i < N; i++$ **do**
    $S.add(estimator(G, K))$
  end
  return $AVG(S)$

Algorithm 1: Number of skeletal cores in the graph

4.2 Estimating the Number of Skeletal Cores

Analyzing the properties of the skeletal cores is important for many applications. For instance, the k-core can be seen as a form of the equilibrium in a natural game-theoretic model [5], and, correspondingly, the skeletal core can be seen as the minimal edge-induced subgraph that still maintains this equilibrium, and question of how this subgraph looks *on average* is crucial for understanding the network as a whole. Another example could be using the average size of skeletal cores to find "breaking point" of the k-core structure, as is demonstrated in Section 6.4.

Because the MCMC algorithm is computationally expensive, here we suggest an alternative approach for the estimation of the *expected* properties of skeletal cores. We provide an algorithm that provides the expected number of skeletal k-cores in the graph, and explain how it can be used to estimate properties of skeletal cores (e.g. average size) without relying on the MCMC approach. An outline of the algorithm can be found at algorithm 1, further referred as "Greedy Estimator". The algorithm takes a graph and k-core numbers of all nodes as input and returns the expected number of skeletal cores in the graph. General idea behind the algorithm is as follows: we "unravel" the DAG H, formed by transitions of the greedy algorithm, discussed in 3.3 to the tree and count its leaves, adjusting our estimate for overcounting, caused by "unravelling".

4.3 Proof of the algorithm correctness

Theorem 2. *Algorithm 1 returns the expected number of skeletal k-cores in the graph.*

Proof. Denote the original input graph as G. Suppose that we use an existing greedy algorithm, suggested in [16] (see Section 3.3) to obtain skeletal core S

from it. Let's create a new graph H, where nodes denote graphs that can be obtained during the process above and edges denote transitions between these graphs. H is directed and acyclic. Nodes without out-edges are skeletal cores of G (by definition). The root of H is G itself. Let's modify the H by "unravelling" the DAG to the tree, by making copies of all nodes that have more than one parent. Let's denote the resulting graph as H_m. In this case, getting the expected number of terminal nodes (leaves) in H_m could be done easily with the algorithm suggested by Knuth in [15] for the estimation of the space of the backtracking tree. In this algorithm we initialize a counter D with 1 at the root of the tree (which denotes the original graph G) and perform a random walk down the tree, multiplying D by an out-degree of every node on the path, until we reach the leaf. When the leaf is reached, the counter will contain the expected number of the leaves in the graph [15].
However, every skeletal core in H_m is copied (and counted) several times. As edges removed from G to obtain any skeletal core S_i may have been removed in any order, there are $A_i!$ ways to reach S_i from G in H, where A_i is a number of edges removed from G to obtain S_i. As any of these paths will lead to the independent copy of the skeletal core S_i in H_m, we know that S_i will be counted $A_i!$ times in H_m. To account for this, we divide our estimate by $A_i!$. To get a better estimate, we repeat experiment n times and use D_{avg} as the final estimate.

Theorem 3. *The running time of Algorithm 1 is* $O(|E|)$.

Proof. First, consider the running time of the function *estimator*. To compute core strength, the function requires $O(|E|)$ time in the first iteration. For subsequent iterations, core strength values can be updated in $O(1)$ after every edge deletion (as at most two values of Core Strength will be affected). Identifying a set of edges R takes $O(|E|)$ time the first time, but we can update it quickly if the selected data structure allows us to quickly access and remove edges with one known endpoint. The function *estimate_size* takes $O(k|E|)$ time, where k is the number of samples. Assuming that k is a small fixed constant s.t. $k << |E|$, the overall running time of the algorithm is $O(|E|)$.

4.4 Estimation of expected properties of skeletal core

We can use a modified version of Algorithm 1 to obtain the expected properties of skeletal cores in the graph. Suppose that we want to get the expected value of property P of the skeletal core (e.g. a number of edges or average degree).

Denote $P(S_i)$ as the value of P for the core S_i and define $C = \sum_{S_i \in S} P(S_i)$. C is the sum of P of all skeletal cores in the space S. Assign 0 as the *cost* of any node in the H which is non-terminal (i.e., not a skeletal core), and assign $P(S_i)$ as the cost of the skeletal core S_i.

In this case, an expected estimate of *total cost* C for one Monte Carlo search will be $P(S_i) * D$ [15], and over multiple experiments it will be $C = \sum_{i=0}^{n} P(S_i) * D_i/n$. The expected value of P will then be $E[P] = C/D_{avg} = P(S_i) * D_i/D_{avg}$.

4.5 Normalized number of skeletal cores

For comparative purposes we introduce a novel metric of a normalized number of skeletal k-cores. Denoting the expected size of the space of skeletal cores as e_{est} and the expected number of edges in the skeletal core as s_{exp}, we can define a *normalized* number of skeletal cores in the graph as $e_n = e_{est}/\binom{|E|}{s_{exp}}$.
Denominator here is the total number of possible subgraphs in G that have number of edges of s_{exp}.

5 Proof for sampling of skeletal k-cores u.a.r.

In this section, we will provide proof for the algorithm for the uniform sampling of the skeletal cores based on the Markov Chain sampling. First, let's assume that we have a skeletal core T and graph G. Let's define and discuss two types of permutations of T which may bring us a new skeletal core:

1. Permutation, based on removal of specific edge $(u, v) \in T$;
2. Permutation, based on addition of specific edge (u, v) s.t. $(u, v) \in G/T$;

5.1 Permutation, based on the edge removal

1. Let's consider removal of an edge (u, v) from the T. After edge removal from T core strength of u and/or v decreases by 1.
2. For the case when $K(u) = K(v)$ (both u and v belong to the same k-core) something of the following is true:
   (a) If before the removal $CS(v) > 1$ and $CS(u) > 1$, then T is not minimal, which is impossible (by definition of the skeletal core).
   (b) If before the removal $CS(v) > 1$ and $CS(u) = 1$ (the other case is symmetric), we need to compensate by adding one edge (u, k) from G which would increase CS(u). Additionally (u, k) might increase $CS(k)$ (if $K(k) = K(u)$).
   In this case:
      i. If in original skeletal core $CS(k) = 1$ we may need to remove some other edge, connected to k.
      ii. If in original skeletal core $CS(k) > 1$ after incrementing of CS(k) by 1 we will not be allowed to remove any other edge from k. T is minimal.

We can count all possible transitions.

1. If before the removal (u, v) $CS(v) = CS(u) = 1$, we need to compensate by adding two edges from G which would increase $CS(u)$ and $CS(v)$ correspondingly. We can count combinations of these edges. For any edge (u, t) or (u, b) if $CS(t) > 1$ (or $CS(b) > 1$) we need to count combinations of the edges that can be removed from t or b (so overall, we may remove (u, v), add up to two edges (u, k) and (u, t), and remove up to two edges connected to the k and b.
   While counting all possible permutations we might need to account for two corner cases.
   (a) If we have triangle (u, k, v) after removal of (u, v), and addition of (u, k) and (v, k) if $CS(k) = 1$ we need to remove up to two existing edges from k (if possible). We need to account for all possible combinations of two edges, if two edges can be removed from k (that happens when we have more than two edges that can be removed from the k), or we need to account for all possible selections of one edge (if only one edge exists that can be removed from the k). It's also possible that we are not allowed to remove any edge from k.

   (b) Another corner case is when we have to remove an edge (u, v), add (u, i) and (v, j) and we have existing edge (i, j) in the T. In this case, we have an additional possible case when we might be able to remove only (i, j) instead of removing two edges connected to i or j correspondingly. We also need to account for this.
2. For the case when $K(u) > K(v)$ (one of the nodes belongs to the higher k-core than the other) all cases are almost identical to the cases above. The only difference is that we need to fix only $CS(v)$, as $CS(u)$ will not be affected by the removal of (u, v).

5.2 Permutation, based on the edge addition

Permutations, based on the edge addition are supposed to be "symmetric" to the edge removal permutation (so we may be able to transform T_{t+1} back to the T_t, if T_t was be transformed to the T_{t+1} with edge removal).

1. Let's consider addition of an edge (u, v). After edge addition core strength of u and/or v increases by 1.
2. For the case when $K(u) = K(v)$ (both nodes are from the same k-core) of the following is true:
   (a) If before the addition of the (u, v) $CS(u) = 1$ and $CS(v) > 1$, we can remove one of the edges connected to u, if their second endpoint has $CS > 1$. For any $(u, v) \in G$ we can count these edges. Additionally, we can remove edge (u, t) with $CS(t)$=1 if we can replace it with another edge from G, connected to t (to have a "symmetric" transformation). We also need to account for these transformations.
   (b) If before the addition of (u, v) $CS(v) > 1$ and $CS(u) = 1$ case is symmetric to previous one.
   (c) If before the addition of (u, v) $CS(v) > 1$ and $CS(u) > 1$, we cannot remove any of the edges that we had (as T was already minimal, which means that any edge connected to u or v already has CS of the second endpoint as 1 (otherwise T wouldn't be skeletal). This transformation doesn't bring us a new skeletal core, so we are not going to add this edge.
   (d) If before the addition of (u, v) $CS(v) = 1$ and $CS(u) = 1$, we can remove one of the edges connected to u, if their second endpoint has $CS > 1$, and we can remove one of the edges, connected to v, if their second endpoint has $CS > 1$. For any $(u, v) \in G$ we can count all combinations of these edges, substructing number of triangles (u, v, k) when $CS(k) = 2$ (so we can remove only one of the edges (u, k) and (v, k). We also need to account for the corner case, discussed in the edge removal case (when we might be allowed to add only one edge instead of two edges). Additionally, we need to count edge combinations that we can add to replace edges (u, t) if $CS(t) = 1$. Note - counting requires in the worst case scenario $O(d^2|E|)$.

   Now, after we counted all these possibilities, we can select one of the **unique** states above with equal probability or stay at the current state with probability $1 - D/D_{max}$. Note, that transitions with edge addition and edge removal

are kind of inverse - if we move from T to T_1 with the addition of some edge e_1 (and consequent changes according to the rules above), we may move back with the removal of the edge e_1 according to the rules above.

3. For the case when $K(u) > K(v)$ (one of the nodes belongs to the higher k-core than the other) all cases are almost identical to the cases above. The only difference is that we need to fix only $CS(v)$, as $CS(u)$ will not be affected by the addition of (u, v).

5.3 Proof of the connectivity of the transitions

We can show that with the transformations above all skeletal cores are connected via skeletal core with maximal cardinality T_{max}. Let's assume that we have arbitrary skeletal core $T = (e_i, e_{i+1}, ...e_k)$ and skeletal core with maximal cardinality (number of edges) $T_{max} = (e_j, e_{j+1}, e_b)$ s.t. $T \neq T_{max}$.

Let's rewrite each skeletal core in form of a binary vector of length $|E|$, where every bit denotes the presence or absence of corresponding edge. Let's define the distance between T and T_{max} as $T \oplus T_{max}$ (number of different bits between two vectors). Let's show that for any T we always can create T_p which has strictly smaller distance to the T_{max} then T (in other words $d(T_{max}, T_p) < d(T_{max}, T)$).
As $T \neq T_{max}$ we going to have some edge $e_1 = (u, v)$ that we have in the T, but don't have in T_{max} (otherwise, T would be a subset of T_{max}, which means that T_{max} is not skeletal, which is impossible by definition). In this case, we let's consider the removal of an e_1 from T. After we remove it, we need to add one or two edges to the T to support the core strength of u and/or v (and maybe remove extra edges, connected to endpoints of these edges). It's impossible to have all edges that are connected to the u, v in T_{max} in T, as in this case e_1 would be absent from the T in the first place. That means that we have at least one or two edges $e_2 = (u, k) \in T_{max}$ and/or $e_3 = (v, k) \in T_{max}$, which we don't have in T. If we remove e_1 and add e_2 (and e_3, if needed), the distance between T and T_{max} is going to decrease by 2 or 3. Even if we remove two edges connected to endpoints of e_2 and e_3. distance is still going to decrease by 1 in the worst case (if these edges belong to the T_{max} - then we add to the T_1 two edges that we have in T_{max}, remove two edges from the T_1 that we have in T_{max} and we remove one edge from T_1 that we don't have in T_{max}).
$|T_1| = |T_{max}|$. As we assume that $T_1 \neq T_{max}$, we are going to have at least one edge in T_1 that we don't have in T. We still can remove it and all other reasoning is the same as above. Even in the worst case we still going to decrease the distance between the T_1 and T_{max} by at least 1.

As the minimal distance change is 1, we can reach from any skeletal core to maximal skeletal core T_{max} in at most $|E|$ steps, and we can reach from any maximal skeletal core T_{max} to any other skeletal core, as transitions are symmetrical (as we defined transitions in the way that inverse transition always exists).

6 Experiments and analysis

In this section, we demonstrate a close relationship between skeletal k-cores and core resilience; and show how skeletal cores can be used to analyze the "breaking point" of the k-core structure.

6.1 Datasets

Networks, used in the experiments are listed in Table 1. We compare networks from different domains: AS denotes networks from autonomous systems; P2P stands for peer-to-peer, BIO indicates graphs from bioinformatics; CA denotes co-authorship networks; INF is used for infrastructure-related graphs; CO denotes collaboration networks; SOC indicates social networks; TECH stands for technological networks; WEB is used for internet network; EMAIL indicates email networks; and MISC contains a set of networks that don't belong to the categories above.

Type	Network	$\|V\|$	$\|E\|$	k_{max}	Type	Network	$\|V\|$	$\|E\|$	k_{max}
AS	auto_as19990111†	549	1249	11	CO	arena_jazz‡	198	2742	29
	auto_as19980318†	3455	6168	10	SOC	wiki‡	889	2914	9
	auto_as19971108†	3015	5156	9		hamsterer‡	2426	16630	24
	oregon010331†	10670	22002	17		musae_facebook†	22470	170823	56
P2P	gnutella08†	6301	20777	10		musae_git†	37700	289003	34
BIO	dmela‡	7393	25569	11	TECH	tech_routers‡	2113	6632	15
	protein‡	1870	2203	5		whois‡	7476	56943	88
CA	erdos‡	5094	7515	7		tech_pgp‡	10680	24316	31
	netscience§	1464	2744	19	WEB	webspam‡	4767	37375	35
	ca-HepPh‡	12008	118521	238	EMAIL	email_enron†	36692	183831	43
	ca_grq†	4158	13422	43		email_EuAll†	265214	364481	37
INF	openflights‡	2939	15677	28	MISC	moreno_innovation§	245	927	6
	inf-power‡	4941	6594	5		norwegian(net1m)§§	1421	3855	11
						moreno_oz‡	217	2345	14

Table 1. Datasets. $|V|$ denotes number of nodes, $|E|$ denotes number of edges, k_{max} denotes highest k-core. † denotes SNAP as a source of the network, ‡ stands for NetworkRepository, § stands for KONECT, §§ stands for Netzschleuder.

6.2 Algorithm Validation

Before using the proposed algorithms to perform network analysis, we demonstrate that they are effective at the desired objectives. First, we show that

the MCMC sampling algorithm reaches its stationary distribution quickly. Any MCMC algorithm needs several steps before convergence. While we do not provide theoretical proof that Markovian chain, defined by the suggested algorithm is rapidly mixing, experiments demonstrate fast convergence of the algorithm. To show the speed of convergence of the algorithm experimentally, we perform a sampling of skeletal cores for a different number of steps, setting the number of steps to be equal to the fixed fractions of the number of edges in the original graph.

Intuitively, after the convergence was achieved, the distribution of the properties of skeletal cores will also be stable. In Figure 1 distribution of sizes of skeletal cores is plotted for a different number of steps of the algorithm. Due to the lack of space, we show plots only for 3 networks. For the higher number of steps distribution of skeletal k-core sizes look very similar, which most likely[3] indicate convergence.

As was seen in Section 4.4, we can compute the expected values of different properties of skeletal cores. To test this algorithm, we compare these estimated properties to the average properties of skeletal cores sampled uniformly at random using MCMC algorithm. Due to space limitations, we perform only an estimation of the expected number of edges in the skeletal cores for selected graphs. Results can be seen in Table 2. We compute E_{avg} as a simple average over sizes of skeletal cores, sampled with a greedy algorithm from [16], E_{adj} is the expected value over skeletal cores (method from Section 4.4), E_u is obtained by sampling skeletal cores u.a.r with MCMC algorithm. For most graphs suggested expected estimate is closer to MCMC estimate.

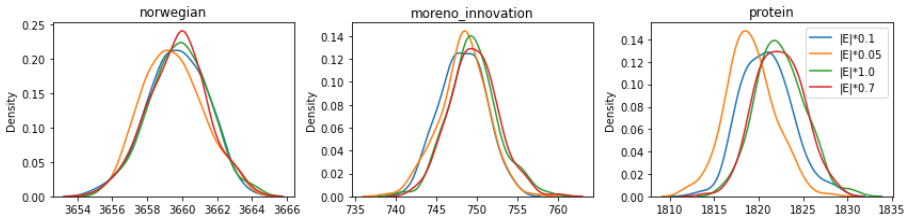

Fig. 1. Distribution of size of skeletal cores, sampled with a different number of steps of MCMC (number of steps are set up as fixed fractions of the number of edges in the graph). Colors denote different number of transitions made by an algorithm until sample was taken.

6.3 Skeletal core analysis

Here, we experimentally validate the theoretical results of the algorithm to estimate the number of skeletal cores, as presented in Section 4.2. For every graph,

[3] as was discussed in [6] convergence diagnostics for MCMC methods don't *guarantee* convergence and should be seen more as statistical analysis instead.

Network	E_{avg}	E_{adj}	E_u
auto_as19971108	4717.13	**4724.53**	4722.28
auto_as19980318	**5649.84**	5648.00	5655.04
auto_as19990111	**1130.45**	1128.28	1131.47
GrQc	11926.69	**11936.83**	11953.45
netscience	2628.29	**2629.01**	2630.30
norwegian	3657.30	**3657.82**	3660.14
wiki	2464.67	**2466.16**	2470.61
erdos	6864.28	**6870.62**	6877.34
protein	1816.31	**1817.76**	1821.81
inf-power	5286.41	**5288.73**	5314.86

Table 2. Expected number of edges in skeletal cores. E_{avg} is simple average over skeletal cores, sampled with a greedy algorithm, E_{adj} is the expected value over skeletal cores (method from Section 4.4), E_u is obtained by MCMC algorithm.

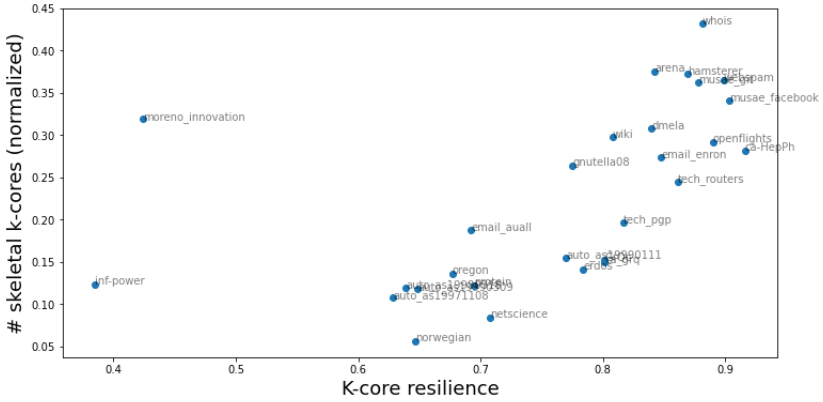

Fig. 2. Normalized number of skeletal cores vs Core Resilience

we sample 50 skeletal cores using "Greedy Estimator" from Section 4.2 and compute a normalized number of skeletal cores as shown in 4.5. Results are plotted against $R_{50}^{(0,50)}$ (Core Resilience computed as described in 3.4) in Figure 2.
As can be seen, there is a strong correlation between the two measures. For most of the graphs, the more skeletal cores we have in the graph, the highest core resilience is. An interesting outlier is *moreno_innovation* network - despite having a lot of skeletal cores, core resilience is relatively low. One possible explanation could be the existence of a very "fragile" region in the graph, which lowers Core Resilience significantly for the whole network.
While the number of skeletal cores is an important metric by itself, relationships between different skeletal cores of the graph (e.g. overlap) can provide an important insight into the network's robustness. One example can be found in Figure 3. As can be seen in this histogram, the *moreno_innovation* network has many edges that belong only to a *fraction* of skeletal cores of the network. In other words, there are many edges that can be useful to skeletal cores, but are not always necessary. This property could be the alternative explaination for unusually high number of skeletal cores in the graph, as was seen in Figure 2, because these edges might support a higher number of skeletal cores comparatively to other networks.

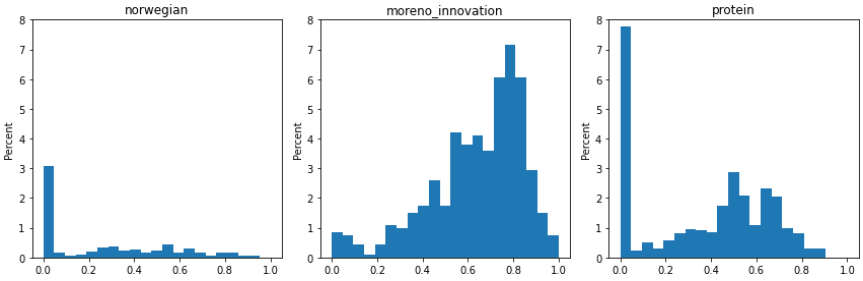

Fig. 3. Fraction of skeletal cores that edge belong to. y-axis denotes the percentage of edges in the graph that belong to a given fraction of skeletal cores. Results were obtained by sampling 200 skeletal cores with MCMC. Edges that belong to all skeletal cores, as a CS of one of the endpoints in the original graph is equal to 1 are ignored.

Estimating probability of edge being part of skeletal k-core. The likelihood that an edge is part of a skeletal core can be used for visualizations of skeletal cores or evaluation of the "centralization" of the core structure [16]. As our work is the first that can estimate this likelihood directly (using an MCMC algorithm) we evaluate heuristics introduced by [16] which we denote as *Centralized Skeletal Score* heuristics (CSS-heuristics) and compare with our proposed method.

In our heuristics we naively approximate the probability that an edge, connected to a node $u \in V$ will be part of a skeletal core.

Let's denote the number of edges that can be removed from node u without dropping k-core number of *any* noded:

$$\omega(u) = |\{v \in N(u) : K(v) > K(u) \vee (K(u) = K(v) \wedge CS[v] \neq 1)\}| \quad (1)$$

Similarly, let's define the number of edges that cannot be removed from the u and will be present in any skeletal core:

$$\gamma(u) = |\{v \in N(u) : K(v) = K(u) \wedge CS[v] = 1\}| \quad (2)$$

As we know that in skeletal core u will have at least $K(u)$ neighbours, we need to "select" at least $K(u) - \gamma(u)$ nodes from $\omega(u)$ for skeletal core to keep k-core numbers of original graph. Thus, for node u:

$$\rho(u) = \begin{cases} max(\frac{K(u)-\gamma(u)}{\omega(u)}, 0) & \text{if } \omega(u) \neq 0 \\ 1 & \text{if } \omega(u) = 0 \end{cases} \quad (3)$$

For an edge $(u, v) \in E$ we define heuristics as:

$$\rho((u, v)) = \begin{cases} max(\rho(u), \rho(v)) & \text{if } K[u] = K[v] \\ \rho(u) & \text{if } K[u] < K[v] \\ \rho(v) & \text{if } K[u] > K[v] \end{cases} \quad (4)$$

For convenience, these heuristics will be referred to as "ρ-heuristics". In Figure 4, heuristics from [16] and the proposed ρ-heuristics are plotted against the ground truth, as estimated with the MCMC algorithm. As can be seen, the correlation between ρ-heuristics and ground truth is significantly higher as compared to the earlier CSS heuristics.

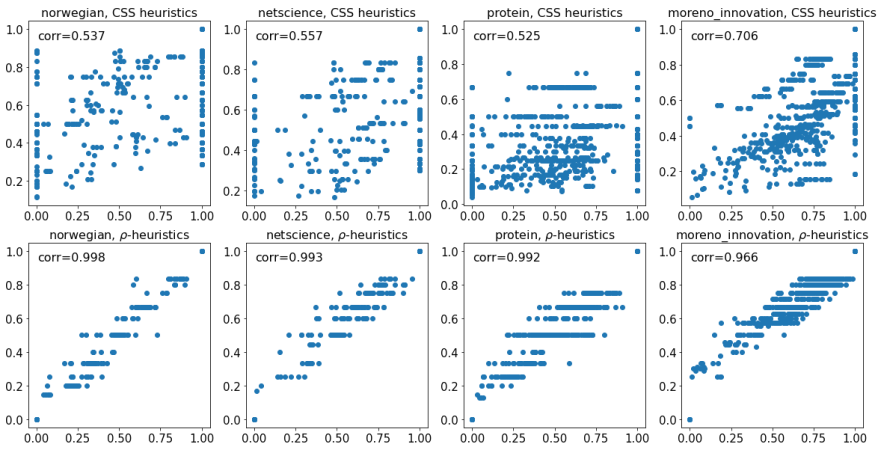

Fig. 4. Comparison of the heuristics

6.4 Identifying the "breaking point" of the k-core structure

The normalized number of skeletal cores from Section 6.2 can be interpreted as follows: after removal of $|E| - s_{exp}$ edges uniformly at random, e_n fraction of the resulted subgraphs is expected to maintain k-core structure (as not all skeletal cores are going to have the same size of e_{est}, that's more of the upper bound). But if we continue removing edges past this point, the graph will have fewer edges than the expected size of skeletal cores, thus, edges will be *fundamentally* unable to support the k-cores. In Figure 5 can be seen that the average size of the skeletal core provides a good estimate of when the k-core structure starts disappearing during the edge removal process.

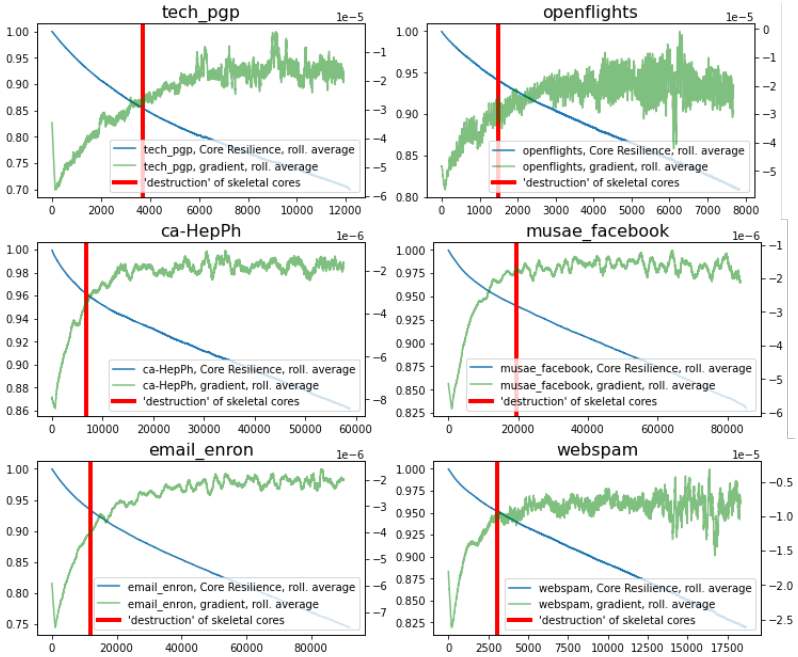

Fig. 5. Skeletal cores and "breaking point" of the network. x-axis denotes number of edges removed, left y-axis denotes Core Resilience of the network for the number of edges removed, right y-axis denotes gradient of Core Resilience. Point of "Skeletal core "destruction" is computed as $|E| - s_{est}$ for the given graph.

7 Conclusion

In this paper, we introduced a novel MCMC algorithm for sampling skeletal cores uniformly at random as well as a novel algorithm for estimating the expected number of skeletal cores and their properties. We experimentally demonstrate

the relationship between the skeletal core structure and the core resilience of the graph, show that skeletal cores can be used to find the "breaking point" of the k-core structure and suggest a heuristic to estimate the likelihood of an edge being part of the skeletal core.

References

[1] Abhijin Adiga and Anil Kumar S Vullikanti. "How robust is the core of a network?" In: *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part I 13*. Springer. 2013, pp. 541–556.

[2] Md Altaf-Ul-Amine et al. "Prediction of protein functions based on k-cores of protein-protein interaction networks and amino acid sequences". In: *Genome Informatics* 14 (2003), pp. 498–499.

[3] José Ignacio Alvarez-Hamelin et al. "k-core decomposition: A tool for the visualization of large scale networks". In: *arXiv preprint cs/0504107* (2005).

[4] Vladimir Batagelj and Matjaz Zaversnik. "An o (m) algorithm for cores decomposition of networks". In: *arXiv preprint cs/0310049* (2003).

[5] Kshipra Bhawalkar et al. "Preventing unraveling in social networks: the anchored k-core problem". In: *SIAM Journal on Discrete Mathematics* 29.3 (2015), pp. 1452–1475.

[6] Stephen P Brooks and Gareth O Roberts. "Assessing convergence of Markov chain Monte Carlo algorithms". In: *Statistics and Computing* 8.4 (1998), pp. 319–335.

[7] Jonathan Cohen. "Trusses: Cohesive subgraphs for social network analysis". In: *National security agency technical report* 16.3.1 (2008).

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

[9] Paul Erdős, Alfréd Rényi, et al. "On the evolution of random graphs". In: *Publ. Math. Inst. Hung. Acad. Sci* 5.1 (1960), pp. 17–60.

[10] Scott Freitas et al. "Graph vulnerability and robustness: A survey". In: *IEEE Transactions on Knowledge and Data Engineering* (2022).

[11] Mohammed Ali Al-garadi, Kasturi Dewi Varathan, and Sri Devi Ravana. "Identification of influential spreaders in online social networks using interaction weighted K-core decomposition method". In: *Physica A: Statistical Mechanics and its Applications* 468 (2017), pp. 278–288.

[12] Priya Govindan et al. "The k-peak decomposition: Mapping the global structure of graphs". In: *Proceedings of the 26th International Conference on World Wide Web*. 2017, pp. 1441–1450.

[13] Mami Iwata and Shin-ichi Sasa. "Dynamics of k-core percolation in a random graph". In: *Journal of Physics A: Mathematical and Theoretical* 42.7 (2009), p. 075005.

[14] Maurice G Kendall. "A new measure of rank correlation". In: *Biometrika* 30.1/2 (1938), pp. 81–93.

[15] Donald E Knuth. "Estimating the efficiency of backtrack programs". In: *Mathematics of computation* 29.129 (1975), pp. 122–136.

[16] Ricky Laishram and Sucheta Soundarajan. "On Finding and Analyzing the Backbone of the k-Core Structure of a Graph". In: *2022 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2022, pp. 1017–1022.

[17] Ricky Laishram et al. "Measuring and improving the core resilience of networks". In: *Proceedings of the 2018 World Wide Web Conference*. 2018, pp. 609–618.

[18] RD Luce and Albert D Perry. "A method of matrix analysis of group structure". In: (1949).

[19] Irene Malvestio, Alessio Cardillo, and Naoki Masuda. "Interplay between k-core and community structure in complex networks". In: *Scientific reports* 10.1 (2020), pp. 1–12.

[20] Sourav Medya et al. "A game theoretic approach for core resilience". In: *International Joint Conferences on Artificial Intelligence Organization*. 2020.

[21] Robert J Mokken et al. "Cliques, clubs and clans". In: *Quality & Quantity* 13.2 (1979), pp. 161–173.

[22] Alberto Montresor, Francesco De Pellegrini, and Daniele Miorandi. "Distributed k-core decomposition". In: *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on principles of distributed computing*. 2011, pp. 207–208.

[23] Chengbin Peng, Tamara G Kolda, and Ali Pinar. "Accelerating community detection by using k-core subgraphs". In: *arXiv preprint arXiv:1403.2226* (2014).

[24] Boris Pittel, Joel Spencer, and Nicholas Wormald. "Sudden emergence of a giantk-core in a random graph". In: *Journal of Combinatorial Theory, Series B* 67.1 (1996), pp. 111–151.

[25] Ahmet Erdem Sarıyüce et al. "Streaming algorithms for k-core decomposition". In: *Proceedings of the VLDB Endowment* 6.6 (2013), pp. 433–444.

[26] Stephen B Seidman. "Network structure and minimum degree". In: *Social networks* 5.3 (1983), pp. 269–287.

[27] Matteo Serafino et al. "Superspreading k-cores at the center of COVID-19 pandemic persistence". In: *medRxiv* (2020), pp. 2020–08.

[28] Kijung Shin, Tina Eliassi-Rad, and Christos Faloutsos. "Corescope: Graph mining using k-core analysis—patterns, anomalies and algorithms". In: *2016 IEEE 16th international conference on data mining (ICDM)*. IEEE. 2016, pp. 469–478.

[29] Haohua Zhang et al. "Using the k-core decomposition to analyze the static structure of large-scale software systems". In: *The Journal of Supercomputing* 53 (2010), pp. 352–369.

[30] Zhongxin Zhou et al. "K-Core Maximization: An Edge Addition Approach." In: *IJCAI*. 2019, pp. 4867–4873.

[31] Weijie Zhu et al. “K-core minimization: An edge manipulation approach”. In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 2018, pp. 1667–1670.