June 2020

The Resilience of k-Cores in Graphs

Ricky Laishram
Syracuse University

ABSTRACT

A k-core of a graph is defined as the maximal subgraph such that all the nodes in the subgraph has at least k neighbors within the subgraph. k-core have been used in a number of applications ranging from anomaly detection and finding influential spreaders in social networks, to studying the robustness of financial and ecological networks.

In our work, we study the effect of missing data (edges or nodes) to the k-core of a graph. In particular, we study three different type of changes. The first type of change is the *core structural change*, in which the rank order of nodes by k-core number is changed. The second type is the change in the size of the k-core, and it is called the *core minimization*. The final change we study is called *graph unraveling*, and it is associated with a change in the size of the graph itself.

We study a graph's resilience changes – how can we efficiently tell if a graph is resilience to each of these changes? We then use our analysis to propose novel algorithms to make small modifications to a graph with the objective of maximizing its resilience. We show experimentally that our proposed method outperforms all considered baselines methods on real-world graphs.

Finally, we study the organization of the different k-shells in a graph (for different values of k). For example, in some graphs there are many connections between shells, while in other graphs, the shells are mostly disconnected from one another. We prove that this organization can have a huge impact on the resilience of a graph to the three changes we studied.

The Resilience of k-Cores in Graphs

by

Ricky Laishram

B.E., Birla Institute of Technology, 2010

M.S., Syracuse University, 2015

DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer & Information Science & Engineering

Syracuse University

June, 2020

This dissertation is dedicated to my parents for always supporting and believing in me.

Appreciation

I would like to express my deepest appreciation for my PhD advisor, Dr. Sucheta Soundarajan. This dissertation would not have been possible without her continual guidance and support.

I would also like to thank Dr. Chilikuri Mohan and Dr. Kishan Mehrotra, who taught me so much about research. Their guidance when I started my PhD has been immensely valuable.

I am also thankful to all my research collaborators (Dr. Ahmet Erdem Sarıyüce of University at Buffalo, Dr. Tina Eliassi-Rad of Northeastern University, Dr. Ali Pinar and Dr. Jeremy D Wendt of Sandia National Lab) for the valuable discussion, advice and ideas.

I also extend my sincere thanks to all my friends in the SUNS and SENSE labs, who made my time in Syracuse so memorable.

Contents

List of Figures

List of Tables

The Resilience of k-Core in Graphs

Chapter 1

Introduction

A graph, $G = \langle V, E \rangle$, is as structure that consists of a set of vertices, V, and connections between pairs of vertices, denoted by E. In the literature, the terms “network”, “nodes” and “links” are interchangeably used for graph, vertices and edges respectively. In the real-world we encounter various type of data that can be modeled as a graph. For example, a social network can be represented by a graph where the nodes are the users and edges represent friendships. Another example is a protein network; the proteins are the nodes and two nodes are connected by an edge if they have an interaction.

When the edges have directions, the graph is called a directed graph; otherwise it is called an undirected graph. Examples of directed graphs includes Twitter followers network, email networks, food web etc.; and Facebook friendship network, protein interaction networks, etc. are some examples of undirected networks. In this dissertation, when we talk about graph we are referring about undirected graphs. In the case of directed graphs, similar ideas as presented can be extended after we take into account the edge direction.

To visualize, graphs are represented by circles (or dots), representing the nodes, and lines connecting them, representing the edges. In the case of directed graphs, the edges have an arrow pointing denoting its direction. Figure 1.1 shows a visualization of the Zachary Karate Club Network [97]. Here, the nodes are people, and two nodes are connected if they interact with each other.

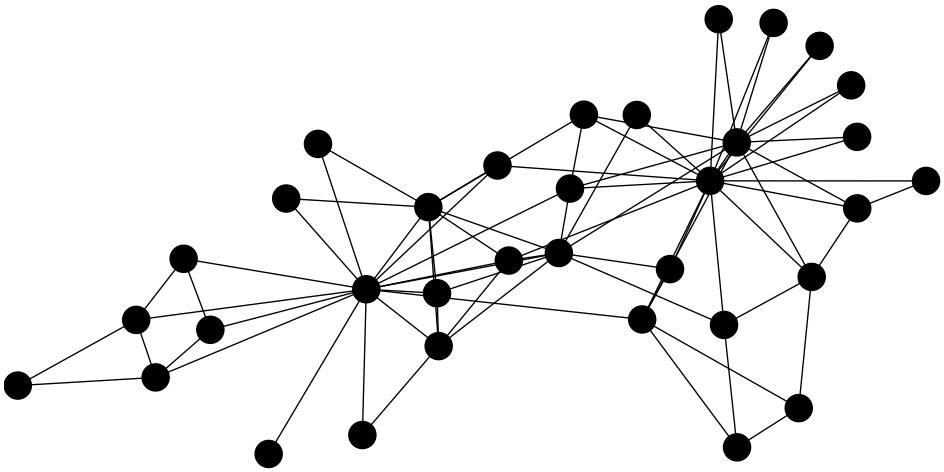


Figure 1.1: Visualization the Zachary Karate Club Network [97]. The dots represents people, and the connections represent interaction between people. Two nodes are connected if they interact with each other.

There are a number of tools and techniques available for analysis of graphs. Representing real-world structures by graphs allows us to better understand and analyze them. For example, if we want to find an important person in a social network, we can use the concept of *node centrality* (Section 2.2) on the graph. If we are interested in understanding the higher level organization of the world wide web – such as the pattern of connections between group of websites that frequently link to each other and to others that rarely link to each other – we can study the *dense substructures* (Section 2.3) of the graphs.

Frequently, we may need to analyze only a part of the entire graph – in such case we consider the relevant subgraph. A *subgraph* of a graph G, is another graph $G' = \langle V', E' \rangle$, such that (1) $V' \subseteq V$ and (2) $E' \subseteq E$. In other words, a subgraph of G is the collection of V', a subset of the nodes in G, and E', some or all the connections between nodes in V' in G. If E' is the set of all the connections between V' that exist in G, we call that subgraph the *induced subgraph* of V'.

Although a subgraph can be induced from any subset of nodes, certain types of subgraphs are of

special interest, because they give us insight into the organization of the network. Some common ones includes communities [41], k-cores [88], k-truss [29], k-peak [44] etc.

In this work, we focus our attention on k-cores. A k-core is defined as*the maximal subgraph such that all nodes in the subgraph have at least* k *connections to other nodes in the subgraph* (Section 2.3.2). By changing the value of k, we can get subgraphs of different importance/centrality. We refer to the hierarchical organization of the k-cores for different values of k as the k-core structure. In general for higher values of k, the resulting subgraph is considered more central/important. The largest value of k such that a node belongs to that k-core is called the core number of that node. As an example, k-cores have been used in many important applications, such as identifying important proteins in protein-protein interaction networks, identifying anomalies (bots) in social networks, speed up community detection, study ecology collapse etc. Refer to Section 2.3.2 for a more thorough review about k-cores.

Because k-cores are used in so many applications, it is important to understand their resilience to errors or changes in the graph. Collection of data is not always perfect – some edges or nodes might have been missed. In such cases, how much does this missing data affect the detected k-core structure? In some networks the missing data can drastically alter the output of an analysis that uses the k-core structure; while it is relatively unaffected in some others. So, understanding this resilience can help us gain better insight into data being analyzed.

In this dissertation we study three different type of changes to the k-cores structure of a graph: (1) Core Structural Change (Chapter 4), (2) Core Minimization (Chapter 5), and (3) Collapsed k-Core (Chapter 6).

Core Structural Changes (Chapter 4): To study a network, we first need to collect the graph data – which nodes are present, and how are they connected to each other? The collected data may be imperfect, and in many cases, there may be missing data. In the chapter on core structural changes, we consider the change in the relative ordering of nodes based on their core number core numbers

when edges (or nodes) are missing. The core numbers of nodes are a form of node centrality – nodes with higher core number can be considered as more important than those with lower values. This is important in a lot of applications such as finding anomalies [89], finding influential spreaders [3, 50, 59] etc.

In Chapter 4, we study this type of change and propose a measure to quantify the resilience of a graph to such changes. We also propose measures that are fast to compute and can serve as proxies for this measure. Finally we propose an algorithm (`MRKC`) to add edges to a graph to maximize this resilience [53].

Core Minimization (Chapter 5): We also come across applications where the ordering of the nodes is not important – rather, what is important is the size of the k-core. Such applications includes the study of ecology collapse [71], jamming transitions [70], etc. Zhu *et al.* studied this problem as the *core minimization* – which are the b edges/nodes such that if deleted, the size of the k-core is minimized?

In Chapter 5, we study the resilience of graphs to core minimization – specifically, given a graph how can we characterize the extent to which it can be affected by core minimization if there are missing data or targeted attacks based on previous works. We propose a measure that is efficient to compute and, motivated by this measure, propose a novel algorithm (`CIM`) to maximize the resilience of a graph to core minimization.

Graph Unraveling (Chapter 6: In this chapter, we do not deal with missing data but rather, the natural tendency of the nodes in some type of networks to remain in the network only if they have sufficient number of neighbors. In many networks users stay active due to their neighbors. For example, a social network users remain active only if they have k active friends – otherwise they will become inactive themselves. This might trigger a cascading collapse of the network until only the k-core is left. So, the goal is to select a fixed number of anchor nodes (i.e., nodes that can be incentivized to remain in the network) to maximize the size of the k-core [14, 15, 98].

Type of Change	Affected	Cause
Core Structural Change	Order of nodes	Missing data
Core Minimization	Size of k-core	Missing data or targeted attacks
Graph Unraveling	Size of graph	Cascading collapse

Table 1.1: Possible changes to a network's k-core structure, their effects, and causes.

In Chapter 6, we study this problem of finding anchor nodes to maximize the size of the anchored k-core. We propose a novel algorithm (`RCM`) that considers not only the immediate effect of an anchor node, but also the effect on subsequent anchor node selection.

Skeletal Core Graph (Chapter 7): Finally in Chapter 7, to investigate the graph structures that lead to different behavior in response to these type of changes, we propose the idea of the *skeletal core graph*. We define a skeletal core of a graph as a minimal subgraph that preserves nodes' core numbers (i.e., removing even one edge will result in at least one node changing core number). Based on the connections between the different k-shells,[1] we describe two skeletal core graphs: the decentralized skeletal core and the centralized skeletal core graph. We then show how these relate the resilience of the k-core structure of graphs.

The major contribution of this dissertation can be summarized as follows:

1. We study the resilience of the k-core structure of graphs to different type of changes and how we can measure and characterize them.

2. For type of change, we propose novel algorithms that tell us what small modification we can make to the network to improve its resilience.

3. Finally, we propose a type of graph called the skeletal core graph to explain the behavior of different graphs to these different type of changes.

[1] The k-shell is the subgraph induced by all the nodes that have a core number of k.

Chapter 2

Background

In this chapter we present the background required for the rest of this dissertation. We will present a detailed discussion about graphs, centrality measures on graphs, dense substructures on graphs and, finally, k-cores. We start with an introduction to graphs. Additional chapter-specific background is provided in the appropriate chapters.

2.1 Graphs

A *graph* G is an ordered pair of disjoint sets $\langle V, E \rangle$ such that $E \subseteq V^2$ [18]. The elements in V are called the *vertices* or *nodes*; and those in E are called *edges* or *links*. If there is an edge $(u, v) \in E$, we say that the nodes u and v are connected (or adjacent, or neighbors) – in this case nodes u and v are the endpoints of the edge (u, v). In some applications, we might allow for edges where both the endpoints are the same nodes. This is called a self-loop; and in this work we assume that no graph have self-loops. When we think about graphs, it is helpful to visualize it with circles (or dots) as nodes and lines connecting them as edges.

A graph can be used to model a number of real-world systems. For example, consider the world wide web (WWW). The web pages can be represented by nodes and hyperlinks can be represented as edges. Another example is a social network where nodes are people and edges represent friendship.

Similarly, various other real-world structures from diverse domains can be modeled by graphs. Due to this the study and understanding of various properties of graphs is extremely important.

In some graphs, the direction of the edges is not important; that is $(u, v) = (v, u)$. An example of such a graph is a friendship graph. These are called *undirected* graphs. Contrast that with the example of the WWW. In this case, the direction of the hyperlinks is important; that is $(u, v) \neq (v, u)$. This is called a *directed* graph. In this work, we assume that all the graphs are undirected.

A graph $G' = \langle V', E' \rangle$ is said to be a subgraph of $G = \langle V, E \rangle$ if $V' \subseteq V$ and $E' \subseteq E$. A subgraph G' that contains all the edges that connects all the edges that connects the nodes in V' in G is called an induced subgraph; that is $E' = (V')^2 \cap E$. There are cases where we are interested in studying only a part of the graph – for example we might be interested in studying only the friendship network of a certain age-group. Then, we can study the induced sub-graph of the nodes that we are interested in. In other cases, we may need to study a subgraph because we do not have access to the entire graph or the entire graph is too large. In such cases, we need to keep in mind the properties that we want to study and select appropriate methods for generating the subgraph [9, 52, 55, 62, 63, 64, 95, 93].

In a graph the *neighbors* of a node $v \in V$ is the set of all the other nodes that have a edge to v. We denote it denote by $\Gamma_G(v)$ and formally,

$$\Gamma_G(v) = \{u \in V : (v, u) \in E\} .$$

If the graph G is clear from the context, we can drop the subscript G. The number of neighbors of a node is called the *degree* of the node. That is,

$$d(u) = |\Gamma(u)|. \tag{2.1}$$

Graphs can also be represented in matrix form. Assuming that $|V| = n$, the *adjacency matrix*, $\mathbf{A}$, of

G is an $n \times n$ matrix such that,

$$\mathbf{A}_{u,v} = \begin{cases} 1 & \text{if } (u.v) \in E. \\ 0 & \text{otherwise.} \end{cases} \tag{2.2}$$

Consider the example of the world wide web (WWW). When a user browses the WWW, she starts from some webpage v_0 and clicks on a link to get to another v_1, and clicks on another link on v_1 to get to v_2, etc. This is called a walk on the graph. A *walk* on a graph is defined as a sequence of edges that connects as sequence of nodes starting from v_0 and ending at v_x. It is not necessary for the starting and ending node to be distinct, and the vertices that a walk passes through might also be repeated. If we restrict a walk to only pass through each node once, we get a *simple path*. In many graphs, the *shortest path* between pairs of nodes is an important property, and it is the minimum number of nodes in a simple path from nodes v_0 to v_x.

In some graphs it might not be possible to reach every node with a walk starting from some other node. So, the idea of connected component is important here. A *connected component* is a subgraph such that every node in the subgraph can be reached by a walk from any other node in the subgraph. We say that a graph (or subgraph) is *disconnected* if it has more than one connected components.

2.2 Centrality Measures

As described in the introduction, an important concept when we use graph for analysis of a real-world network is the concept of centrality – which are the important/central nodes? Various measures have been proposed that capture different ideas of importance – there are some that captures importance with the idea of popularity, some consider flow in an network, and others looks at paths in the networks. In this section, we discuss some of the important centrality measures.

Degree Centrality: The degree centrality of a node v is the fraction of nodes that are connected to v. For $v \in V$, the normalized degree centrality is given by $\frac{|\Gamma(v)|}{|V|-1}$. In many social networks, the degree centrality is a a good measure of importance because important people generally have more connections.

Eigenvector Centrality: In many cases, only the number of connections is not a good indicator of importance. For example, in the WWW a webpage v may be connected to a lot of other web pages, but if those are not important, v is likely not as important as another u that has fewer, but more important, connections.. This is the motivation of the *eigenvector centrality* [73], and the eigenvector centrality of node v defined as the v-th entry of the vector $\mathbf{x}$ such that $\mathbf{A} \cdot \mathbf{x} = \lambda \cdot \mathbf{x}$.

Betwenness Centrality: In a communication network, the shortest paths are very important. This is the idea behind *betweenness centrality*, which is defined as the sum of the fraction of shortest paths that pass through the node [37]. Because calculating the betweenness centrality requires computation of shortest path between all pairs of nodes, it is very expensive for large graphs. However, approximate methods have been proposed [21].

Closeness Centrality: Another centrality measure that makes use of the shortest paths is *Closeness Centrality*. The closeness centrality of a node is defined as the reciprocal of the average shortest path length to all nodes that can be reached by a walk [13]. Like in the case of betweeness centrality, computation of closeness centrality is also very expensive on large graphs.

2.3 Dense Substructures

There are many ways to describe hierarchical structures in graphs, such as k-core [88], k-truss [30], k-peak [44], communities [74] etc. In this section, we will provide some background on some of these structures.

2.3.1 Community

One of the most commonly used dense substructure in graphs are communities. Communities have many definitions, but in general, a good community is one that has more internal connections than expected. In [74] Newman & Girvan proposed 'modularity' as a measure of the strength of community structure, and algorithm based on modularity maximization have become some of the most popular techniques for community detection.

Various community detection methods based on modularity maximization has been proposed [28, 72, 16, 77]. However, one of the most popular is the *'Louvain modularity maximization'* proposed by Blondel *et al.*. This is a greedy algorithm to find communities in a network by grouping nodes in such a way as to maximize the modularity. Community detection methods based on modularity suffers from the resolution limit [56]; nonetheless they remain popular for their effectiveness and efficiency. Other methods of community detection includes random walk based methods [17, 84], statistical inference [78, 47] etc.

2.3.2 k-Core

The k-*core* of a graph, $G = \langle V, E \rangle$, is the maximal subgraph such that every node in the subgraph has at least k neighbors in the subgraph [88, 66]. If a node belongs to the k-core but not in the $(k+1)$-core, we say that the *coreness* (or *core nummber*) of the nodes in k. We will denote this by, $\kappa(u, G)$ for $u \in V$. The coreness of a node can also be considered as a centrality measure.

The subgraph induced by the by the nodes with coreness of k is called the k-*shell*. Note that the k-core and k-shell need not be connected. In a graph the largest value of k such that the k-core is not empty is called the *degeneracy* of the graph, and the associated core is called the *degeneracy core*.

Figure 2.1 shows a toy example demonstrating the different k-cores and k-shells. We can see that

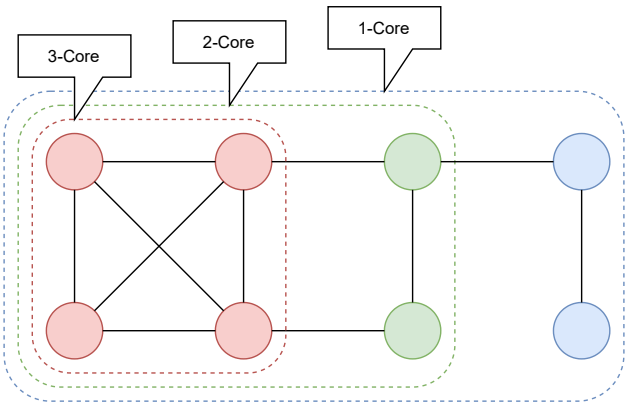


*Figure 2.1: An example graph showing the **k**-cores and the **k**-shells. The subgraph induced by the red nodes is the 3-core, the one induced by the red and green nodes is the 2-core, and the entire graph is a 1-core. The red, green and blue nodes by themselves induces the 3, 2, and 1-shells.*

all the red nodes have 3 other red nodes as neighbors. So, they form the 3-core. Similarly, the green nodes have 2 neighbors that are either red or green; and thus form the 2-shell. Together with the red nodes, they form the 2-core Finally, the entire graph forms a 1-core. The red, green and blue nodes by themselves induces the 3, 2, and 1-shells.

The process of finding all the *k*-cores in a graph is called *k-core decomposition*. There is an efficient algorithm for performing a k-core decomposition [10]. The algorithm works by iteratively removing nodes that have less than *k* neighbors and stopping when there are no more nodes to remove. The running time of this algorithm scales linearly with the number of edges.

Chapter 3

Related Works

In this chapter, we describe previous literature related to k-core and related areas.

3.1 k-Core Decomposition

Erdos and Hajnal [34] described the first k-core related concept in 1966, defining the degeneracy of the graph as the maximum core number of a vertex in the graph. Matula introduced the min-max theorem [67] for the same concept, but in the context of graph coloring. Roughly simultaneously, Seidman [88] and Matula and Beck [66] defined the k-core subgraph as the maximal connected subgraph where each vertex has at least degree k.

Seidman stated that k-cores are good *seedbeds* that can be used to find further dense substructures, but did not provide a principled algorithm for finding k-cores [88]. Matula and Beck [66], on the other hand, give algorithms for finding the core numbers of vertices, and for finding all the k-cores of a graph (and their hierarchy) by using these core numbers, since there can be multiple k-cores for the same k value.

Batagelj and Zaversnik introduced an efficient implementation that uses the bucket data structure to find the core numbers of vertices [10]. In contrast to previous work [88, 66], they defined the k-core as a possibly disconnected subgraph.

The k-core decomposition has been used in numerous applications, including network visualization [5, 99, 102], studying the topology of large networks (such as the Internet) [8, 24], accelerating community detection [79], and studying the resilience of communities [38].

k-cores have been used for applications in a variety of scientific fields. Altaf-Ul-Amine *et al.* proposed a method for predicting the functions of proteins based on the k-core decomposition of the protein-protein interaction network [4]. In [70], Monroe *et al.* explained jamming transitions (when particles are packed such that movement is not possible) by the emergence of the k-core in the particle contact network. In [71], the authors used the k-core to predict the structural collapse of ecosystems.

Thanks to the practical benefit and linear complexity of the k-core decomposition, there has been a great deal of recent work in adapting k-core algorithms for different data types or setups. Cheng *et al.* [25] introduced the first external-memory algorithm, and Wen *et al.* [94] and Khaouid *et al.* [49] provided further improvements in this direction.

To handle the dynamic nature of the real-world data, Sariyuce *et al.* [85] introduced the first streaming algorithms to maintain the k-core decomposition of a graph upon edge insertions and removals. Lie *et al.* [58], Zhang *et al.* [101], and Esfandiari *et al.* [35] have also introduced methods to maintain k-core structure in the case of streaming data.

Motivated by the incomplete and uncertain nature of the real network data, O'Brien and Sullivan [76] proposed new methods to locally estimate core numbers (K values) of vertices when the entire graph is not known, and Bonchi *et al.* [20] showed how to efficiently perform the k-core decomposition on uncertain graphs, which has existence probabilities on the edges.

There has been a lot of works on extending the notion of k-cores to other network settings. Sariyuce *et al.* generalized k-cores to higher order structures [86], and Giatsidis *et al.* adapted the idea of k-cores to directed and weighted graphs [39, 40].

3.2 Applications of k-Core

In [51], the authors studied an extremist web forum – the nodes are users and message threads; and there is an edge between a user and message thread if the user posted in that thread. They found that the users who has high core number ($k = 8$ in their dataset) are the influential advisors within the extremist network.

k-cores can be used to find bots in social networks. In [89], Shin *et al.* developed a method to find anomalous nodes (bots) in a social network based on their core number and degree. They found that, in general, nodes in a social network follows a 'mirror pattern' – the core number of a node is strongly correlated with its degree. They found that anomalous nodes deviates from this pattern and proposed a method to measure the deviation from the mirror pattern to detect anomalies.

k-core has also been used for anomaly detection in transcriptional regulatory networks [91], behavior of customers in online banks [90], online user generated contents [19], internet routing [69], wallets on crypto-currency platform [80] etc.

The study of influential spreaders in social networks is another area where k-cores have been applied with promising results. In [3], the authors proposed a method of assigning edge weights in an online social network. They, then, proposed a weighted k-core and showed that this method can capture influential spreaders more effectively than other measures like PageRank, degree centrality etc.

In [82], the authors evaluated the effectiveness of different types of centrality measures in finding the different types of influential spreaders using the SIR model [48]. They found that the influential spreaders identified by k-core are the ones that can reach the furthest distance in the graph the fastest. In addition to these works, there has also been a plethora of works that uses k-core and variants to identify influential spreaders [33, 43, 59, 61, 89, 96].

Other applications of k-core includes network visualization [7, 23, 32, 75, 100], studying the topology

of large networks (such as the Internet) [6, 24], accelerating community detection [79] etc.

3.3 Changes to k-Core Structure

Data collection is not always perfect – sometime there are missing edges or nodes. It is even possible that there are missing data due to attacks. In this section, we present some of the recent works on the different type of changes to the k-core structure due to missing data.

3.3.1 Core Structural Change

There are only a few works that study the sensitivity of the order of nodes based on their core number. Most closely related to our work is the study by Adiga and Vullikanti, investigating the robustness of the top cores under sampling and in noisy networks [1]. They reported that the success in recovering the top cores under sampling and noise exhibits non-monotonic behavior with the amount of samples and noise.

In [53], we proposed a measure for the resilience of the k-core structure and a method of inserting edges to improve the resilience. In our work, we follow a more general approach and quantify the resilience of the core numbers, and the impact of the neighbor vertices on the stability. In addition, we propose edge insertion heuristics to strengthen the core numbers while preserving the existing core decomposition.

3.3.2 Core Minimization

There have been a few recent works that core minimization, but most of them focus on finding nodes/edges to remove that minimizes the k-core the most. Zhang *et al.* [99] studied this problem with the objective of finding the critical users. They defined the critical users as those whose removal from the network will lead to the size of the k-core being minimized. They proposed a

greedy algorithm called `CKC` to efficiently find a given number of critical users.

Zhu *et al.* [104] also studied the problem of k-core minimization. What differentiates their work from the previous one was that they were focused on finding the critical edges – that is, the edges whose removal leads to the minimization of the k-core. Medya *et al.* [68] also worked on the problem of finding the critical nodes that leads to core minimization but approached it from a game-theoretic perspective.

Schmidt *et al.* [87] studied the problem of finding the minimal set of nodes whose removal destroys the k-core. They relate it to the problem of finding the minimal contagious set [31, 81, 45, 36]. They provided a upper bound on the size of the minimal contagious set, and provided a heuristic based approach to finding it.

3.3.3 Graph Unraveling

The cascading collapse of a graph due to users, with not enough engagement leaving, was first described by Bhawalkar *et al.* [14] as the *anchored k-core* problem. The problem was inspired by the observation that a user in a social network is motivated to stay only if her neighborhood meets some minimal level of engagement: in k-core terms, she will stay if k friends are also in the network. Bhawalkar *et al.* defined the anchored k-core as the subgraph that is computed using the usual k-core decomposition algorithm, but with the modification that selected 'anchor' nodes are not deleted during the process. These anchored nodes may represent, for example, nodes that are recruited to remain active in the network, even if their friends are inactive. The anchored k-core problem, then, is the problem of selecting a specified number b anchor nodes such that the number of nodes in the anchored k-core is maximized.

Bhawalkar *et al.* showed that for a general graph the anchored k-core problem is solvable in polynomial time for $k \leq 2$, but is NP-hard for $k > 2$ [15]. They also showed that the problem is W[2]-hard with respect to the number of anchors and Chitnis *et al.* showed that the problem is W[1]-hard with

respect to the number of nodes in the anchored k-core [26].

Zhang *et al.* proposed a greedy algorithm, called `OLAK`, for the anchored k-core problem [98]. `OLAK` operates over b_{max} iterations, where b_{max} is the allowable number of anchor nodes. In each iteration, a node that is not in the anchored k-core but which would generate the largest number of followers if anchored is selected as the next anchor. Because only a single anchor node at a time is considered, and only nodes from the $(k-1)$-shell[1] can become followers when anchoring a single node, `OLAK` considers only follower nodes from the anchored $(k-1)$-shell during each iteration.

Zhou *et al.* [103] studied a problem that is close to the anchored k-core problem – which edges should be added to maximize the size of the k-core. However, this is fundamentally different from the anchored k-core problem because the graph cannot be modified in the anchored k-core problem.

[1] The k-shell is the subgraph of the k-core $\setminus$ $(k-1)$-core.

Chapter 4

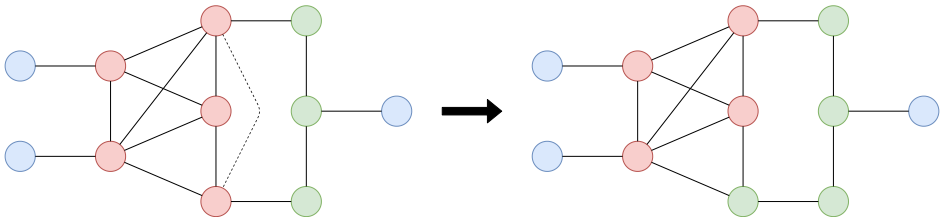
Core Structural Changes

In many network applications, we may encounter the problem of missing edges or nodes. For example, in technological networks, edges may be lost due to dropped communication links, and in router networks, nodes might drop due to routers being turned off. Or, in the case of social networks, edges or nodes might be missing during the data collection process.

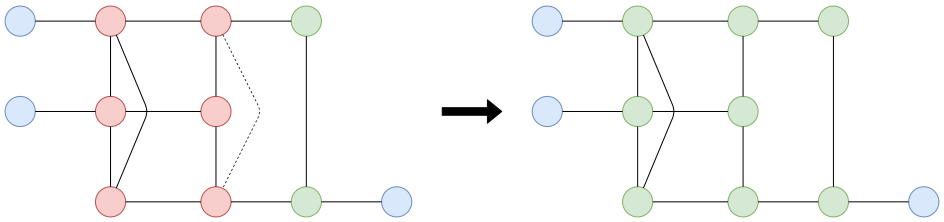
It is thus valuable to understand the resilience of the k-core of the network to missing edges and nodes. In this section, we introduce the concept of *core resilience*, which quantifies the degree to which a network's core structure changes when nodes or edges are missing. In this case, we consider only the case when nodes and/or edges are missing uniformly at random.

The first step in understanding how networks' core structure change due to missing edges and nodes is to define a metric to measure this. To this end, we propose the *Core Resilience*.

To demonstrate, consider the graph shown in Figure 4.1a. The red, green and blue nodes have coreness of 3, 2 and 1 respectively. If the dashed line is deleted, only one node changes coreness from 3 to 2. Contrast this with the graph in Figure 4.1b. In this case, if the dashed edge (or rather any edge between red nodes) is removed, all the red nodes changes coreness from 3 to 2. So, the second graph has a lower core resilience. This example also shows how inaccurate a study that uses k-core can be if the original graph has very low core resilience.



(a) Example of case where k-core structure does not change a lot.



(b) Example of case where k-core structure changes a lot.

Figure 4.1: Toy examples showing a case where k-core structure does not change much (Figure 4.1a), and one where it changes a lot (Figure 4.1b).

4.1 Core Resilience

Formally, *we define the (r, p)-core resilience of a network G as the rank correlation between the top $r\%$ nodes (as ranked by core number) in the original network to that of the network after $p\%$ of the the edges or nodes have been removed uniformly at random.* We denote the (r, p)-core resilience of a graph G to edge deletion by $\mathcal{R}_r^{e(p)}(G)$, and that due to node deletion by $\mathcal{R}_r^{n(p)}(G)$. We will use $\mathcal{R}_r^{\cdot(p)}$ to refer to (r, p)-core resilience in general. The intuition behind the core resilience is that the ranking of the nodes by their core number reflects the k-core hierarchy – nodes that are ranked higher (in descending order) are higher up in the hierarchy. So, a measure of rank correlation with and without the missing data can measure the change in the k-core hierarchy.

Let $G = \langle V, E \rangle$ be a network, and let G^p represent the network obtained removing $p\%$ of the edges (or nodes) from G randomly. Let the top $r\%$ nodes (by core numbers) in G be denoted by V_r. Define a set M_r^p such that,

$$M_r^p := \{(\kappa(u, G), \kappa(u, G^p)) : u \in V_r\}, \tag{4.1}$$

where $\kappa(u, G)$ is the core number of node u in network G[1].

Then, the (r, p)-core resilience of G is given by,

$$\mathcal{R}_r^{\cdot(p)}(G) \coloneqq \tau_b\left(M_r^p\right) \tag{4.2}$$

where $\tau_b(\cdot)$ is the modified Kendall's tau-b rank correlation[2]. The definition is not tied the Kendall's tau-b rank correlation. We can replace $\tau_b(\cdot)$ by any other measures of rank correlation.

While $\mathcal{R}_r^{\cdot(p)}$ gives rich, detailed insight into the core resilience of the different cores of the network at different levels of edges or nodes deletion, in some applications it may be preferable to use a simpler measure. We thus define an aggregate measure, the (r, p_l, p_u)-*core resilience*. *We define the* (r, p_l, p_u)-*core resilience of a network as the mean* (r, p)-*core resilience as we vary* p *from* p_l *to* p_u. We denote the (r, p_l, p_u)-core resilience of G by $\mathcal{R}_r^{\cdot(p_l,p_u)}(G)$.

$$\mathcal{R}_r^{\cdot(p_l,p_u)}(G) \coloneqq \frac{\int_{p_l}^{p_u} \mathcal{R}_r^{\cdot(x)}(G)\,dx}{p_u - p_l} \tag{4.3}$$

In practice, we approximate the integral in Equation 4.3 by a summation with step size 1.

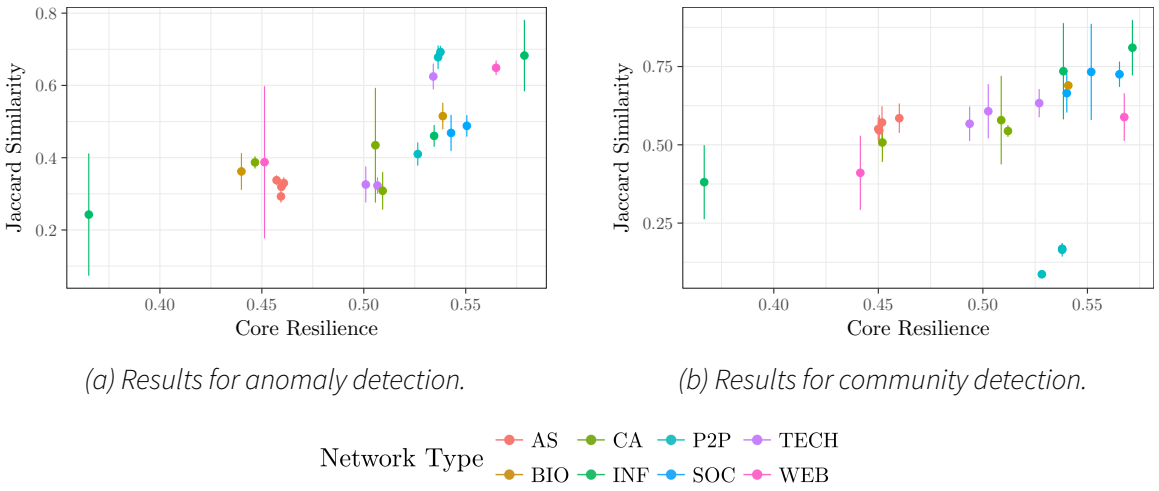
It should be noted that there are a number of graph robustness measures, but the concept of core resilience specifically concerns the k-core structure of the network, and so is not directly related to these existing measures. To verify this we compared the *Natural Connectivity* [46] to the Core Resilience of various real-world networks, and did not observe any significant correlation.

4.2 Motivating Applications

The concept of Core Resilience is helpful in applications where the k-core structure of the network under missing edges or nodes is important. In this section we will discuss two such applications: (1)

[1] If a node has been deleted, its core number in G^p is 0.

[2] We make the modification to count ties as concordant pairs.



(a) Results for anomaly detection.

(b) Results for community detection.

Figure 4.2: Similarity between anomalies (Figure 4.2a) and communities (Figure 4.2b) found in the full network G and the sample G' for different real-world networks. The x-axis is the Core Resilience ($\mathcal{R}_{50}^{n(0,50)}(G)$) of the different networks against node deletion, and the y-axis is the Jaccard Similarity. As expected, in the networks with high Core Resilience, the results on the sample is more similar to that on the full network in general.

anomaly detection, and (2) community detection.

Assume that we have a network $G = \langle V, E \rangle$ and a subgraph $G' = \langle V', E' \rangle$, where G' is the result of random walk on G.

If we perform anomaly detection [89] or community detection [79] on G', how well do the results on G' reflect the true anomalies and communities in G? Because these applications make use of the k-core structures, we expect the results to more closely match that of the original graph if the original graph has high core resilience.

We verify this experimentally on multiple real-world networks, and the sample we use is generated by a random walk with half the number of nodes in the network as the budget. (The dataset we use are given in Table A.1.)

4.2.1 Anomaly Detection

In this application, we perform anomaly detection on the full network G using the *CORE-A* method proposed in [89] to find the anomalous nodes V_α. This method operates on the intuition that nodes with high core numbers also have high degrees. So for a given node, the difference between the ranking in terms of the degree and core number (referred to as *dmp* in [89]) should be fairly small. However, anomalous nodes (for example, someone in a social network who paid to get more followers) deviate significantly from this pattern. By looking at the *dmp* values of the nodes, the anomalies are identified in the CORE-A algorithm.

We find anomalies in the subgraph G' with the same method, and refer to the set of these anomalies as V'_α. We then use Jaccard Similarity to determine how close the result on G' is to that on G.

$$J_\alpha(V_\alpha, V'_\alpha) = \frac{|V_\alpha \cap V'_\alpha|}{|(V_\alpha \cap V') \cup V'_\alpha|}$$

We present results in Figure 4.2a. We can observe that the anomalies found in the sample are more similar to those in the full network for networks with high core resilience.

4.2.2 Community Detection

By finding a central region of the network, k-cores can be used to accelerate community detection. We perform community detection using the method proposed in [79] and the Louvain method on the original network G. We denote the communities in G by C. Then, we perform community detection with the same method on G', to get the communities C'.

We compute the similarity between C and C' as the mean Jaccard Similarity between the commu-

nities in C' to its best match community in C.

$$J_c(C, C') = \frac{1}{|C'|} \sum_{c \in C'} \frac{|c \cap \beta(c, C)|}{|c \cup (\beta(c, C) \cap V')|}$$

where $\beta(x, Y)$ is a function that maps the community x to another community y such that $|x \cap y|$, and there are no other $x' \in X$ that maps to y.

Figure 4.2b shows the results of these experiments on community detection. In the networks with higher Core Resilience, the nodes that are grouped together in the same community in the sample are more frequently grouped together in the original communities as well. The only exceptions to this are two P2P networks, for which the similarity is low even though they have relatively high core resilience. This is because there are very few communities in the original network, but only a single, giant community. So, $\beta(c, C) = \emptyset$ for most $c \in C'$.

These two applications demonstrate that if we know the Core Resilience of a network, we can use it as an indicator of how much we should expect core-based observations on incomplete data to reflect those on the original.

4.3 Characterizing Core Resilience with Node Level Properties

Computing the (r, p)-core resilience of a network requires repeated computation of the k-core. Because the time complexity of the k-core decomposition algorithm is $\mathcal{O}(|E|)$, it may not be practical to compute the (r, p)-core resilience in larger graphs. It is thus valuable to characterize the core resilience of the network without directly computing the (r, p)-core resilience (and, as we will see, this characterization allows us to develop an effective algorithm for improving a network's core resilience).

In this section, we propose two node properties based on a network's structure: (1) *Core Strength*, and (2) *Core Influence*. The core strength of a node is a measure of how likely its core number will

decrease when edges are deleted from the network. The core influence of a node is a measure of the extent to which nodes with lower core numbers depend on that node for their own core numbers. In Sections 4.3.3 and 4.3.2, we describe the core influence and core strength properties in more details.

We also define an overall network property, based on the core strength and core influence of the nodes in the network. We describe this in more detail in Section 4.3.4. We perform experiments on real world networks of various types to show the relationship between these measures and the core resilience of the network.

4.3.1 Notations

Before describing the Core Influence and Core Strength properties, we first introduce some notations. We split the neighbors of $u \in V$ into three sets: (1) $\Delta_<(u, G)$, (2) $\Delta_=(u, G)$, and (3) $\Delta_>(u, G)$ representing, respectively the neighbors of u with core number less than, equal to, and greater than that of u.

$$\Delta_<(u, G) = \{v \in \Gamma(u) : \kappa(v) < \kappa(u)\} \tag{4.4}$$

$$\Delta_=(u, G) = \{v \in \Gamma(u) : \kappa(v) = \kappa(u)\} \tag{4.5}$$

$$\Delta_>(u, G) = \{v \in \Gamma(u) : \kappa(v) > \kappa(u)\} \tag{4.6}$$

$$\Delta_\geq(u, G) = \Delta_=(u, G) \cup \Delta_>(u, G) \tag{4.7}$$

We also define a set V_δ of nodes where each node $u \in V_\delta$ has at least one neighbor node, v, with a larger core number, i.e., $K(u, G) < K(v, G)$. That also means the following:

$$V_\delta = \{u \in V : |\Delta_=(u, G)| < \kappa(u, G)\}. \tag{4.8}$$

4.3.2 Core Strength

The Core Strength of node u is the minimum number of u's neighbors that need to drop to a lower core for u to also drop to a lower core.

We denote the core strength of u in G by $CS(u, G)$. The assumption that that the coreness of the neighbors does not change is not necessarily required, but it makes computation of the core strength very fast.

For all nodes u in network G, u gets its core number due to connections to $\Delta_{\geq}(u, G)$. Thus, the core strength of node $u \in G$ is given by,

$$CS(u, G) = |\Delta_{\geq}(u, G)| - \kappa(u, G) + 1. \tag{4.9}$$

Intuitively, the core strength of a node u describes how likely it is to retain its core number when it loses connections. A node with a high core strength has many redundant connections (i.e., many connections to other nodes with equal or higher core number), and so is less likely to drop its core number if its connections are deleted.

Algorithm 1 The algorithm to calculate the core strengths of all the nodes.

```
1: function CalculateCoreStrength(G = ⟨V, E⟩)
2:     κ ← CalculateCoreNumber(G)
3:     CS ← {}
4:     for u ∈ V do
5:         CS[u] ← |{v ∈ Γ(u) : κ(v) ≥ κ(u)}| − κ(u) + 1
6:     end for
7:     return CS
8: end function
```

Theorem 4.1 (Complexity of Algorithm 1)**.** *The time complexity of Algorithm 1 is $\mathcal{O}(|E|)$; and the space complexity is $\mathcal{O}(|V|)$.*

Proof. Given a network $G = \langle V, E \rangle$, computing the core strength of all the nodes is possible once

the k-core decomposition is performed, which takes $O(|E|)$ time. For each node we need to count the number of neighbors with greater or equal core number, which is also linear in the number of edges, $O(|E|)$. So, the time complexity of computing the core strength of all nodes is $\mathcal{O}(|E|)$.

In Algorithm 1, the only additional space required is to store the core strengths of all the nodes. So, the space complexity is $\mathcal{O}(|V|)$. □

4.3.3 Core Influence

The Core Influence of a node u in network G is a measure of the extent to which u affects the core numbers of neighbor nodes with lower core numbers. We denote it with $CI(u, G)$.

For a node u, the set of nodes that 'immediately' depend on u for their core numbers is $\Delta_{\leq}(u, G)$, i.e. the neighbors of similar or lower coreness. If there is an edge (u, v) such that $\kappa(u, G) = \kappa(v, G)$, both u and v influences each other for their coreness.

In order to compute core influence, the first step is to create a matrix M of size $|V| \times |V|$ such that,

$$M_{u,v} = \begin{cases} 1 & \text{if } u = v \\ \frac{\kappa(u,G)}{|\Delta_{\geq}(u,G)|} & \text{else if } (u, v) \in E \wedge \kappa(u, G) \leq \kappa(v, G) \\ 0 & \text{otherwise} \end{cases} . \tag{4.10}$$

Let $\mathbf{r}$ be the eigenvector of the matrix M. Then, the core importance of node u is r_u.

Theorem 4.2 (Complexity of Algorithm 2)**.** *The time complexity of Algorithm 2 is $\mathcal{O}(|E|)$; the space complexity is $\mathcal{O}(|E|)$.*

Proof. To compute the approximate core influence of all nodes in $G = \langle V, E \rangle$, we need to perform k-core decomposition first ($O(|E|)$). The matrix $\mathbf{M}$ can be created in $\mathbf{O}(|E|)$. With the power method, the eigenvector can be calculated in $\mathcal{O}(|V|)$. So, the overall computation takes $O(|E|)$.

Algorithm 2 The algorithm to calculate the core influence of all the nodes.

```
1:  function CalculateCoreInfluence(G = ⟨V, E⟩)
2:      κ ← CalculateCoreNumber(G)
3:      M ← 0_{|V|×|V|}
4:      for (u, v) ∈ E do
5:          if κ(u) ≥ κ(v) then
6:              x ← |{w ∈ Γ(v) : κ(w) ≥ κ(v)}|
7:              M_{u,v} ← 1/x
8:          end if
9:      end for
10:     r ← EigenVector(M)
11:     CI ← {}
12:     for u ∈ V do
13:         CI[u] ← r_u
14:     end for
15:     return CI
16: end function
```

In Algorithm 2, the space needed to store the matrix M is $\mathcal{O}(|E|)$ assuming we store it as a sparse matrix. So,t he space complexity is $\mathcal{O}(|E|)$. □

Approximate Core Influence: In many applications we found that the influence of a node to other nodes of higher coreness is more importance. So we can discard the contributions from the nodes of same coreness to approximate the core influence. In that case, for (u, v) if $\kappa(u, G) = \kappa(v, G)$, we set $M_{u,v} = 0$. In this case, we can guarantee convergence in one step.

4.3.4 Core Influence-Strength

Core Strength and Core Influence describe node level properties. To characterize the network, we need an aggregate measure that describes the network level property.

Assume that $CI_f(G)$ is the f percentile of core influence of all nodes in G. Let $S_f(G)$ be the set of nodes in G with core influence equal to or greater than $CI_f(G)$.

$$S_f(G) = \{u \in V : CI(u, G) \geq CI_f(G)\} \tag{4.11}$$

Then we define the *Core Influence-Strength* as the mean core strength of $S_f(G)$. We denote it by $CIS_f(G)$,

$$CIS_f(G) = \frac{\sum_{u \in S_f(G)} CS(u, G)}{|S_f(G)|}. \tag{4.12}$$

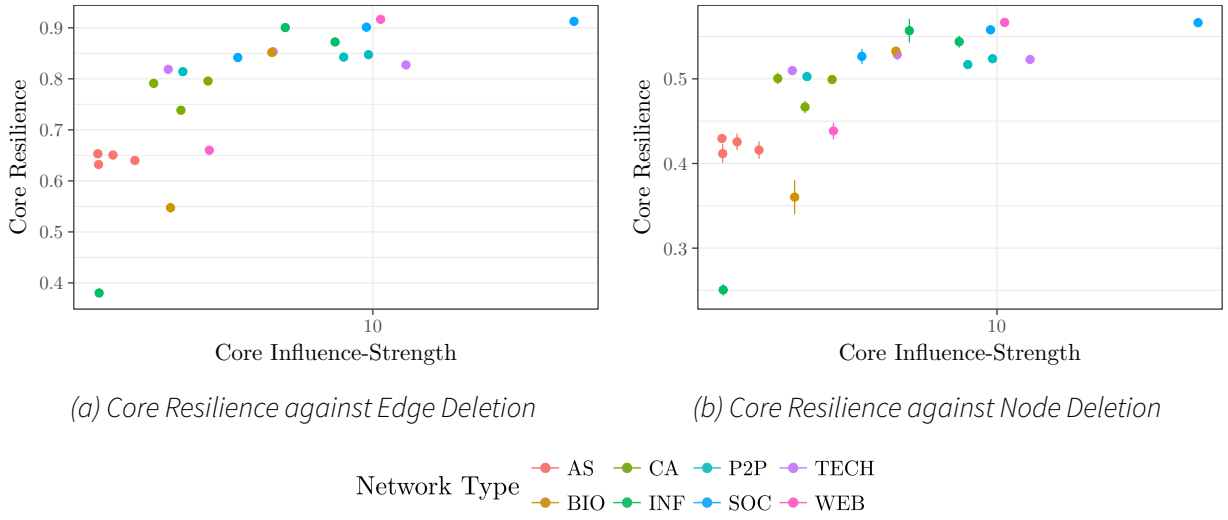
If a network has high $CIS_f(G)$ for high f, this means that the most influential nodes are unlikely to drop their core number when they lose connections to their neighbors. We expect such networks to have high core resilience. In contrast, the networks for which $CIS_f(G)$ is low are expected to have low core resilience.

4.3.5 Experiments

To verify that CIS reflects actual core resilience, we perform experiments on 22 real-world networks of different types (Table A.1). These networks were downloaded from SNAP[3] and Network Repository[4]. The Core Resilience ($\mathcal{R}_{100}^{\cdot(0,50)}(G)$) vs Core Influence-Strength ($CIS_{95}(G)$) for edge deletion is shown in Figure 4.3a, and that for node deletion is shown in Figure 4.3b.

In these figures, each point is the core resilience of a network (with the network type color-coded), and is the result of 10 experiments. We observe that, as expected, the resilience is higher for networks with high Core Influence-Strength. However the relation between Core Influence-Strength and Core Resilience is sub-linear - that is it increases rapidly for low values, but for networks high Core Influence-Strength the difference in Core Resilience is not significant. Additionally we observe that the Core Resilience of `P2P` networks generally have lower Core Resilience, while that of `SOC` networks tend to be higher in terms of both edge and node deletion.

[3] `https://snap.stanford.edu/`
[4] `http://networkrepository.com/`



(a) Core Resilience against Edge Deletion

(b) Core Resilience against Node Deletion

Figure 4.3: Core Resilience ($\mathcal{R}_{100}^{\cdot(0,50)}(G)$) against Core Influence-Strength ($CIS_{95}(G)$) for various networks. Figure 4.3a shows the core resilience against edge deletion vs Core Influence-Strength, and Figure 4.3b shows the core resilience against node deletion vs Core Influence-Strength. We can observe that the Core Resilience is higher for networks with higher Core Influence-Strength, which is consistent with what we expect.

4.4 Improving the Core Resilience of a Network

Now that we have defined the core resilience of a network and proposed measures to characterize the core resilience of a graph, in this section we address the problem of '*If we can add* b *edges, to improve the core resilience of a network without changing* k*-core structure, where should we add the edges.*

Our initial results in Section 4.3 suggest that edges should be added to bolster the nodes with high Core Influence; i.e., give them higher Core Strength, in order to increase the core resilience of the network as a whole. We propose a new algorithm called *Maximize Resilience of k-core* (`MRKC`).

Node deletion can be considered a special case of edge deletion, as deleting a node is equivalent to deleting the edges of that node (Appendix A.1). For this reason, the algorithm for improving the core resilience of a network against edge deletion is the same as for node deletion.

The `MRKC` algorithm consists of two steps: (1) generating candidate edges and (2) assigning edge

priority. We discuss these steps in detail in Sections 4.4.1 and 4.4.2 respectively.

4.4.1 Generating Candidate Edges

Given a network $G = \langle V, E \rangle$, the first step in `MRKC` is to determine which edges can be added to the network without changing the k-core structure. Let G' be the graph after adding the edges, then the k-core structure does not change if,

$$\forall u, v \in V,\ \kappa(u, G) \square \kappa(v, G) \implies \kappa(u, G') \square \kappa(v, G'), \tag{4.13}$$

where $\square$ can be $<$, $>$ or $=$. There are two ways to satisfy this:

1. The coreness of no nodes changes. That is, $\forall u \in V, \kappa(u, G) = \kappa(u, G')$.
2. If the coreness of node u increases[5], all the nodes with higher or same coreness also has to increase by the same amount.

Because changing the coreness of a lot of nodes may not be possible in many cases (because we might need to add more edges than allowed), we make sure that the coreness of no node change during the edge addition.

Let E' be the set of edges that do not exist in G. The size of E' is on the order of $|V|^2$. This is clearly too many edges to check, so we need a method to quickly filter out the edges that would change the coreness if added to G.

`MRKC` accomplishes this by adapting the purecore-based method described in [85], which examines the endpoint of each potential edge (the *purecore* of a node u is the set of nodes that have the same coreness as u and could be affected by a change in the coreness of u).

Let us denote the purecore of node g in graph G by $PC(u, G)$. We split E' into two sets E_{sim} and E_{dif},

[5] It is not possible for coreness to decrease due to edge addition.

such that, $\kappa(u, G) = \kappa(v, G)$ for all $(u, v) \in E_{sim}$; and $\kappa(u, G) \neq \kappa(v, G)$ for all $(u, v) \in E_{dif}$.

From the set E_{sim}, we generate subsets E^i_{sim} such that:

1. $\bigcup E^i_{sim} \equiv E_{sim}$; i.e. is all edges in E_{sim} are in some E^i_{sim}.
2. $E^i_{sim} \cap E^{j \neq i}_{dif} \equiv \emptyset$; i.e. all E^i_{sim} are disjoint.
3. No two edges in E^i_{sim} are connected via the nodes that have same core number with the endpoints of those edges.

Because all the edges have endpoints that are not in the other's purecore, we can insert E' to G, and if there is a node that changes coreness, we can pinpoint which edge in E' caused it. Assume that there are n_{sim} such subsets.

Similarly, we split E_{dif} into subsets E^i_{dif} in the same way as E_{sim}, but with additional conditions that if there are two edges in E^i_{dif} that have the same endpoints, the other two nodes cannot have the same coreness.

Again in this case if on adding E^i_{dif} to G, the coreness of any node changes, we can identify which edge in E^i_{dif} caused that. Let us assume that there are n_{dif} such subsets.

Then, instead of checking all $|E'|$ edges one-by-one, we need to check only $n_{sim} + n_{dif}$ times.

We can further speed up the generation of the candidate edges. Assume that $E^i_{\cdot}$ is the set of nodes currently being tested. Let k_{min} and k_{max} be the minimum and maximum core number of the nodes involved in $E^i_{\cdot}$. Then, adding the $E^i_{\cdot}$ can only change the core numbers of nodes u where $k_{min} \geq \kappa(u, G) \geq k_{max}$.

So, instead of running k-core decomposition on the entire network after adding the edges, we can add the edges to the k_{min}-core subgraph of the original network, and run the k-core decomposition on the subgraph. Again because, no node with core number above k_{max} will be affected, we do not need to run the k-core decomposition to completion - we can stop after the k_{max}-core has been

found.

4.4.2 Assigning Edge Priority

After obtaining the set of edges that can be added to the network, `MRKC` selects which subset of edges to add. To do this, `MRKC` assigns each edge $(u, v) \in E'$ a priority based its endpoints u and v. As discussed before, the goal is to improve the core strength of the nodes with high core influence. So the priority value for each node u is assigned as $\frac{CI(u)}{CS(u)}$.

There are three cases that needs to be considered based on the coreness of the endpoints, u and v: (a) $\kappa(u, G) > \kappa(v, G)$, (b) $\kappa(u, G) < \kappa(v, G)$, and (c) $\kappa(u, G) = \kappa(v, G)$.

In the case of $\kappa(u, G) > \kappa(v, G)$, addition of the edge (u, v) will only affect $CI(v, G)$; $CI(u, G)$ will be unaffected. On the other hand, if $\kappa(u, G) = \kappa(v, G)$, both $CI(u, G)$ and $CI(v, G)$ will be affected by addition of (u, v). So, for all $(u, v) \in E'$, `MRKC` assigns priority as,

$$\rho(u, v) = \begin{cases} \frac{CI(u,G)}{CS(u,G)} & \text{if } \kappa(u, G) < \kappa(v, G) \\ \frac{CI(v,G)}{CS(v,G)} & \text{if } \kappa(u, G) > \kappa(v, G) \\ \frac{CI(u,G)}{CS(u,G)} + \frac{CI(v,G)}{CS(v,G)} & \text{if } \kappa(u, G) = \kappa(v, G) \end{cases} . \tag{4.14}$$

At each step, `MRKC` selects the edge with the highest priority and adds it to the network until we reach the budget, i.e., maximum number of edges allowed to be added. The set E' needs to be updated after any edge (u, v) is inserted, but we can make it efficient by checking only for those edges that has an endpoint in $PC(u, G) \cup PC(v, G)$. Updates to core influence and core strength can also be done in similar way.

		Edge Deletion ($\mathcal{R}_{50}^{e(0,50)}$)					Node Deletion ($\mathcal{R}_{50}^{n(0,25)}$)				
Type	Network	Original	MRKC	RANDOM	DEGREE	CORE	Original	MRKC	RANDOM	DEGREE	CORE
AS	AS_733_19971108	0.58	**0.65**	0.60	0.61	0.58	0.35	**0.44**	0.40	0.38	0.36
	AS_733_19990309	0.62	**0.72**	0.65	0.67	0.62	0.36	**0.48**	0.41	0.43	0.37
	Oregon1_010331	0.66	**0.78**	0.71	0.72	0.72	0.42	**0.49**	0.45	0.44	0.45
	Oregon1_110428	0.67	**0.79**	0.72	0.72	0.71	0.41	**0.50**	0.46	0.42	0.44
BIO	BIO_Dmela	0.80	**0.84**	0.82	0.83	0.83	0.48	**0.55**	0.49	0.49	0.48
	BIO_Yeast_Protein	0.49	**0.71**	0.55	0.57	0.56	0.34	**0.47**	0.38	0.37	0.37
CA	CA_GrQc	0.75	**0.81**	0.74	0.76	0.74	0.43	**0.51**	0.43	0.42	0.42
	CA_HepTh	0.69	**0.78**	0.71	0.70	0.72	0.40	**0.45**	0.38	0.40	0.41
	CA_Erdos992	0.69	**0.72**	0.70	0.69	0.71	0.44	**0.49**	0.42	0.43	0.43
INF	INF_OpenFlights	0.87	**0.89**	0.88	0.87	0.87	0.51	**0.57**	0.51	0.52	0.51
	INF_Power	0.49	**0.77**	0.36	0.42	0.38	0.29	**0.46**	0.26	0.25	0.27
P2P	P2P_Gnutella08	0.73	**0.79**	0.72	0.75	0.73	0.40	**0.51**	0.43	0.45	0.43
	P2P_Gnutella09	0.71	**0.78**	0.73	0.72	0.73	0.39	**0.50**	0.42	0.45	0.43
	P2P_Gnutella25	0.69	**0.81**	0.71	0.73	0.74	0.39	**0.47**	0.41	0.40	0.41
SOC	SOC_Hamster	0.84	**0.86**	0.85	0.85	0.85	0.50	**0.54**	0.52	0.52	0.50
	SOC_Wiki_Vote	0.76	**0.82**	0.75	0.77	0.77	0.43	**0.51**	0.45	0.45	0.47
	SOC_Advogato	0.88	**0.91**	0.89	0.88	0.89	0.52	**0.61**	0.52	0.50	0.51
TECH	TECH_Ppg	0.81	**0.86**	0.81	0.81	0.82	0.47	**0.53**	0.49	0.50	0.51
	TECH_Router_rf	0.83	**0.86**	0.83	0.83	0.83	0.49	**0.55**	0.51	0.48	0.48
	TECH_Whois	0.89	**0.91**	0.89	0.89	0.89	0.52	**0.65**	0.57	0.59	0.59
WEB	WEB_Spam	0.87	**0.90**	0.88	0.87	0.87	0.51	**0.56**	0.51	0.52	0.52
	WEB_Webbase	0.61	**0.75**	0.60	0.59	0.60	0.38	**0.45**	0.42	0.43	0.44

Table 4.1: Improvement in Core Resilience of the top 50% *nodes (by core number) on adding* 5% *new nodes by* ***MRKC****, random (****RANDOM****), highest mean degree (****DEGREE****) and highest mean core number (****CORE****) of the endpoints. It can be observed that* ***MRKC*** *outperforms all the baselines.*

4.4.3 Experiments

To evaluate MRKC, we added up to 5% new edges to real-world networks to improve their core resilience.

The networks we used for our experiments are given in Table A.1. Adding edges to improve core resilience is applicable to only some type of networks. For example, in social networks, we cannot force people to form connections. However, we included these kind of networks in our experiments for the sake of completeness.

For comparison, we consider three baseline methods where the edges in E' are added (1) randomly (RANDOM), (2) in decreasing order of the sum of the degrees of the endpoints (DEGREE), and (3) in decreasing order of the sum of the core numbers of the endpoints (CORE). We run each experiment 10 times, and present the mean values. In Figure 4.4, we show the comparison of the core resilience of different networks with edges added by MRKC and the three baselines. The y-axis is the core resilience, and the x-axis is the percentage of edges added. Because of space limitations, we cannot present the plots for all the networks, and so we give them in Table 4.1 when 5% new edges are added.

We observe that MRKC outperforms all considered baseline methods. In cases where the initial core resilience is low, MRKC can improve it by a large amount (for example in INF_Power, BIO_Yeast). However, if a network already has high core resilience to begin with, MRKC cannot improve it by much (as in INF_OpenFlights, TECH_Whois).

In the case of AS networks, the core resilience (with respect to both edge deletion and node deletion) is low, and after adding the edges by MRKC, the core resilience is increased significantly - up to 17.9% and 25.7% for edge deletion and node deletion respectively. However, for the TECH networks, the core resilience against edge deletion is already high. So on adding edges by MRKC, we could achieve an improvement of only 3.4%.

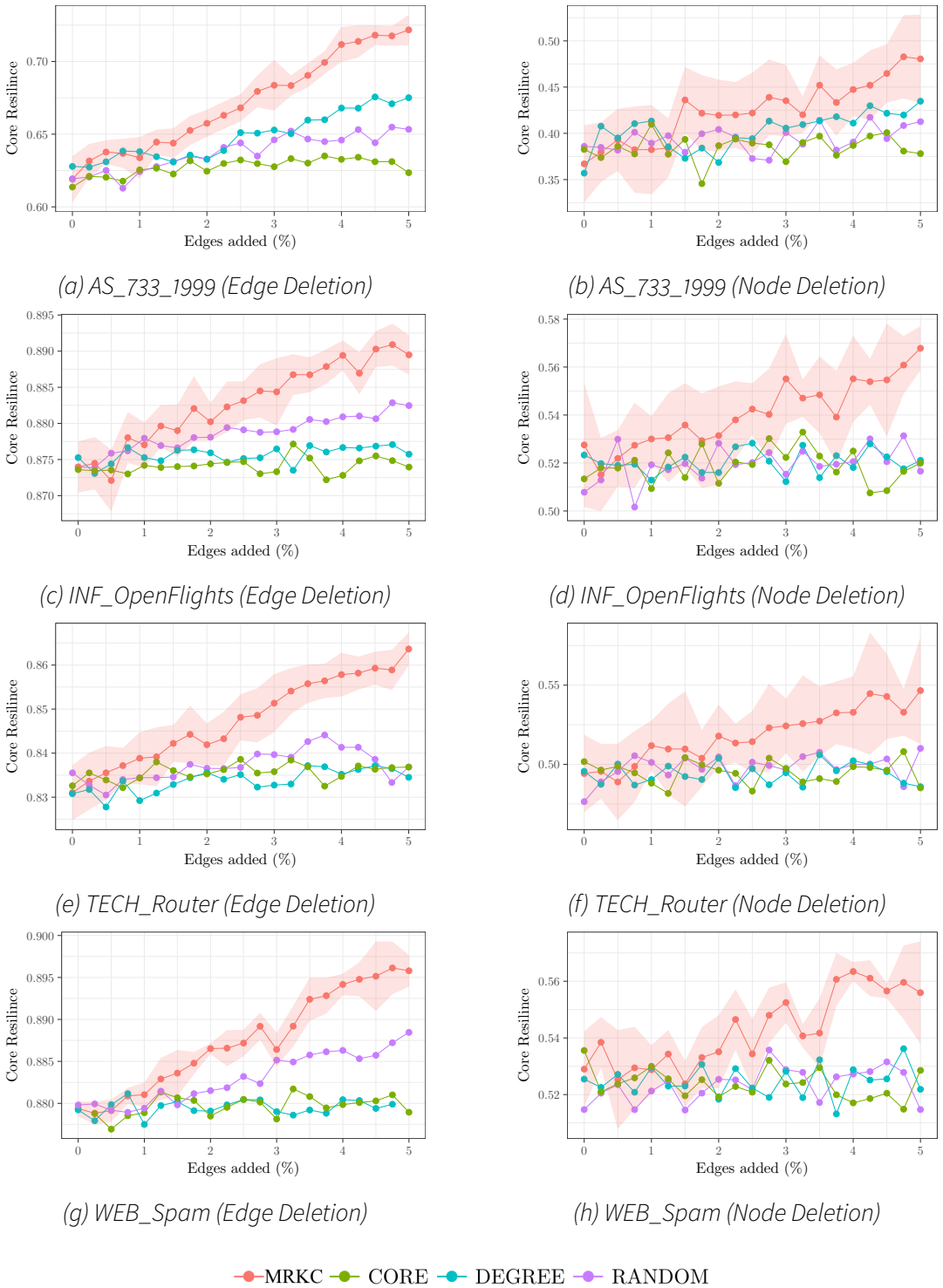


*Figure 4.4: Change in Core Resilience against percentage of new edges added for different real-world networks. The y-axis is the core resilience and the x-axis is the percentage of new nodes added by the different algorithms. The figures in the left column (Figures 4.4a, 4.4c, 4.4e, 4.4g) are for edge deletion, and those in the right column (Figure 4.4b, 4.4d, 4.4f, 4.4h) are for node deletion. In all cases, **MRKC** outperforms the baselines.*

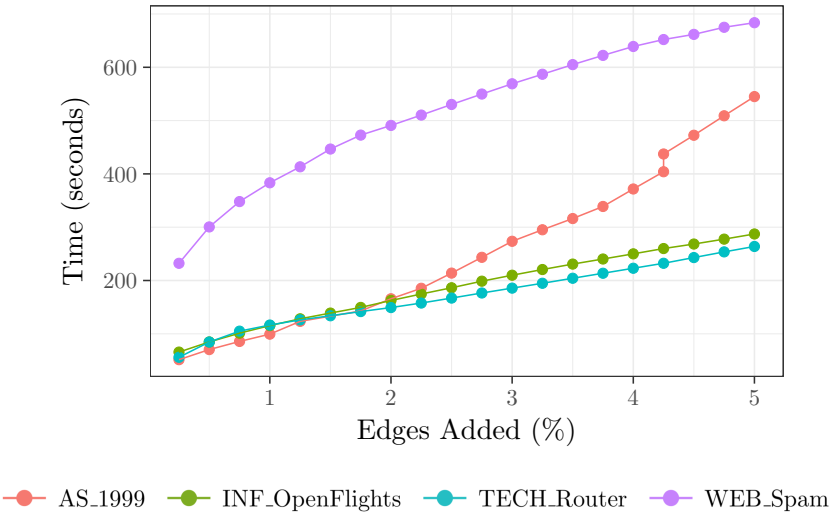


Figure 4.5: Running time of our method for improving core resilience (`MRKC`) on different networks. The x-axis is the amount of new edges added (in %), and the y-axis is the time taken to add the edges (in seconds).

In the plots shown in Figure 4.4, we observe that the rate of improvement of `MRKC` in the case of node deletion is lower than that for edge deletion in the same network. This is because the core resilience due to edge deletion cannot be less than that of node deletion (Equation A.1).

Running Time: In Figure 4.5, we show the time taken to add the new edges according to our method for four networks. The x-axis is the amount of new edges added (in%), and the y-axis is the time taken to add the edges. The values are the means over 10 runs.

`MRKC` checks for all edges that can be added without changing core number in the first step. This is why we observe in Figure 4.5 that the plots do not start at the same points. After the initial candidate edges generation, we no longer need to check all the edges - if an edge (u, v) is added, we only need to check the purecore of u and v, so the following edge insertions are faster. The only exception is the `AS_1999` network, where the runtime increases constantly. This is because there are a large number of nodes with large purecores, so subsequent checks still take a significant amount of time for this network.

4.5 Conclusion

In this chapter, we discussed the problem of capturing how the k-core structure of a network changes due to deleted edges or nodes. To address this we proposed a measure called *Core Resilience* of a network, which measures how much the ordering of the nodes by core number is affected when there are missing edges and nodes.

Computing the core resilience of a large networks is computationally expensive, and so we proposed two node measures based on network structure. The two measures - *Core Strength* and *Core Influence*, can be used together to tell us if a network is likely to have high core resilience or not. We proposed a method called Maximize Resilience of k-core (`MRKC`) to add edges to a network without changing the core number of any node, such that the core resilience of the resulting network is improved. We tested our method against baselines on multiple real-world networks, and found that it can improve the core resilience against edge deletion by 19% on average, and against node deletion by 19.7% over the original network.

Chapter 5

Core Minimization

In Chapter 4, we considered the change to the ordering of nodes based on core number due to missing edges or nodes. In some applications, the membership of the nodes in some k-core is more important than the global ordering of nodes. The literature describes the *Core Minimization* problem, which asks how likely it is that nodes in the true k-core of a graph are to be in the observed k-core of that graph if there is missing data. [68, 99, 104].

As an example consider the toy graph shown in Figure 5.1. Here all the nodes belong to the 3-core; but if the red node (or rather, any node) is deleted, they are no longer in the 3-core.

There has been various recent works on the problem of core minimization. Zhang *e al.* [99] pro-

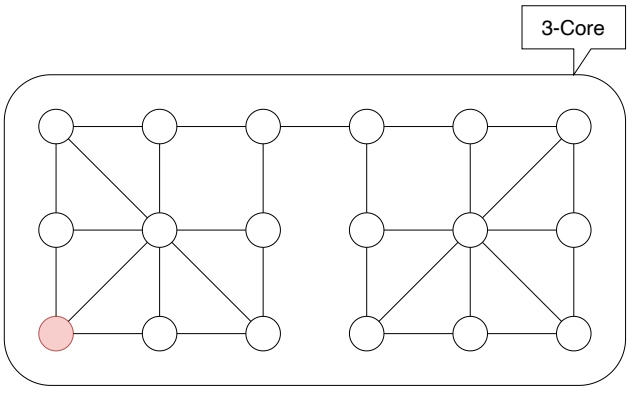


Figure 5.1: A toy graph showing collapsed k-core. The entire graph is a 3-core; but if the red node is deleted, all the rest of the nodes are no longer in the 3-core.

posed a method of finding 'critical users' – that is the nodes that when deleted reduces the size of the k-core the most. They proposed a greedy algorithm to find such critical users. Medya *et al.* [68] showed that solving the core minimization problem is NP-hard.

In contrast to these earlier works, our goal is not to find the set of nodes that minimizes the k-core by the most – but rather to characterize the resilience of the k-core of a graph to such minimization attacks. In this chapter, we try to answer the following questions:

1. How can we characterize the resilience of a k-core to core minimization?
2. If we can *anchor* some nodes, which nodes should we select to improve the resilience to core minimization?

Another very closely related problem is the *Anchored k-Core* problem [14]. In the anchored k-core problem, one seeks to find a set of nodes to 'anchor,' or retain within the anchored k-core, even if their degree within the k-core subgraph is less than k: other nodes in the anchored k-core must thus have at least k connections either to other nodes in the subgraph or to the anchors. The objective of the anchored k-core problem is to maximize the size of the resulting *a*nchored k-core [15], in hopes of preventing a cascading exodus. We will describe this in Chapter 6.

5.1 Motivating Application

Because the k-core of a graph gives us the 'central' nodes in the graph, there are various applications that depends on the membership of the nodes in the k-core. Here, we describe a few examples.

Example 1: k-core in the WWW has been used to identify web-spam. In [57], the authors found that the spam nodes are grouped together with other spam nodes in a connected component of the k-core. They proposed a method to identify the spam nodes using this information.

If the resilience of the k-core to core minimization is low, the spam nodes can delete some nodes/edges to better hide the other spam nodes. So, it is important to understand the resilience of k-core to core minimization.

Example 2: A financial network is one where the nodes are publicly traded companies and they are connected by an edge if the similarity in their trading pattern determines if two nodes are connected [65]. In [22], the authors studied the robustness of such financial networks. They found that the size of the k-core (for a high value of k) is a good indicator of the robustness of the financial system. They found that if the distribution of the nodes in the different k-shells follows a U-shape – more nodes in very high and very low k-shells and very few nodes in the intermediate, the financial network is more robust against external shocks.

An attacker (with enough capability) can manipulate edges in such network by manipulating the trading behavior of some companies. If the goal of the attacker is to reduce the robustness of the system they can manipulate the edges with the object of minimizing the size of the k-core (for high k) so that most of the nodes falls to intermediate shells. So, it is important to understand not only how robust the financial system is to external shocks, but also the core minimization attacks. If the resilience to core minimization attack is low, it is also important to identify which are the companies that needs to be kept alive (i.e. anchored) so that as to improve the resilience to such core minimization attacks.

5.2 Characterizing the Resilience to Collapsed k-Core

In the case of core resilience (Chapter 4), we are interested in measuring how resilient multiple k-cores are. So, the ordering between the different cores matters – which led to us defining it based on rank correlation. In this case we are interested in just a single k-core – we do not care if the nodes changes coreness as long as they are still in the k-core. For example, if we are interested in the 5-core, it does not matter if the coreness of a node changes from 10 to 9. All that we care is that the

node still remains in the 5-core.

Although the literature on collapse k-core generally talks about node deletion, we will also focus on edge deletion in this chapter as that simplifies the analysis. As described in Appendix A.1, node deletion can be considered as another side of edge deletion.

Let $G_k = \langle V_k, E_k \rangle$, be the k-core of a graph and let $\mathbb{G}$ be the set of graphs after removing $p\%$ of the nodes. We define the *Collapse Resilience* of the k-core of G as,

$$\mathcal{R}(G_k) = \frac{\sum_{G' \in \mathbb{G}} |\{v \in V_k : \kappa_{G'}(v) \geq k\}|}{|\mathbb{G}| \cdot |V_k|}. \tag{5.1}$$

That is, *the collapse resilience of the k-core is the expected fraction of nodes that remains in the k-core over all the possible subgraphs that results due to $p\%$ of edge removal*. If we are interested in the average fraction of collapsed nodes, $1 - \mathcal{R}(G_k)$ give us that value.

In practice, it is not possible to find $\mathbb{G}$. So we approximate it through sampling. However even the sampling method might not be computationally efficient enough for some cases, so we propose the concept of *Core Instability*.

5.2.1 Core Instability

Motivated by the idea of core strength (Section 4.3.2), we propose the the idea of *Core Instability* of the k-core which is a measure of what fraction of nodes in the k-core are likely to drop out of the k-core due to an edge deletion. We expect k-cores with high core instablity to collapse more easily.

We begin by measuring how many neighbors of a node u in G needs to drop out of the k-core for u to also drop out. We call this the *Relative Core Strength* of u with respect to the k-core, and it is

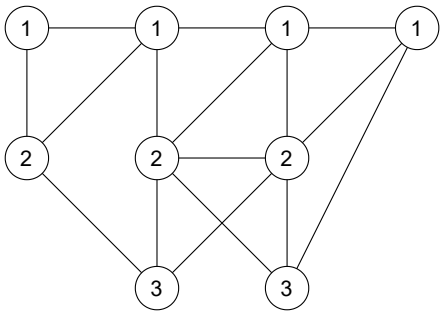


Figure 5.2: An example of a core unstable subgraph. The number inside the nodes are the relative core strength of the nodes. Notice that if any edge that has a node with relative core strength of 1 *as one endpoint is deleted, the entire structure collapses, and none of the nodes in the subgraph are in the* k*-core.*

given by,

$$rCS(u, G, k) = |\{v \in \Gamma_G(u) : \kappa_G(v) \geq k\}| - k + 1. \tag{5.2}$$

Then, we define a *Core Unstable Subgraph* in the k-core as the maximal connected subgraph of the k-core such that:

1. All nodes in the subgraph with relative core strength of 1 are connected.
2. All nodes in the subgraph with relative core strength greater than 1 are connected to as many nodes with lower relative core strength as its relative core strength.

The idea behind the core unstable subgraph is that, if any edge that has a node with relative core strength of 1 loses an edge, the entire subgraph drops core number (Theorem 5.2). As an example, consider Figure 5.2. Here the numbers inside the nodes are their relative core strengths. We can observe that if any edge that has a node with relative core strength of 1 as one endpoint is deleted, the entire subgraph collapsed out of the k-core. So, the idea of the core unstable subgraph allows us to quantify how close the k-core as a whole is to collapse.

Let $G_k = \langle V_k, E_k \rangle$, be the k-core, and let $G' = \langle V', E' \rangle$ be a core unstable subgraph. Assume $r(G')$ be the number of edges that has one a node of relative core strength of 1 as one endpoint.

That is,

$$r(G') = |\{(u, v) \in E_k : u \in V' \wedge rCS(u, G, k) = 1\}|. \tag{5.3}$$

If we are dealing with one edge deletion, then the probability of deleting one of these edges is $\frac{r(G')}{|E_k|}$. If an edge is deleted, the fraction of nodes (out of all nodes in the k-core) that drops out of the k-core is $\frac{|V'|}{|V_k|}$. So, we can define the *core instability* of a k-core, as the expected fraction of nodes that drops out of the k-core due to an edge deletion. Formally the core instability of the k-core of the graph G is given by,

$$CT(G, k) = \sum_{G' \in \mathcal{U}} \frac{r(G')}{|E_k|} \cdot \frac{|V'|}{|V_k|}, \tag{5.4}$$

where $\mathcal{U}$ is the set of all core unstable subgraphs in the k-core.

Finding all the core unstable subgraphs in the k-core is straight forward – simply start with all connected components of nodes with relative core strength 1; then incrementally add nodes of higher relative core strength that satisfies the conditions. Algorithm 3 describes this process in more details.

Algorithm 3 The algorithm to find the core unstable subgraph.

```
1:  function FindCoreUnstable(G_k)
2:      r ← RelativeCoreStrength(G_k)
3:      C ← Connected components in the subgraph induced by {u ∈ V' : r[u] = 1}
4:      r_max ← max_{u∈V_k} r[u]
5:      for S ∈ C do
6:          for k ∈ [2, 3, ..., r_max] do
7:              T ← {u ∈ Γ_{G_k}(S) : r[u] = k ∧ |Γ_{G_k}(u) ∩ S| ≥ k}
8:              Update S with T
9:          end for
10:     end for
11:     return C
12: end function
```

Theorem 5.1 (Complexity of Algorithm 3)**.** *The running time of Algorithm 3 is* $\mathcal{O}(|V_k|)$*, and the space*

complexity is $\mathcal{O}(|E|)$.

Proof. If we group together nodes by their their relative core strength, at each step, we only need to check for nodes from within a group. So, over the entire process of building up one core unstable subgraph, we would have checked each group once. The running time of the algorithm is then, $\mathcal{O}(|C||V_k|)$. Since $|C| \ll |V_k|$, we can write it as $\mathcal{O}(|V_k|)$.

At the most, the space required to store C is approximately equal to that of V. So, the space complexity of Algorithm 3 is $\mathcal{O}(|V_k|)$. □

Theorem 5.2. *For* $G' \in \mathcal{U}$*, if an edge* $(u, *)$*, such that* $rCS(u, G, k) = 1$*, is deleted from* G'*, all the nodes in* G' *drops out of the* k*-core.*

Proof. By construction we can see that all the nodes with relative core strength of 1 will drop out of the k-core.

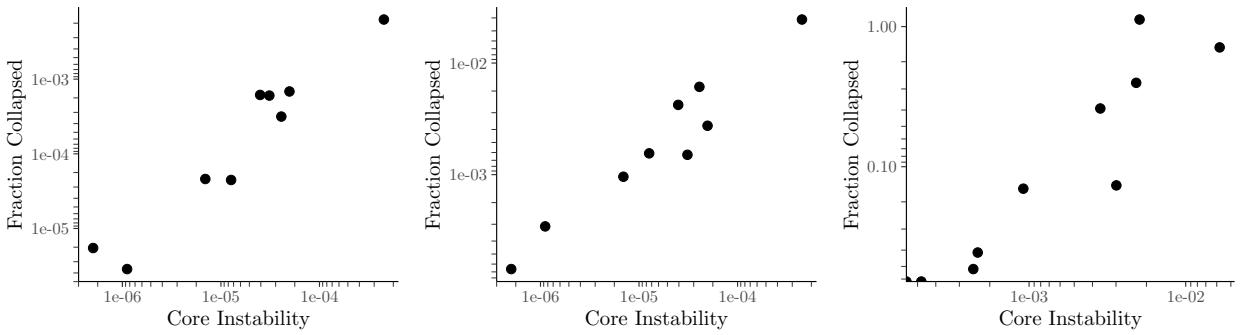
As a result, the nodes with relative core strength of 2 will also drop out of the k-core.

Through the same argument, all the nodes in the core unstable graph will drop out of the k-core. □

5.2.2 Experiments

To test if the size of collapse in the k-core with higher core instability is larger than those with lower instability, we perform experiments on real-world networks. We consider three cases: (1) random edge deletion, (2) random node deletion, and (3) greedy node deletion [99].

In the random edge deletion, an edge that connects two node in the k-core is randomly deleted. The random node deletion is similar except we delete nodes. In the greedy node deletion, the node that minimizes the size of the k-core the most if deleted is deleted at each step. In all these cases after each deletion the nodes that are in the k-core is updated. For greedy node deletion, we consider



(a) Random Edge Deletion *(b) Random Node Deletion* *(c) Greedy Node Deletion*

Figure 5.3: Fraction of nodes that collapsed due to random edge (fig. 5.3a), random node (Figure 5.3b) and greedy node [99] (Figure 5.3c) against the Core Instability for various real-world graphs (denoted by the dots). Here, the number of nodes or edges deleted is 20 *(*5 *for greedy node deletion), and we consider the* 10*-core. We can observe that in networks with higher core instability, the collapse is higher.*

only small graphs because the algorithm scales very poorly with the number of nodes.

Figure 5.3 shows the fraction of nodes that drops out of the 10-core against the core instability due to random edge deletion (Figure 5.3a), random node deletion (Figure 5.3b), and greedy node deletion (Figure 5.3c). In these figures, the dots represents different networks from various domains ranging from social networks to biological networks. The number of edges/nodes deleted for the random case is 20 and it is 5 for the greedy algorithm (because greedy algorithm is very slow). We consider the 10-core in all the cases. Because the greedy algorithm scales very poorly with the network size, we consider only small networks for Figure 5.3c. The values of fraction collapsed given are the average values of 30 trials.

We can see that in networks with higher core instability, a larger fraction of nodes drops out of the k-core in all the cases. This indicates that the core instability gives us a measure of the collapse resilience of a graph – graphs with higher core instability have lower collapse resilience.

5.3 Anchoring Nodes to Minimize Collapse

In this section we discuss ways to minimize the collapse of the k-core due to node or edge deletion. To do this we refer to the idea of 'anchored k-core' [14]. The anchored k-core of a graph $G = \langle V, E \rangle$, is defined as the subgraph such that all the nodes in the subgraph have at least k neighbors within the subgraph or a set $A \subseteq V$. The set of nodes A is called the set of 'anchor' nodes. Through appropriate selection of these anchor nodes, we seek to minimize the collapse.

As an example, in the context of a social network, we can think of the anchor nodes as those users who are given incentives to remain in the network. Of course, we are limited with the number of anchor nodes we can select – we will call this the *anchor budget*. This is related to the *anchored k-core* problem [98, 54], and we will discuss this in the next chapter.

Let $\hat{\kappa}(u, G, A)$ be the core number of node u in graph G anchored with set of nodes $A \subseteq V_k$. Then, the collapse resilience in the presence of the anchors A is given by simply replacing $\kappa(*)$ in Equation (5.1) to $\hat{\kappa}(*)$. That is,

$$\hat{\mathcal{R}}(G_k, A) = \frac{\sum_{G' \in \mathbb{G}} |\{v \in V_k : \hat{\kappa}(v, G', A) \geq k\}|}{|\mathbb{G}| \cdot |V_k|}. \tag{5.5}$$

If b is the number of anchors allowed, the goal is to find A^* such that,

$$A^* = \underset{A \subset [V_k]^b}{\arg\max}\, \hat{\mathcal{R}}(G_k, A). \tag{5.6}$$

5.3.1 Shortcoming of Naive Method

If we have a method of selecting nodes to remove to minimize the size of the k-core, a naive method of selecting anchor nodes might to be simply anchor the solutions from the method – preventing them from deletion or dropping out of the k-core. However, as we will show that does not always

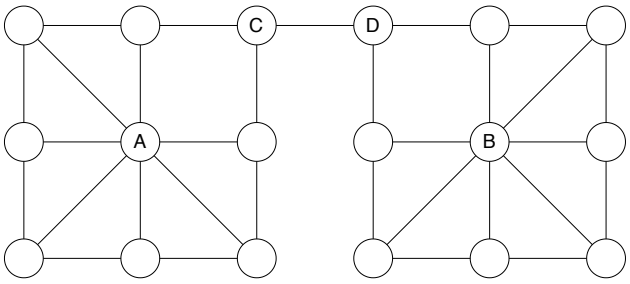


Figure 5.4: Toy example demonstrating the shortcomings of the naive method of anchor selection in increasing the collapse resilience. In the naive method, either node A or B will be selected as anchors. However, we can see that even after anchoring node A or B, any edge deletion collapses the entire 3-core.

give good anchor nodes.

The k-core minimization technique we will use here is the greedy algorithm `CKC` proposed in [99]. Basically, the idea is to greedily select the node whose deletion results in the largest decrease in the size of the k-core. In the naive adaptation of this method, instead of deletion, these nodes are anchored.

Consider the graph shown in Figure 5.4, where all the nodes are in the 3-core. Suppose that we are dealing with one node deletion, and one anchor selection. The naive method will select node A or B because deletion of either of these nodes results in the largest number of nodes dropping out of the 3-core (and they have the highest degree). Suppose that node A is anchored. In that case the greedy node deletion algorithm will delete node B – resulting in the entire 3-core collapsing (except the anchor node). If we anchored node C or D, only half of the nodes in the 3-core will collapse.

This example demonstrates that the naive method of anchor selection does not work because all that the naive method does is to remove one solution from the `CKC` algorithm.

5.3.2 Maximizing the Collapse Resilience of the k-Core

The idea of *core unstable subgraphs* motivates our algorithm for anchor selection. The core unstable subgraph is defined as the maximal connected subgraph of the k-core such that all the nodes

in the subgraph with relative core strength of 1 are connected, and all the nodes other nodes in the subgraph are connected to as many nodes wit lower relative core strength as its relative core number 5.2.1.

We know that in a core unstable subgraph if an edge with one endpoint at a node with relative core strength of 1 is deleted, the entire subgraph drops out of the k-core (Theorem 5.2). By definition the anchor nodes cannot drop out of the k-core. So, the anchor nodes should not be considered as part of the core unstable subgraph. Thus, in the presence of anchor nodes, we redefine the core unstable graphs to exclude the anchor nodes. As a consequence, edges adjacent to an anchor node can also not be candidates for removal. That is Equation (5.3) has to be updated as,

$$\tilde{r}(G') = |\{(u, v) \in E_k : u \in V' \setminus A \wedge v \notin A \wedge rCS(u, G, k) = 1\}|. \tag{5.7}$$

So, given a core unstable subgraph, G', the anchor nodes can serves two functions: (1) minimize $|V'|$, the size of the core unstable subgarph, and/or (2) minimize $r(G')$, the number of edges with an endpoint a node in V' with relative core strength of 1. This is the intuition behind our anchor selection algorithm, which we call *Core Instability Minimization* (`CIM`).

Consider a core unstable subgraph $G' = \langle V', E' \rangle$ of the k-core G_k, and let $r(G')$ be the number of edges with an endpoint in a node in V' with relative core strength of 1. If a node u is anchored, let $\delta(u, G', A)$ be the relative size of the resulting core unstable subgraph. That is,

$$\delta(u, G') = \begin{cases} \frac{|V'|}{|V_k|} & \text{if } u \notin V' \\ \frac{|\{v \in V' : rCS(v, G') \leq |N(v, G') \setminus \{u\}|\}|}{|V_k| - 1} & \text{otherwise} \end{cases}. \tag{5.8}$$

Let $\gamma(u, G')$ be the relative number edges whose deletion leads to the collapse of G' (excluding

the ones with endpoint at u). Then,

$$\gamma(u, G') = \frac{|\{(x, y) \in E_k : x \in V' \setminus \{u\} \wedge rCS(x, G') = 1 \wedge y \notin A \cup \{u\}\}|}{|\{(x, y) \in E_k : x, y \notin A \cup \{u\}\}|}, \tag{5.9}$$

where A is the set of anchor nodes already selected.

Then, for each node $u \in V$, the drop in core instability due to u is then given by,

$$\sum_{G' \in \mathcal{U}} \left(\frac{r(G')|V'|}{|E_k||V_k|} - \gamma(u, G')\delta(u, G') \right). \tag{5.10}$$

If we set,

$$\alpha(u) = \sum_{G' \in \mathcal{U}} \gamma(u, G') \cdot \delta(u, G'), \tag{5.11}$$

at each step we select the node with the **lowest** $\alpha(*)$ and anchor it. The process is repeated until the required number of anchors are selected. Algorithm 4 gives the `CIM` algorithm in more details. Note that `FindCoreUnstableAnchored()` is similar to Algorithm 4, except that we take into consideration anchor nodes.

Theorem 5.3 (Complexity of Algorithm 4)**.** *The time complexity of Algorithm 4 is $\mathcal{O}(b|V_k|)$; and the space complexity is $\mathcal{O}(|V_k||)$.*

Proof. Finding the core unstable subgraphs can be done in $\mathcal{O}(|V_k|)$. For each core unstable subgraph, we need to update the scores for at most $|V_k|$ nodes. So, to find one anchor, the running time is $\mathcal{O}(|V_k| + |\mathcal{U}||V_k|) \approx \mathcal{O}(|V_k|)$, since $|\mathcal{U}| \ll |V_k|$. So, the running time of `CIM` to find b anchors is $\mathcal{O}(b|V_k|)$.

The space required during `FindCoreUnstableAnchored` is $\mathcal{O}(|V_k|)$. No additional space is required during the other steps. So, the space complexity is $\mathcal{O}(|V_k||)$. □

Example: To demonstrate `CIM` with a working example, let us consider the toy example we con-

Algorithm 4 The algorithm for Core Instability Maximization (`CIM`).

function CIM(G_k)
  $A \leftarrow \emptyset$
  while $|A| < b$ **do**
    $\alpha : V \twoheadrightarrow 0$
    $\mathcal{U} \leftarrow \mathrm{FindCoreUnstableAnchored}(G_k, A)$
    for $G' \in \mathcal{U}$ **do**
      for $u \in V'$ **do**
        $\alpha(u) \leftarrow \alpha(u) + \gamma(u, G') \cdot \delta(u, G')$
      end for
      for $u \in \Gamma_{G_k}(V') \setminus V'$ **do**
        $\alpha(u) \leftarrow \alpha(u) + \gamma(u, G') \cdot |V'|$
      end for
    end for
    $u \leftarrow \arg\min_{v \in V_k} \alpha(u)$
    $A \leftarrow A \cup \{u\}$
  end while
  return $\mathcal{A}$
end function

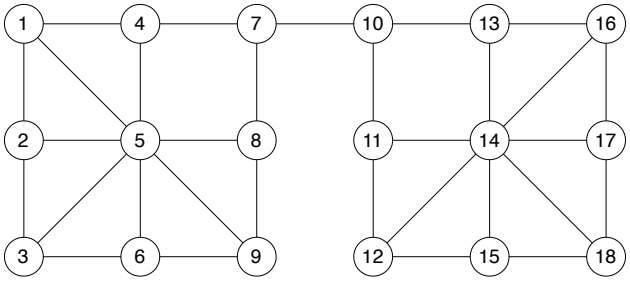


Figure 5.5: Toy example demonstrating the shortcomings of the naive method of anchor selection in increasing the collapse resilience.

sidered in Section 5.3.1 shown in Figure 5.5 again. We are considering only one anchor and node deletion in this example. As demonstrated earlier, removal of node 5 results in the entire 3-core collapsing.

In this example, the entire graph is one core unstable subgraph. So, the core instability 1 – that is, whatever edge we delete, the entire 3-core will collapse.

The $\alpha(*)$ for all the nodes are given in Table 5.1. We can see that nodes 7 and 10 have the lowest $\alpha(*)$ scores. If either of them are selected as anchor, the expected fractions of nodes that collapse

Nodes	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	18
$\alpha(*)$	1.0	1.0	1.0	1.0	1.0	1.0	0.5	1.0	1.0	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Table 5.1: Values of $\alpha(*)$ for the toy example.

is reduced from 1 to 0.49. This is clearly much better than the naive method.

5.3.3 Experiments

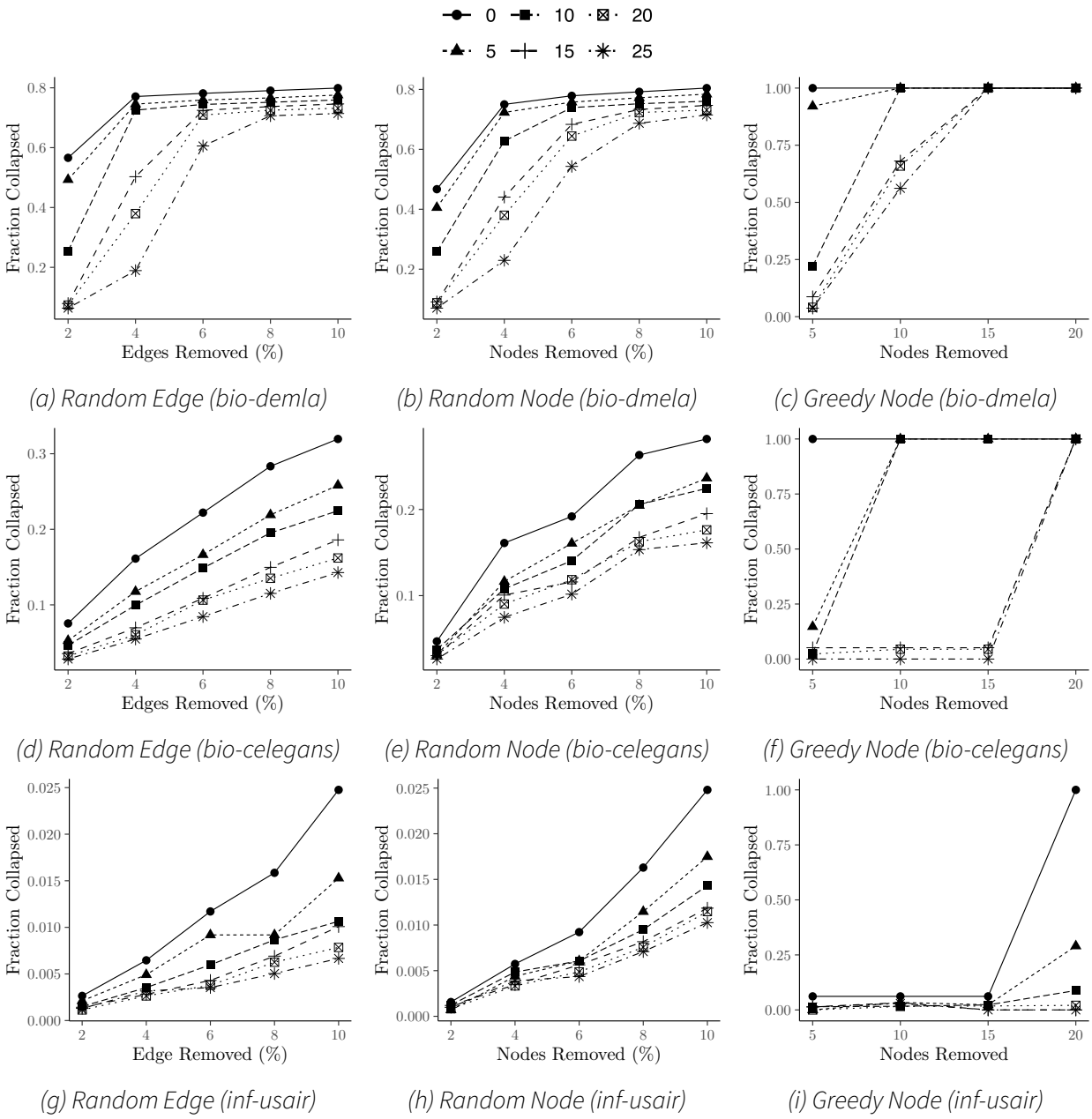
To evaluate the effectiveness of `CIM`, we perform experiments on multiple real-world graphs. We consider random edge removal, random node removal and greedy node removal. Because the greedy node removal is slow for larger graphs, we consider only small graphs.

We perform two types of experiments: (1) performance comparison of `CIM` for different numbers of anchor nodes, and (2) performance comparison between `CIM` and baseline algorithms.

Experiment 1: For the first experiment, we use only `CIM` as the anchor selection algorithm. We vary the number of anchors from 0 to 25 in steps of 5. For all the experiments we consider only the 10-core. The datasets we use are *bio-dmela*, *bio-celegans* and *unf-usair*, and these datasets are all publicly available. We selected these datasets because the greedy node removal algorithm is slow on larger graphs.

Figure 5.6 shows the fraction of collapsed nodes against the number of removed edges/nodes for various number of anchor nodes. We can see that in all the cases, increasing the number of anchors reduces the fraction of collapsed nodes. In the cases of bio-celegans and inf-usair graphs, adding 25 anchors reduces the collapsed nodes by more than half. Note that the anchored nodes are not counted in calculating the fraction of collapsed nodes (either as not collapsed or as being in the k-core).

The results for the greedy node removal in the case of bio-dmela and bio-celegans is very interesting. We can observe that with just 5 nodes removal, the entire 10-core collapses. However, by anchoring



(a) Random Edge (bio-demla) *(b) Random Node (bio-dmela)* *(c) Greedy Node (bio-dmela)*

(d) Random Edge (bio-celegans) *(e) Random Node (bio-celegans)* *(f) Greedy Node (bio-celegans)*

(g) Random Edge (inf-usair) *(h) Random Node (inf-usair)* *(i) Greedy Node (inf-usair)*

Figure 5.6: Fraction of nodes that collapsed from the 10*-core against the edges/nodes removed for different number of anchors selected through* ***CIM****. The different lines represents different amount of anchor nodes. It can be observed that in all the cases, selecting more anchors results is lower fraction of collapsed nodes.*

nodes selected by `CIM` the number of nodes whose removal required for the collapse increases to 15 and 20 respectively.

Experiment 2: For the second experiment, we compare `CIM` against other baselines in reducing the collapse. We consider three baseline algorithms: **Random** (anchors selected randomly from V_k), **Degree** (nodes in V_k with highest degree selected as anchors), and **Naive** (described in Section 5.3.1). We consider 25 anchor nodes for this experiment, and consider the same three types of collapse as before: random edge deletion, random node deletion and greedy node deletion.

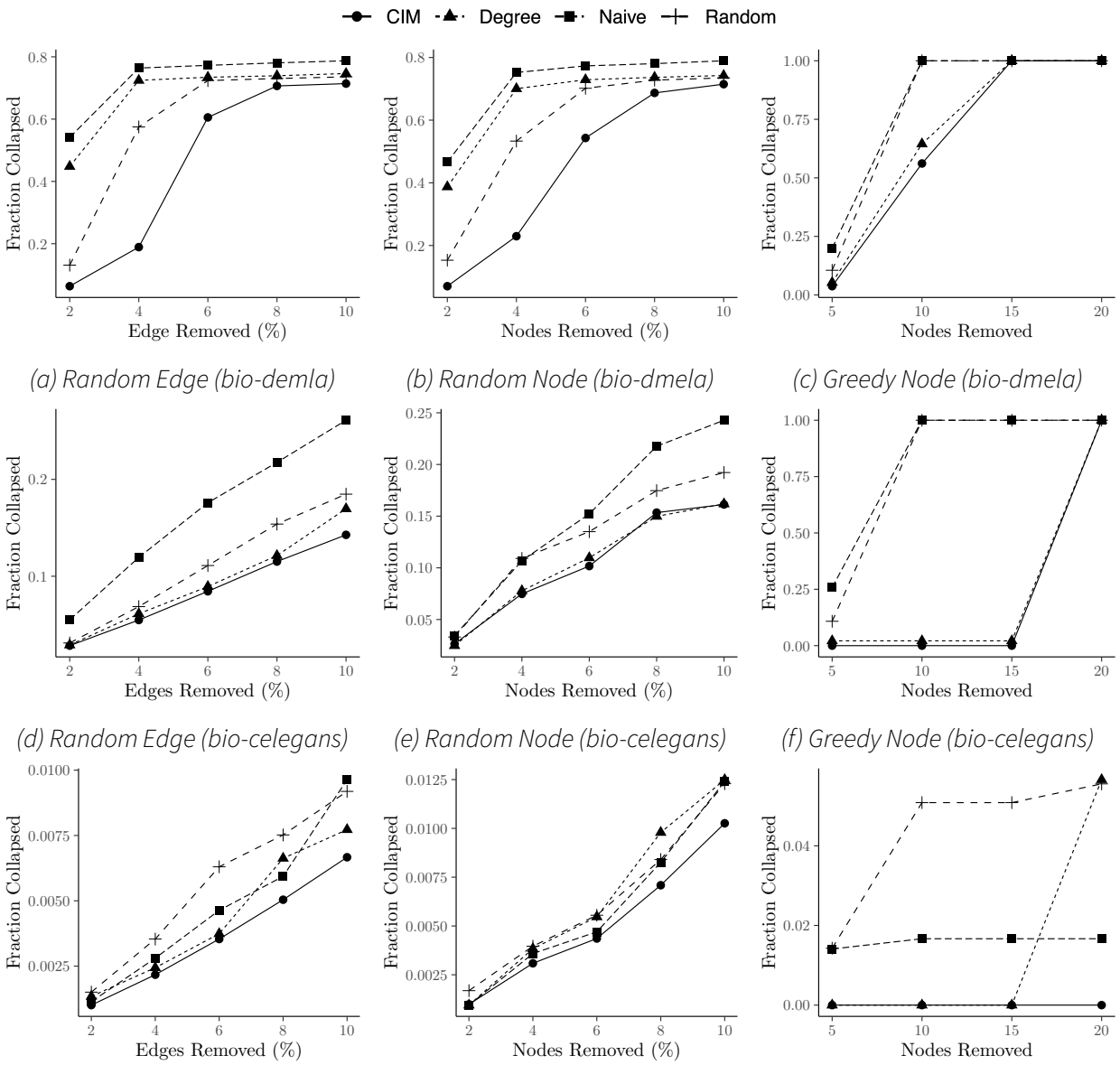
Figure 5.7 shows the comparison of `CIM` against various baseline algorithms. In all the cases, `CIM` results in smaller collapse against all three – random edge deletion, random node deletion and greedy node deletion.

Among the baselines, the performance of the different algorithms varies wildly. In bio-dmela, random performs better than the other baselines for the random deletions – but degree outperforms it in the greedy node removal. Generally greedy seems to work reasonably well among the baselines. Of particular interest is the performance of the naive anchor selection based on the greedy algorithm. In most of the cases, it performs the worst.

These results (Experiments 1 & 2) show the effectiveness of `CIM` in preventing/minimizing the collapse of the k-core.

5.4 Conclusion

In this chapter we discussed the resilience of a single k-core to collapse – that is if we are interested not in the global k-core structure, but only care about how many nodes in the k-core remain when there is missing data. To this end, we proposed a measure called *Core Instability* that can tells us how likely a cascading collapse is likely to happen in a graph. We then use show experimentally that this measures works in real-world graphs.



(a) Random Edge (bio-demla) *(b) Random Node (bio-dmela)* *(c) Greedy Node (bio-dmela)*

(d) Random Edge (bio-celegans) *(e) Random Node (bio-celegans)* *(f) Greedy Node (bio-celegans)*

(g) Random Edge (inf-usair) *(h) Random Node (inf-usair)* *(i) Greedy Node (inf-usair)*

Figure 5.7: Fraction of nodes that collapsed from the 10*-core against the edges/nodes removed for different anchor selection algorithms. The different lines represents different anchor selection algorithms. It can be observed that in all the cases,* ***CIM*** *outperforms all the other algorithms. In bio-celegans network, anchors selected based on* ***Degree*** *performs as well as* ***CIM****. In all these experiments the number of anchors is* 25*.*

Motivated by this measure, we then consider the problem of anchoring nodes to minimize that cascading collapse – both to randomly missing data and more targeted attacks. We propose an algorithm called *Core Instability Maximization* to select the anchors, and we show that it minimizes the collapse in real-world graphs.

In the next chapter, we consider a related problem called the anchored k-core problem. If we want to maximize the size of the anchored k-core by anchoring a fixed number of nodes, which ones should we select?

Chapter 6

Graph Unraveling

In Chapter 5, we discussed the resilience of a k-core to cascading collapse. We presented a measure to quantify how close a graph is to such collapse and proposed a method of selecting anchor nodes to minimize the cascade. In this chapter, we consider the *anchored k-core problem* [14, 15].

The participation of a person in social networking platforms is often motivated by the participation of others [60]. People take part in such platforms in order to engage with others; and in return, they produce content that appeals to others. In other words, people's incentives for participation on a platform depend partially on the number of people to whom they can reach. When these incentives are low, people may leave the platform. This decreased participation may affect the participation of others, further decreasing the incentives for participation. Considering the social-networking platform as a complex network among people, locally decreased participation may cause a cascading exodus from the platform. Finding (and incentivizing) the critical individuals whose active participation are key to the larger participation in the network is an essential problem.

As an example consider the example graph shown in Figure 6.1. Here the green nodes have a core number of 3, the blue ones have a core number of 2 and the red node has a core number of 1. Suppose a user stay on the platform if at least 3 friends are also on the platform. Then, the red node will leave as it has only one friend – this in turn causes the number of friends of the blue node to drop to 2 and they will also leave. At the end only the green nodes will remain active on the

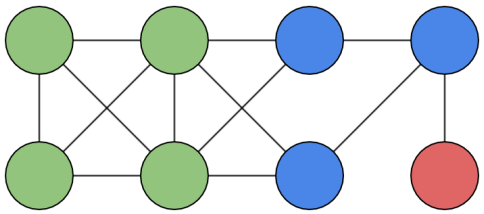


Figure 6.1: An anchored k-core example. The green nodes form a 3-core. If the red node is anchored, the entire graph becomes an anchored 3-core.

network. This cascading exodus of nodes/users was first described by [14] as the graph unraveling problem. We present more motivating applications inSection 6.1.

One way to prevent this graph unraveling is to anchor some nodes – that is give some incentives to some nodes to remain active on the platform. These nodes are referred as the anchored nodes. In the anchored k-core problem [14], one seeks to find a set of nodes to 'anchor,' or retain within the anchored k-core, even if their degree within the k-core subgraph is less than k: other nodes in the anchored k-core must thus have at least k connections either to other nodes in the subgraph or to the anchors. The objective of the anchored k-core problem is to maximize the size of the resulting *anchored k-core* [15], in hopes of preventing a cascading exodus. In the literature, the nodes (excluding the anchors) that are in the anchored k-core but not in the original k-core are called *followers*. Given a fixed number of anchors, finding the optimal sets of anchors to maximize the size of the k-core is known to be NP-hard for $k > 2$ [14].

If we take a look at the example graph in Figure 6.1 again, we can see that if the red node is anchored, the rest of the nodes become a part of the anchored 3-core – thus preventing the graph unraveling.

The algorithmic challenge behind the anchored k-core problem lies in the ability to foresee cumulative effect of groups of anchor nodes, not just individual nodes. It is possible that the addition of the first few anchor nodes make no difference, but the addition of one more anchor makes a drastic difference. A good algorithm should be able to foresee the big future pay-off even when the immediate benefits are small. This ability to foresee future benefits becomes essential especially when the budget for anchored nodes is large.

We propose `Residual Core Maximization (RCM)`, a novel algorithm for the anchored k-core problem. `RCM` selects anchors based on two measures – *Anchor Score* and *Residual Degree*. If the number of anchors needed to convert a connected component is more than the anchor budget available, the anchors are selected based on the anchor score. Otherwise, the anchor selection depends on the residual degree, and `RCM` selects the candidate anchors with the highest anchor scores.

6.1 Motivating Example

Example 1: Consider an online social friendship network (e.g., Facebook). It has been shown that users remains on such networks the activity of their friends [60] – if they have enough friends active on the social network, they are also likely to remain active. If we assume that a user remains active if at least k friends are also active in the social network, the k-core forms the sub-graph of the users who are active on the network. Therefore, it of interest to the owner to the operator of the social network to maximize the size of the k-core in such network.

Example 2: In many online multiplayer games, users need to group up to attempt the high level quests. If k is the number of people required to attempt these quests, the users who already have at least k active friends have a better experience because they can invite these friends to these quests. On, the other hand those who less than k active friends have to use the 'looking for group' feature and are grouped with random people. That is the players who are in the k-core of the friendship network have a better experience and are likely to stay active. Therefore, it is of interest to maximize the size of the k-core in this friendship network.

In these examples, the people/users/players who are provided an incentive to remain active are the anchors, and the people who are in the anchored k-core as a result are the followers.

6.2 Anchored k-Core Problem

The *anchored k-core* problem was introduced by Bhawalkar *et al.* in 2012 [14]. The problem was inspired by the observation that a user in a social network is motivated to stay only if her neighborhood meets some minimal level of engagement: in k-core terms, she will stay if k friends are also in the network. Bhawalkar *et al.* defined the anchored k-core as the subgraph that is computed using the usual k-core decomposition algorithm, but with the modification that selected 'anchor' nodes are not deleted during the process. These anchored nodes may represent, for example, nodes that are recruited to remain active in the network, even if their friends are inactive. The anchored k-core problem, then, is the problem of selecting a specified number b anchor nodes such that the number of nodes in the anchored k-core is maximized. Bhawalkar *et al.* showed that for a general graph the anchored k-core problem is solvable in polynomial time for $k \leq 2$, but is NP-hard for $k > 2$ [15]. They also showed that the problem is W[2]-hard with respect to the number of anchors and Chitnis *et al.* showed that the problem is W[1]-hard with respect to the number of nodes in the anchored k-core [26].

Zhang *et al.* proposed a greedy algorithm, called `OLAK`, for the anchored k-core problem [98]. `OLAK` operates over b_{max} iterations, where b_{max} is the allowable number of anchor nodes. In each iteration, a node that is not in the anchored k-core but which would generate the largest number of followers if anchored is selected as the next anchor. Because only a single anchor node at a time is considered, and only nodes from the $(k-1)$-shell can become followers when anchoring a single node, `OLAK` considers only follower nodes from the anchored $(k-1)$-shell during each iteration.

6.3 Problem Definition

Consider a graph $G = \langle V, E \rangle$, and let $N(u)$ denote the set of neighbors of $u \in V$ in G. We use $\overline{G_k}$ to refer to the subgraph induced by $\overline{V_k} = V \setminus V_k$.

Notation	Description
A	The set of nodes that are anchored.
b	The anchor budget.
$V_{k,A}$	The nodes in the anchored k-core with anchors A.
$\overline{V_{k,A}}$	The nodes in V, but not in $V_{k,A}$.
$\mathcal{F}(A)$	The nodes in the anchored k-core, but not in k-core.
$N(u)$	The neighbors of node u in the graph G.
C_a	The set of candidate anchors.
C_f	The set of candidate followers.
$\delta(u, A)$	Residual degree of node u with anchors A.

Table 6.1: Notations used in this chapter.

Consider $A \subset \overline{V_k}$. The *anchored k-core* of G with anchors A is the maximal subgraph $G_{k,A} = \langle V_{k,A}, E_{k,A} \rangle$ such that $\forall u \in V_{k,A}$ one of the following holds:

(1) u is an anchor node, i.e., $u \in A$,

(2) u has at least k neighbors in $V_{k,A}$, i.e., $|N(u) \cap V_{k,A}| \geq k$.

The anchored k-core of a graph can be computed like the usual k-core – but with the nodes in A kept in the graph even if their degree is below k. In many applications, there is a bound on the number of anchor nodes. We denote this *anchor budget* by b. The '*followers*' are the non-anchor nodes that are not in the k-core but are in the anchored k-core, and are denoted by $\mathcal{F}(G, k, A)$, where

$$\mathcal{F}(G, k, A) = V_{k,A} \setminus (V_k \cup A).$$

For brevity, we will use $\mathcal{F}(A)$ when the G and k are clear from the context.

The anchored k-core problem was introduced in [15] as follows: *If we are given an anchor budget of b, which nodes should be anchored so that the number of followers is maximized?* Formally, the

objective is to find the set A^* such that

$$A^* = \underset{A \subseteq \left[\overline{V_k}\right]^b}{\arg\max} \left|\mathcal{F}\left(k, A\right)\right|$$

where $\left[\overline{V_k}\right]^b = \left\{X \subseteq \overline{V_k} : |X| = b\right\}$.

6.4 Need for Look-Ahead Ability

Before the current work, the previous state-of-the-art algorithm for the anchored k-core problem is `OLAK`, a purely greedy algorithm that, in each iteration, anchors the node that would add the most followers [98]. This method has been demonstrated to work well on many real-world networks. However, it suffers from certain limitations. Zhang *et al.* showed that with such a selection procedure, the considered followers can only come from the (anchored) $(k-1)$-shell (that is, the nodes in the $(k-1)$-core, but not in the k-core). Most importantly, as we show in Section 6.5.1, the set of all candidate followers is $C_f \subseteq \overline{V_{k,A}}$. Combining these two results, the candidate followers in a greedy method is,

$$C'_f = \left(\overline{V_{k-1,A}} \setminus \overline{V_{k-2,A}}\right) \cap C_f.$$

This means that there are two conditions for this type of purely greedy method to succeed:

1. If ratio $f_k = \frac{|C'_f|}{|C_f|}$ is large, then most of the followers comes from the $(k-1)$-shell. The greedy anchor selection algorithm will work well in this case.

2. Even if f_k is low, if the anchor budget is low enough that the maximum number of follower possible is close to or less than $|C'_f|$, purely greedy methods will work well.

The upper bound on the coreness of a node is its degree, and in most real-world networks the degree distribution follows a power-law distribution. So, the ratio f_k decreases rapidly as k increases

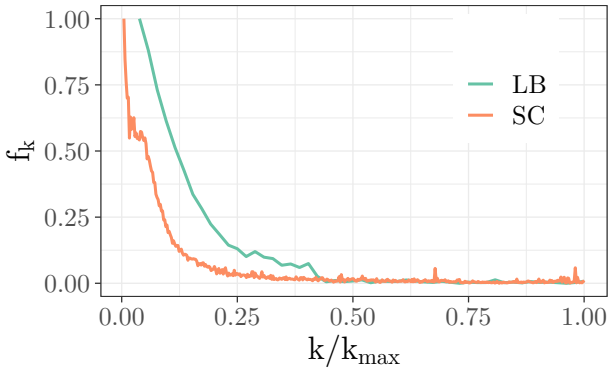


*Figure 6.2: Relation between f_k and k/k_{max} for different networks. f_k is the ratio of candidate followers that are in the $(k-1)$-shell to the total candidate followers. Note that the ratio decreases rapidly as k increases, indicating that a greedy approach that focuses on $(k-1)$-shells may not perform well. Here, `LB` and `SC` are different networks (described in Table **??**).*

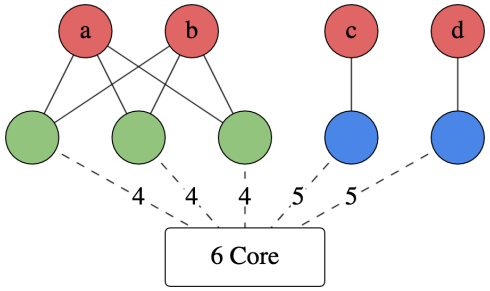


Figure 6.3: In this example, we seek to maximize the size of the anchored 6-core. The red nodes are the candidate anchors, the green nodes are in 4-shell and blue nodes are in 5-shell. The edges between the 6-core and the rest of the nodes are shown with dashed lines and the number represents the number of edges.

in most real-world networks. As an example, Figure 6.2 shows the value of f_k against $\frac{k}{k_{max}}$ for two real-world networks – LB and SC. (These networks are described in more detail in Table **??**.). The value of f_k drops very quickly – indicating that such algorithms will not be able to convert a huge fraction of the potential follower into actual followers, for most values of k.

To demonstrate the shortcoming with an example, consider the example in Figure 6.3. In this example, we seek to maximize the size of the anchored 6-core by anchoring 2 nodes. The red nodes are the candidate anchors, the green nodes are in the 4-shell, and the blue nodes are in the 5-shell. For visual clarity, the edges between the 6-core and the rest of the nodes are represented by dotted lines, and the number represents the number of edges. It is clear that a greedy approach will select

nodes c and d as anchors, resulting in 2 new followers. However, had a and b been anchored, there would have been 3 new followers.

6.5 Method: Residual Core Maximization

In this section we will describe the components that makes up *Residual Core Maximization* (`RCM`) and how they combine together to select the anchor nodes.

6.5.1 Candidate Followers and Anchors

We begin by deriving the necessary conditions for a node to be a *candidate follower* from the definition of k-core, and then use that to find the *candidate anchors*.

Consider the adjacency matrix $\mathbf{M}$ of $\overline{G_{k,A}}$, the subgraph of G left after removing the anchored k-core subgraph with anchors A. Assume that additional anchors $A' \subseteq \overline{V_{k,A}}$ are introduced, where $\overline{V_{k,A}}$ is the set of nodes in $\overline{G_{k,A}}$. Let δ be an element-wise function over $\mathbf{x}$ such that $\gamma(\mathbf{x}_v) = 1$ if $\mathbf{x}_v \geq 0$ and 0 otherwise.

Let $\mathbf{c}$, $\mathbf{a}$, $\mathbf{s}$ be vectors of length $|\overline{V_{k,A}}|$ such that:

$$\mathbf{c}_v = |N(v) \cap V_{k,A}|.$$

$$\mathbf{a}_v = \begin{cases} k & \text{if } v \in A' \\ 0 & \text{otherwise} \end{cases}.$$

$$\mathbf{s}_v = \begin{cases} 1 & \text{if } v \in \mathcal{F}(A \cup A') \setminus \mathcal{F}(A) \\ 0 & \text{otherwise} \end{cases}.$$

Then, by the definition of anchored k-core,

$$\mathbf{s} = \gamma\left(\mathbf{M}\mathbf{s} + \mathbf{a} + \mathbf{c} - k\mathbf{1}\right) = \gamma\left(\mathbf{M}\mathbf{s} + \mathbf{a} - \mathbf{r}\right). \tag{6.1}$$

From equation 6.1, it is easy to see that for a node $v \notin A'$, if $|N(v)| < k$, it is not possible to have $s_v = 1$. So, the *candidate followers* are given by,

$$C_f = \{v \in \overline{V_{k,A}} : |N(v)| \geq k\}.$$

Now consider a node v selected as an anchor. If $N(v) \cap C_f = \emptyset$, it is not possible for v to bring in new followers. So the set of *candidate anchors* is,

$$C_a = \{v \in \overline{V_{k,A}} : |N(v) \cap C_f| > 0\}.$$

We can, therefore, discard any nodes not in $C_f \cup C_a$, from the following analysis.

Theorem 6.1. *Nodes that are not in C_f cannot become followers. That is, $\forall v \in \left(\overline{V_{k,A}} \setminus C_f\right), \nexists A' \subseteq \overline{V_{k,A}}$ such that $v \in \mathcal{F}\left(k, A \cup A'\right)$.*

Proof. Since, $v \notin C_f$, $|N(v)| < k$. If $v \in \mathcal{F}\left(k, A \cup A'\right)$, by definition $|N(v) \cap V_{k,A\cup A'}| \geq k \implies |N(v)| \geq k$. This is a contradiction. So, $v \notin \mathcal{F}\left(k, A \cup A'\right)$. □

Theorem 6.2. *Adding any subset of $\overline{V_{k,A}} \setminus C_a$ to the set of anchors will not change the set of followers. That is, $\forall A' \subseteq \left(\overline{V_{k,A}} \setminus C_a\right), \mathcal{F}\left(k, A\right) = \mathcal{F}\left(k, A \cup A'\right)$.*

Proof. Consider $A' \subseteq \left(\overline{V_{k,A}} \setminus C_a\right)$. It is easy to show that $\mathcal{F}\left(k, A\right) \subseteq \mathcal{F}\left(k, A \cup A'\right)$. So,

$$\mathcal{F}\left(k, A\right) \setminus \mathcal{F}\left(k, A \cup A'\right) = \emptyset. \tag{6.2}$$

Let $D = \mathcal{F}(k, A \cup A') \setminus \mathcal{F}(k, A)$. By Theorem 6.1, $D \subseteq C_f \subseteq \overline{V_{k,A}}$. Then by the definition of anchored k-core, $\forall v \in D$,

$$|N(v) \cap V_{k,A \cup A'}| \geq k$$
$$|N(v) \cap (V_{k,A} \cup A' \cup D)| \geq k$$
$$|N(v) \cap (V_{k,A} \cup D)| + |N(v) \cap A'| \geq k.$$

Because $A' \cap C_a = \emptyset, \nexists u \in A'$ such that $u \in N(v)$ (by definition of C_a). So, $|N(v) \cap A'| = 0$. Then,

$$|N(v) \cap (V_{k,A} \cup D)| \geq k.$$

This means that $V_{k,A} \cup D$ is the set of nodes in the anchored k-core with anchors A, because by definition, the anchored k-core is the maximal set. So all the nodes that are in the anchored k-core with anchors $A \cup A'$ are already in the set $V_{k,A}$. Then, $D = \emptyset$.

$$\mathcal{F}(k, A \cup A') \setminus \mathcal{F}(k, A) = D = \emptyset. \tag{6.3}$$

Therefore, from (6.2) and (6.3), we get $\mathcal{F}(k, A) = \mathcal{F}(k, A \cup A')$. □

6.5.2 Residual Degree

In equation 6.1, for $v_f \in C_f$, if there are $\mathbf{r}_v$ additional neighbors in $A' \cup \mathcal{F}(A \cup A')$ due to the anchors, v_f will also become a new follower. Intuitively, $\mathbf{r}_v$ tells us how 'far' $v \in C_f$ is from becoming a follower – nodes with lower value can be converted to new followers more easily. In the rest of the discussion we refer to this value as the *Residual Degree* and denote it with $\delta(v|A) = k - |N(v) \cap V_{k,A}|$.

6.5.3 Residual Core

When nodes $A' \subseteq C_a$ are added to A as anchors, which nodes in C_f become followers? To answer this we define the *Residual Core* subgraph. The residual core subgraph (with respect to the new anchors A') is defined as the maximal subgraph such that every node in the subgraph has at least as many neighbors in the subgraph or A' as its residual degree. We denote the residual core of A' with $R^*_{A'}$.

$R^*_{A'}$ gives us all the new followers due to A' (Theorem 6.3), and it can be found efficiently as described in Algorithm 5.

Algorithm 5 The algorithm to find residual core.

```
1:  function FindResidualCore(G, C_f, A')
2:      G_f ← Subgraph of G induced by C_f
3:      𝒢 ← Connected components in G_f
4:      𝒢 ← {S ∈ 𝒢 : (∃v ∈ S : N(v) ∩ A' ≠ ∅)}
5:      X ← Nodes in all the subgraphs in 𝒢
6:      while Y ≠ ∅ do
7:          Y ← {v ∈ X : |N(v) ∩ (X ∪ A')| < δ(v)}
8:          X ← X \ Y
9:      end while
10:     return X
11: end function
```

Theorem 6.3. $\mathcal{F}(A \cup A') \setminus \mathcal{F}(A) = R^*_{A'}$.

Proof. Let $Y = \mathcal{F}(A \cup A') \setminus \mathcal{F}(A)$. Consider $v \in Y$. Then, $|V_{k,A\cup A'} \cap N(v)| \geq k$ and $|V_{k,A} \cap N(v)| < k$. We know that, $V_{k,A\cup A'} = V_{k,A} \cup A' \cup Y$, where $V_{k,A}$ and $A' \cup Y$ are mutually exclusive by definition. So $\forall v \in Y$,

$$|V_{k,A\cup A'} \cap N(v)| \geq k$$

$$|(V_{k,A} \cup A' \cup Y) \cap N(v)| \geq k$$

$$|((A' \cup Y) \cap N(v))| \geq \delta(v|A).$$

By definition of residual core, we can now see that Y is the residual core with anchors A'.

Therefore, $\mathcal{F}(A \cup A') \setminus \mathcal{F}(A) = R^*_{A'}$. □

Theorem 6.4. *Algorithm 5 correctly finds the residual core.*

Proof. Let $V_{\delta,A}$ be the correct residual core, and S be the set returned by Algorithm 5.

By construction it is easy to verify that, $S \subset \overline{V_{k,A}}$ and $S = \{v \in V_{\delta,A} : |N(v) \cap S| < \delta(v, A)\}$. So, $S \in \mathcal{V}_{\delta,A}$.

Since $\delta(v, A)$ is defined only for $v \in C_f$, $V_{\delta,A} \subseteq C_f$. □

6.5.4 Bounds on the Number of Anchors

Let G_f be the graph induced from G by the nodes in C_f, and let $\mathcal{G}$ be the set of connected components in G_f. If nodes in $G' \in \mathcal{G}$ become followers, they cannot effect the residual degree of nodes in other components. Thus, we consider each component separately.

Then for $G' \in \mathcal{G}$, let V'_i be the set of nodes in G' that can become follower without relying on nodes not in G', and let V'_o be the set of nodes in G' that need anchors not in G' become followers. That is,

$$V'_i = \{v \in V' : |N(v, G')| \geq \delta(v)\}$$
$$V'_o = V' \setminus V'_i$$

where V' is the set of nodes in subgraph G', and $N(v, G')$ is the set of neighbors of v in G'.

If anchors A' are selected such that all the nodes in V'_o become followers, $G\prime$ become a a residual core, and converts the remaining nodes, V'_i to followers as well (Theorem 6.3).

As an example, consider Figure 6.4. In this example, the G' is indicated by the rectangle, and the

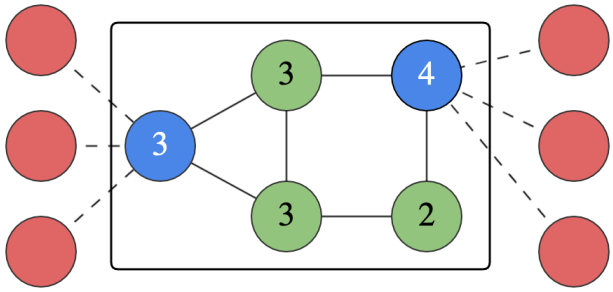


Figure 6.4: The nodes inside the box form G', and the number represents their residual degrees. The red nodes are the nodes in $C_a \setminus C_f$. The green nodes and blue nodes are V_i' and V_o' respectively.

numbers inside the nodes are the residual degrees of the nodes. The red nodes are nodes in C_a. We can see that the green nodes have at least $\delta(*)$ neighbors within G'; but the blue nodes need anchors from among the red nodes. So, the green and blue nodes form V_i' and V_o', respectively. It is easy to see that if the blue nodes are converted to followers, the G' becomes a residual core.

By construction, the only neighbors of V_o' not in G' are in $C_a \setminus C_f$. It can be seen that each node $v \in V_o'$ needs $\delta(v) - |N(v, G')|$ anchors from $C_a \setminus C_f$ to become followers. We denote it by $\delta'(v)$. Then consider,

$$\beta^{\perp}(G') = \max_{v \in V_o'} \delta'(v)$$

$$\beta^{\top}(G') = \sum_{v \in V_o'} \delta'(v)$$

$$\beta^{*}(G') = \min_{v \in V_o'} \delta'(v)$$

If we want to convert all nodes in G' to followers, we need at least $\beta^{\perp}(G')$ anchors from $C_a \setminus C_f$. So, this gives us the lower bound on the number of anchors required.

Now consider the case where none of the nodes in V_o' have any common anchor. In this case all nodes need to be anchored separately. Then, $\beta^{\top}(G')$ give us an upper bound on the number of anchors required.

If the remaining anchor budget s b', we have:

1. $b' \geq \beta^{\top}(G')$. All the nodes in C' can be converted to followers.
2. $\beta^{\perp}(G') \leq b' < \beta^{\top}(G')$. In this case, the budget may or may not be enough to convert all the nodes in C' to followers.
3. $b' < \beta^{\perp}(G')$. The budget is not enough to convert all the nodes in G' to followers. But it might be possible for some nodes to become followers.
4. $b' < \beta^{*}(G')$. None of the nodes in G' can become a follower.

For a given component, depending on these case, we need different anchor selection strategies.

6.5.5 Residual Anchor Selection

If the anchor budget remaining is enough to convert all nodes in G' to followers, we need to select the minimum number of anchors needed. Since the nodes in V_i' already have enough neighbors in G', it is enough to consider only V_o'.

We thus need to select the minimum number of anchors from $C_a \setminus C_f$ such that each node $v \in V_o'$ is connected to at least $\delta'(v)$ anchors.

Formally, we have a mapping $\delta' : V_o' \twoheadrightarrow \mathbb{Z}_{\geq}$, and a bipartite graph $G_o = \langle V_o', C_a \setminus C_f, E_o' \rangle$ where E_o' is the set of edges between V_o' and $C_a \setminus C_f$. The problem is to find the set A' such that,

$$S = \left\{ \hat{A} \subseteq C_a \setminus C_f : \forall v \in V_o', |N(v, G_o) \cap \hat{A}| \geq \delta'(v) \right\}$$
$$A' = \arg\min_{X \in S} |X|.$$

Finding the minimum number of residual anchors is NP-hard and so we propose a heuristic algorithm for this task (Algorithm 6). At each step, the algorithm selects the node from $C_a \setminus (C_f \cup A')$ that has the most neighbors in T, and adds it to A'. Here T is the set of nodes such that all the

nodes in T still requires additional anchors to become followers.

Algorithm 6 Algorithm to find the residual anchors of a connected component.

```
1:  function ResidualAnchors
2:      A' ← ∅
3:      T ← V'_o
4:      while T ≠ ∅ do
5:          v ← argmax_{u ∈ C_a \ (C_f ∪ A')} |N(u) ∩ T|
6:          A' ← A' ∪ {v}
7:          T ← {u ∈ T : δ'(u, G') > |N(u) ∩ A'|}
8:      end while
9:      return {(A', V')}
10: end function
```

Theorem 6.5. *Residual anchor selection in NP-hard.*

Proof. We will show this by reducing the set cover problem to the residual anchor selection problem. Suppose we have a set cover problem with finite sets $U \subseteq \mathbb{Z}_+$ and $S = \{S_0, S_1, ...\}$ such that $S_i \subseteq U$.. The set cover problem is to find the set S^* such that,

$$S^* = \arg\min_{S' \subseteq \overline{S}} |S'|$$
$$\text{s.t.} \bigcup_{X \in S'} X = U$$

Let us generate the following,

$$R = \{0, 1, \dots, |S| - 1\}$$
$$E = \{(i, j) : i \in U \wedge i \in S_j\}.$$

Now we can construct a bipartite graph $B = \langle U, R, E \rangle$. By construct, there is a one-to-one mapping between R and S. So, $(i, j) \in E$ denotes the membership of $i \in U$ to $S_j \in S$. So, with this this

construction, the set cover problem can be stated as: find R^* such that,

$$R^* = \arg\min_{R' \subseteq R} |R'|$$
$$\text{s.t.} \bigcup_{r \in R^*} N(r, B) = U.$$

If we have $\delta' : U \twoheadrightarrow 1$, the problem has reduced to the residual anchor selection, where U and R correspond to V'_o and $C_a \setminus C_f$. So the residual anchor selection problem in NP-hard. □

Theorem 6.6. *Algorithm 6 gives a solution that is within a factor $h_{|V'_o|}$ of the optimal solution where, $h_{|V'_o|} = \sum_{i=1}^{|V'_o|} \frac{1}{i}$.*

That is, if the solution found is A' and the optimal is A^*, $\frac{|A'|}{|A^*|} = h_{|V'_o|}$.

Proof. We need to show two things: (1) Algorithm 6 gives a valid Residual Anchor, and (2) the solution is at most h_l times the size of the optimal.

It follows directly form Theorem 6.3, that Algorithm 6 gives a valid Residual Anchor.

Now to prove the second part, we will show that the problem reduces to the set multi-cover problem.

Consider the set R such that for all $v \in C_a \setminus C_f$, the set of neighbors of v in V'_o is in R.

$$R = \{N(v) \cap V'_o : v \in C_a \setminus C_f\}.$$

Then the problem of finding Residual Anchors reduces to finding $S \subseteq R$ such that for all $v \in V'_o$,

$$|\{T \in S : v \in T\}| \geq \delta'(v).$$

By construction, $|S|$ is equal to the number of the Residual Anchors. So, the problem is equivalent to

finding the set S with minimum cardinality. This is a generalization of the set cover problem called the set multi-cover problem, and it can be solved by the greedy algorithm with h_l-approximation [92], where,

$$h_l = \sum_{i=1}^{|V_o'|} \frac{1}{i}.$$

□

6.5.6 Anchor Score based Anchors Selection

If the anchor budget is not enough to convert all the nodes in G' to followers, we want to convert as many as possible. To quantify the quality of a candidate anchor node with respect to maximizing the number of followers we propose a node-level measure called the *Anchor Score*. Denote all the nodes in G' by C_f', and consider C_a' such that $C_a' = \{v \in C_a : N(v) \cap C_f' \neq \emptyset\}$.

Then, we define the Anchor Score of $v \in C_f' \cup C_a'$ as

$$\alpha(v) \stackrel{\text{def}}{=} 1 + \sum_{u \in C_f \cap N(v)} \frac{\alpha(u)}{\delta(u)}. \tag{6.4}$$

The intuition is that nodes that are connected to others with high anchor score and low residual degree are important themselves. If nodes with high anchor scores are anchored, this helps in converting its neighbors into followers, which may themselves also be important.

To calculate the anchor scores of all nodes in $C_f' \cup C_a'$, we have $|C_f' \cup C_a'|$ equations:

$$\mathbf{q} = \mathbf{1} + \mathbf{D}\mathbf{q}, \tag{6.5}$$

where $\mathbf{q}$ is the vector of anchor scores, $\mathbf{1}$ is a vector of 1's, and $\mathbf{D}$ is a matrix such that $\mathbf{D}_{i,j} = \frac{1}{\delta(j)}$ if edge (i, j) exist, otherwise 0.

Depending on the membership of a node in C'_a and/or C'_f, we have the following conditions:

1. $v \in C'_f \setminus C'_a$. Since $C'_f \cap N(v) = \emptyset$ by definition, $\alpha(v) = 1$.
2. $v \in C'_f \cap C'_a$. In this case, $\alpha(v)$ appears on both sides of equation 6.5.
3. $v \in C'_a \setminus C'_f$. Here, v cannot appear on the right of the equation. So, $\alpha(v)$ is simple to calculate once the other two cases have been calculated.

To compute anchor scores, we first set the score for $C'_f \setminus C'_a$ to 1. We next restrict computation of Equation 6.5 to only the nodes in $C'_f \cap C'_a$, and calculate the anchor scores. Finally, we calculate the anchor scores of $C'_a \setminus C'_f$ using Equation 6.4 and the previously calculated anchor scores.

After calculating the anchor scores, the node with the highest value is selected as the next anchor. The process repeats as long as there is budget left. Algorithm 7 describes this process.

Algorithm 7 Algorithm to find the anchors based on anchor score.

```
1:  function ASAnchors
2:      A', F', 𝕊 ← ∅, ∅, ∅
3:      while |A'| < b do
4:          Calculate the Anchor Scores α(∗)
5:          v ← arg max_{u ∈ C'_f ∪ C'_a} α(u)
6:          R ← FindResidualCore(A' ∪ {u})
7:          A' ← A' ∪ {v}
8:          F' ← F' ∪ R
9:          𝕊 ← 𝕊 ∪ {(A', F')}
10:         Remove R and v from C'_a and C'_f
11:         Update δ(∗)
12:     end while
13:     return 𝕊
14: end function
```

6.5.7 Residual Core Maximization

In this section, we put together the pieces of our proposed algorithm *Residual Core Maximization* (`RCM`). The main idea of `RCM` is to divide the graph into multiple connected components of C_f, and

then to find anchors for these subgraphs separately depending on $\beta^{\top}(G')$ (Section 6.5.4). Algorithm 8 describes RCM in detail.

The first step is to generate $\mathcal{G}$, the connected components of the subgraph induced with C_f. RCM then generates the (anchors, followers) tuples for the components, denoted by $\mathbb{S}$. This step can be performed in parallel. Next, the problem comes down to finding a set A such that,

$$\hat{\mathbb{S}} = \left\{ \mathbb{S}' \subseteq \mathbb{S} : \left| \bigcup_{S \in \mathbb{S}'} S[0] \right| \leq b \right\}$$
$$\mathbb{S}^* = \arg\max_{\mathbb{S} \in \hat{\mathbb{S}}} \left| \bigcup_{S \in \mathbb{S}} S[1] \right|,$$

where $S[i]$ denotes the i-the element in the tuple S. This problem is close to the set union knapsack problem.[1] So, we use a greedy algorithm that selects $S^* \in \mathbb{S}$ that maximizes $\frac{|S^*[1] \setminus F|}{|S^*[0] \setminus A|}$, where A and F are the sets of anchors selected so far and the followers as a result. This is described in Algorithm 9.

After S^* (or the approximation) is computed, RCM selects anchors as,

$$A = \bigcup_{S \in \mathbb{S}^*} S[0].$$

6.6 Running Time of RCM

In this section we will discuss the running time of RCM. We begin by discussing the running time of the various components described so far.

Selecting Candidate Anchors: Selection of candidate anchors requires only counting the neigh-

[1] The set union knapsack problem is a generalization of the knapsack problem in which the weight is calculated based on union of sets rather than sum of numbers [42]. In our problem, the value is also calculated based on set unions.

Algorithm 8 The Residual Core Maximization algorithm.

function ResidualCoreMaximization
  $A, \mathbb{S} \leftarrow \emptyset, \emptyset$
  Find C_a, C_f and calculate $\delta(*)$
  $\mathcal{G} \leftarrow$ Connected components in G_f
  for G' in $\mathcal{G}$ **do**
    if $\beta^*(G') > b$ **then**
      continue
    else if $\beta^{\perp}(G') > b$ **then**
      $\mathbb{S} \leftarrow \mathbb{S} \cup$ `ASAnchors(`G'`)`
    else if $\beta^{\perp}(G') \leq b$ **then**
      $\mathbb{S} \leftarrow \mathbb{S} \cup$ `ResidualAnchors(`G'`)`
    else
      $\mathbb{S} \leftarrow \mathbb{S} \cup$ `ResidualAnchors(`G'`)`
      $\mathbb{S} \leftarrow \mathbb{S} \cup$ `ASAnchors(`G'`)`
    end if
  end for
  $A \leftarrow$`SolutionSelection`$(\mathbb{S}, b)$
  return A
end function

Algorithm 9 The algorithm for solution selection in Residual Core Maximization.

function SolutionSelection
  $A, F \leftarrow \emptyset, \emptyset$
  while $|A| < b$ **do**
    $S^* \leftarrow \arg\max_{S \in \mathbb{S}} \frac{|S[1] \setminus F|}{|S[0] \setminus A|}$
    $\mathbb{S}$`.remove(`S^*`)`
    if $|A \cup S^*[0]| \leq b$ **then**
      $A \leftarrow A \cup S^*[0]$
      $F \leftarrow F \cup S^*[1]$
    end if
  end while
  return A
end function

bors of nodes in $\overline{V_{k,A}}$. So, C_f and C_a can be found in $O\left(|\overline{V_{k,A}}|\right)$.

Residual Degree: To find the residual degree, we need to count neighbors of all the nodes in C_f. This can be done in $O\left(|C_f|\right)$.

Connected Components: The connected components of G_f can be found in $O\left(|E_f|\right)$, where E_f is the set of edges in G_f.

Bound on Number of Anchors: For a component $G' \in \mathcal{G}$, we first need to find the set of nodes V'_o and V'_i. This requires only counting the number of neighbors of the nodes in G'. So, it can be done in $O\left(|V'|\right)$. Then we need to count the neighbors of V'_o to find $\beta^{\top}\left(G'\right)$, $\beta^{\perp}\left(G'\right)$ and $\beta^{*}\left(G'\right)$. The running time of this step is $O\left(|V'_o|\right)$. Then, the overall running time for the component G' is $O\left(V'\right)$. Since we need to find the bounds for all the components, the total running time is $O\left(|C_f|\right)$.

Residual Anchors: In Algorithm 2 (main paper), we need to check for anchors in $\left(C_a \setminus C_f\right) \cap N\left(V'_o\right)$. The number of iterations in the algorithm is of the order of $|V'_o|$ and $\left|\left(C_a \setminus C_f\right) \cap N\left(V'_o\right)\right| \leq \beta^{\top}\left(G'\right)$. So, the running time for component G' is $O\left(\beta^{\top}\left(G'\right)|V'_o|\right)$. Assuming that we need to find the residual anchors for all the components, the running time is $O\left(\sum_{G' \in \mathcal{G}} \beta^{\top}\left(G'\right)|V'_o|\right) \approx O\left(|C_f|\right)$.

Anchor Score based Anchors: For a component G', to find the Anchor Score of all the nodes in $C'_f \cup C'_a$. This can be done in $O\left(|E'_{fa}|\right)$, where E'_{fa} is the set of anchors in the induced subgraph of $C'_f \cup C'_a$. We then need to find the followers of the selected anchor with `FindResidualCore()` and this takes $O\left(|C'_f|\right)$. Then, if we consider all the components, the time to find b anchors is $O\left(b \cdot \left(|E_{fa}| + |C_a|\right)\right) \approx O\left(b \cdot |E_{fa}|\right)$, where E_{fa} is the set of edges in the induced subgraph of $C_f \cup C_a$.

Overall Running Time: By combining the running time of all different parts, we can get the overall

running time of RCM as,

$$O\left(|\overline{V_{k,A}}| + |C_f| + |E_f| + |C_f| + |E_{fa}|\right) \approx O\left(|E_{fa}|\right).$$

6.7 Experiments

We evaluate the performance of RCM against various baselines both in finding followers and efficiency in doing that. We also compare to the optimal algorithm described by Bhawalkar *et al.* [14] for $k = 2$.

Table B.1 lists the real-world networks used in our experiments. These datasets are available at Network Repository[2] and SNAP.[3] We consider social, web, and collaboration networks of various sizes, ranging from a few thousands to more 1 million edges. We treat all graphs as undirected.

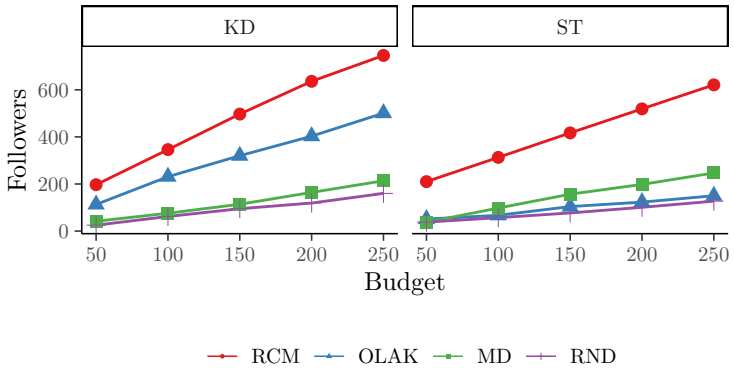
All experiments are performed on a 2.3 GHz 8-core machine with 128GB of RAM that runs Ubuntu 18.04. Algorithms are implemented in Python 3.5.2. Unless otherwise stated, we use only the sequential version of RCM in the following discussion and results.

6.7.1 Comparison Against Baseline Algorithms

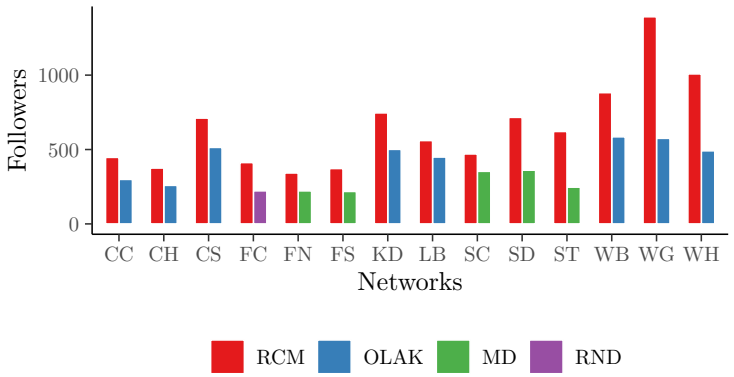
We consider three baseline algorithms for finding anchor nodes. The first is OLAK, the current state-of-the-art algorithm for anchor nodes selection [98]. OLAK greedily selects one anchor node at a time, and recomputes the anchored k-core decomposition in each step. OLAK has been demonstrated to work well on a number of real-world networks. For fair running time comparison, we implement OLAK in Python.

The second baseline is *Maximum Degree* (MD). This algorithm selects a node from C_a that has the

[2] http://networkrepository.com
[3] https://snap.stanford.edu/data/index.html



(a) Followers vs Budget.



(b) Followers at $b = 250$.

*Figure 6.5: Number of followers found by **RCM** and various baselines (at k fixed at the median value). In Figure 6.5a, the number of followers against the budget is shown for some selected networks. In 6.5b, the number of followers at $b = 250$ for all the networks considered is shown. Only **RCM** and the best baseline is shown. We can see that **RCM** selects the anchors that result in the largest number of followers in all cases. (**Higher values are better.**)*

maximum number of neighbors in C_f as anchor. The third baseline is *Random* (`RND`), which selects anchors randomly from C_f. In all baselines, after an anchor node has been selected, the new anchor and followers are removed from C_a and C_f.

We set k to the median core number of the network (given in Table **??**) and vary the anchor budget from 50 to 250 in increments of 50.

Figure 6.5a shows the number of followers for varying budgets for some selected networks and Figure 6.5b shows the followers at $b = 250$ for `RCM` and the best baseline on all networks. `RCM`, shown

in red, clearly outperforms all the baselines. As expected, the results are closer to OLAK for lower budgets, but the difference increases for higher budgets. Among the baselines, no single algorithm is always the best. The results for all baselines are in the supplementary material. We also perform experiments with $b = 100$ and various k. The results for this experiment are in the supplementary material. We observe that RCM outperform the baselines in all the cases considered.

Comparison of Time to find Followers: To compare the runtime efficiency of the various algorithms, we consider the time to find each follower. Figure 6.6a shows the time to to find a follower against the budget and Figure 6.6b shows the result for RCM and the best baseline[4] for all the network at $b = 250$. In all the cases RCM is much faster than all the baselines. Note that in many algorithms, the average time to find a follower drops as the budget increases because the size of C_a and C_f drops (as nodes become followers and anchors).

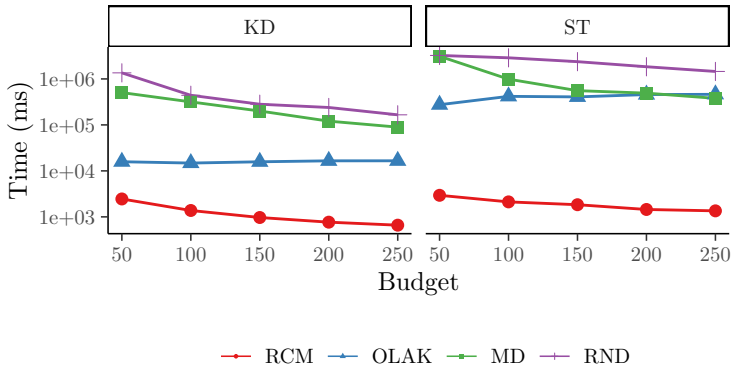
6.7.2 Comparison with Optimal Solution

In this section, we compare the performance of RCM against the optimal solution. Bhawalkar *et al.* [14] proposed an algorithm for finding the optimal solution for $k \leq 2$. We also include OLAK in the comparison. For these experiments we consider $k = 2$ and $b = 50$.

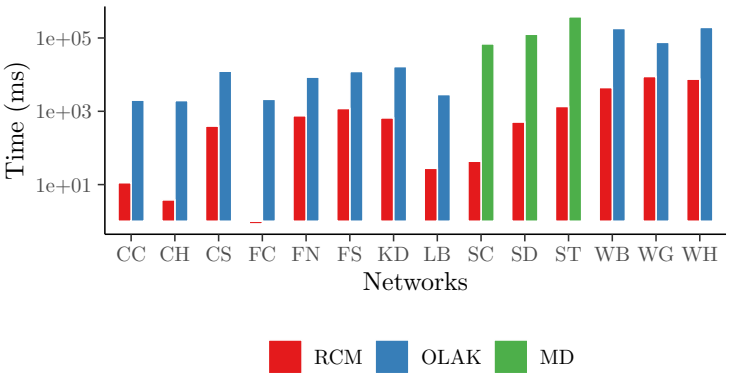
We also perform experimental comparison for $k > 2$. In this case, there is no efficient algorithm for a general graph. So, the optimal algorithm in this case is exhaustive search over C_a. Because of this, we are limited to small budgets and $|C_a|$. For this case we consider the networks FC, FS and FN for $k = 3$ and $b = 10$. We denote the optimal solution by OPT.

Table 6.2 shows the comparison between RCM, OPT and OLAK. In all cases, the number of followers due to RCM is very close to that found by OPT. The followers due to OLAK are much fewer in all the networks. Additionally, RCM is around 100 times faster than OPT.

[4]Results for all the baselines are in the supplementary material.



(a) Time to find a follower vs budget.



(b) Time to find a follower at $b = 250$.

*Figure 6.6: Average time to find a follower by **RCM** and baselines. In Figure 6.6a, the the time at different budgets is given for selected networks, and in Figure 6.6b the time at $b = 250$ is shown for **RCM** and the best baseline. The value of k is given in Table B.1. **RCM** is much faster than the baselines in all the cases. (**Lower values are better.**)*

6.7.3 Experimental Analysis of RCM

In this section evaluate the various aspects of `RCM` – (a) the contribution of `AnchorScore()` and `ResidualAnchors()` to the overall performance, (b) the speedup due to parallelization, and (c) scalability with network size.

We evaluate the contribution of `ResidualAnchors()` and `ASAnchors()` by designing versions of `RCM` that use only one of them. We denote these as `RCM-RC` and `RCM-AS` respectively. Results are shown in Figure 6.7a. We observe that results are clearly better when we use both `ResidualCore()` and `ASAnchors()`. Additionally, `RCM-RC` outperforms `RCM-AS` in two out of the three networks.

Network	k	b	Alg.	Followers	Time (ms)
KD	2	50	RCM	114	1.2×10^2
			OPT	115	1.9×10^4
			OLAK	97	1.5×10^4
LG	2	50	RCM	150	3.5×10^2
			OPT	152	2.9×10^4
			OLAK	133	9.6×10^3
LB	2	50	RCM	160	9.0×10^1
			OPT	161	4.0×10^3
			OLAK	117	2.1×10^3
WG	2	50	RCM	180	3.9×10^3
			OPT	186	2.6×10^5
			OLAK	95	6.2×10^4
FC	3	10	RCM	9	1.7×10^1
			OPT	10	3.7×10^5
			OLAK	8	1.6×10^3
FS	3	10	RCM	8	4.6×10^1
			OPT	10	3.7×10^5
			OLAK	5	1.7×10^4
FN	3	10	RCM	10	3.2×10^1
			OPT	10	1.6×10^4
			OLAK	9	7.4×10^3

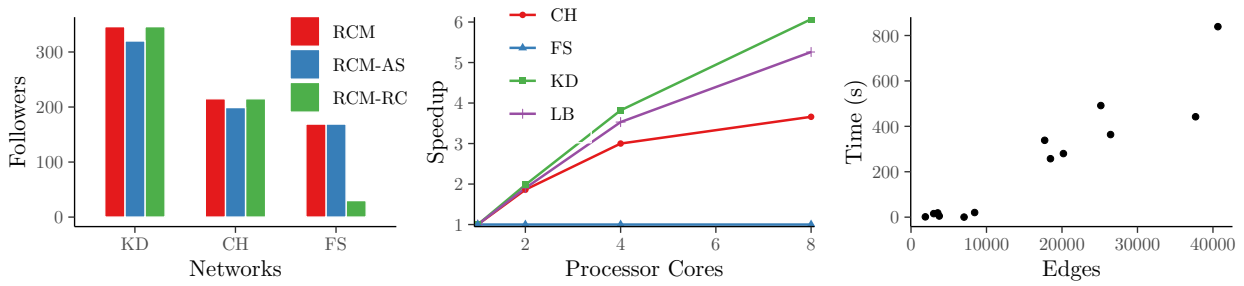
*Table 6.2: Comparison of **RCM**, **OPT** and **OLAK**. Observe that in all the cases, **RCM** is very close the **OPT** while being multiple magnitudes faster.*

RCM-AS outperforms RCM-RC in the network FS because $|\mathcal{G}| = 2$ and the budget is not enough to completely convert any component to followers.

To evaluate the speedup due to parallelization (Section 6.5.7), we limit the number of CPU cores available and compare the computation time.[5] Figure 6.7b shows the results of this experiment. In most networks RCM achieves significant speedup with CPU cores. However, in the case of FS network there is no speedup. This is because there are only two components – a large one and a very small one, making parallelization ineffective.

We evaluate the scalability of RCM with network size. As described in Section 6.6 the runtime of RCM is given by $O(|E_{fa}|)$, where E_{fa} is the set of edges in the subgraph induced by $C_f \cup C_a$. Figure 6.7c shows the running time of RCM against $|E_{fa}|$ for all the networks in Table **??**. As expected, the runtime

[5] The k value is given in Table B.1 and $b = 100$.



(a) Follower count due to ***RCM****,* ***RCM-RC*** *and* ***RCM-AS****.*

(b) Speed up due to parallel computation.

(c) Running time of ***RCM*** *against* $|E_{fa}|$*.*

Figure 6.7: Experimental results for analysis of ***RCM****. Figure 6.7a shows the contribution of different parts of* ***RCM****, Figure 6.7b shows the speedup due to parallel computation, and Figure 6.7c shows the running time against* $|E_{fa}|$*.*

is linear in $|E_{fa}|$.

6.8 Conclusions

We addressed the anchored k-core problem: given an anchor budget, what is the set of anchor nodes that should be selected to maximize the number of followers? We proposed a method, called *Residual Core Maximization* (RCM). Through extensive experimental analysis, we demonstrate that RCM performs significantly better than the state-of-the-art algorithms. On average, RCM finds 1.65 times the followers found by the best baseline method, while taking being 500 times faster. We also compared RCM against the optimal solution and observed that the number of followers found by RCM is very close to the optimal; and the time to find each follower is around 100 times faster.

Chapter 7

Skeletal Core Graph

In the preceding chapters, we studied the resilience of k-cores to various types of changes. The behavior of a graph to a certain type of change is dictated by various factors – including the number of 'extra' edges and the structure of the graph itself. For example, when we studied the core resilience Chapter 4, the number of extra edges is captured with core strength, and we found that it plays an important part in determining how resilient a graph is. Higher core strength generally translates to higher core resilience. Similarly, when we study the anchored k-core problem Chapter 6, we observed that there are some graphs in which it is easy to select anchors that has a lot of followers; and in some others the number of followers is very low. We know that extra edges does play a role here too – the residual degree is a measure of that. With regards to the collapsed k-core problem Chapter 5, we know that if there are very few nodes with relative core strength of 1, we are likely to find smaller core unstable graph. Thus, the number of 'extra' edges directly have an affect on the collapse resilience of a graph.

However, we also know that the graphs structure beyond these metrics plays a very important role. In the core resilience, this is captured with core influence – some nodes are more important than others based where it is located in the graph. Similarly, in anchored k-core problem, we know that the connected components in the induced subgraph of the candidate anchors is an important factor in determining if we can find anchors with a lot of followers or not. Lastly, we know that the size

of the core unstable graph is dependent on the graph structure.

So, to better understand the behavior of different graphs to these changes, it is important to understand purely the effect that structural organization of the different shells has. For example, does a graph in which there are a lot of connections between the different shells have higher resilience to core structural changes and why? Answer to such questions can help us in designing better algorithms to improve the resilience or estimate the resilience of a large graph.

So, in this chapter we introduce the idea of *Skeletal Core Graph*. We can think of the skeletal core graph of a graph as the minimal graph that has the same k-core structure but without all the extra edges. We consider two extreme cases of skeletal core graphs based on the connections between the shells and show how the resilience is affected. Given a graph, we also propose a way to quantify where its skeletal core lies within these two extreme cases.

We begin by describing the skeletal core graph and properties associated with it in Section 7.1. We describe the two extreme cases of skeletal core graph – *Centralized* and *Decentralized* Core Graphs. We propose *Core Centralized Score* which is a measure of where a skeletal core graph falls between these extreme cases. Then we describe how we describe the skeletal core sub-graph of a graph (Section 7.1.2, and given a graph, estimate where its skeletal core subgraph is likely to fall between the centralized and decentralized core graphs. In Section 7.2, relate the core structural change of a graph to its skeletal core subgraph; and in Section 7.3 we explain how the different structures of the skeletal core subgraph can help explain some of the observed behavior in the graph unraveling problem.

For a graph $G = \langle V, E \rangle$, we will use the notations described in Table 7.1.

Notation	Description
$G_k = \langle V_k, E_k \rangle$	The k-shell subgraph.
$E_{i,j}$	The edges between the i-shell and the j-shell.
$\kappa_G(v)$	The core number of node v in graph G.
k_G^*	The degeneracy of the graph G.
$\Gamma_G(v)$	The neighbors of v in graph G.
$\Gamma_G^k(v)$	The neighbors of $v$in $\kappa(v)$-core.

Table 7.1: Notations used in Chapter 7.

7.1 Skeletal Core Graph

We define a *skeletal core graph* as the graph $G^\sigma = \langle V^\sigma, E^\sigma \rangle$ such that for any $G' = \langle V, E' \rangle$, where $E' \subset E^\sigma$, $\nexists v \in V : \kappa_{G^\sigma}(v) \neq \kappa_{G'}(v)$. That is, it is the graph where any edge removal results in at least one node dropping its core number. Since the since the skeletal core graph does not have any 'extra' edges its behavior regarding the resilience to changes in the core structure is purely due to connections between the shells (and consequently within the shell).

Theorem 7.1 (Core Strength Condition for Skeletal Core Graph). *If $G^\sigma = \langle V^\sigma, E^\sigma \rangle$ is a skeletal core graph, there exists no edge (u, v) such that $CS(u) > 1$ and $CS(v) > 1$, where $CS(u)$ is the core strength of node u (Section 4.3.2).*

Proof. We can see that in any graph, if the core number of any node changes on removal of an edge (u, v), the core number of u and/or v should have also changed. That is, it is not possible for a node other than u, v to change core number but for both u and v to not change when edges (u, v) is deleted.

Assume that there exists an edges (u, v) in the skeletal core graph G^σ such that $CS_{G^\sigma}(u) > 1$ and $CS_{G^\sigma}(v) > 1$. Then, if we remove this edge to get graph $G' = \langle V^\sigma, E^\sigma \setminus \{(u, v)\} \rangle$, the core strength of u and v drops by at most 1. That is, $CS_{G'}(u) \geq 1$ and $CS_{G'}(v) \geq 1$. So, by definition of core strength, the neither u or v changes core number; and consequently there are no other nodes in the graph that changes core number due to the edge deletion.

This means that G^{σ} was not a skeletal core graph. Hence, proved by contradiction. □

Theorem 7.1 provide us a way to check if a given graph is a skeletal core graph efficiently. After the k-core decomposition, we need to calculate the core strength of all the nodes, and check if there are any edge where both endpoints have core strength greater than 1. Algorithm 10 describes this in more details.

Algorithm 10 Algorithm to check if a graph is a core skeletal graph or not.

```
1: function CheckSkeletalCore(G = ⟨V, E⟩)
2:     CS ← CoreStrength(G)
3:     for (u, v) ∈ E do
4:         if CS[u] > 1 ∧ CS[v] > 1 then
5:             return False
6:         end if
7:     end for
8:     return True
9: end function
```

Theorem 7.2 (Complexity of Algorithm 10)**.** *The time complexity of Algorithm 10 is $\mathcal{O}(|E|)$; and the space complexity is also $\mathcal{O}(|E|)$.*

Proof. The time complexity of calculating the core strength of all nodes in a graph is $\mathcal{O}(|E|)$. Then we need to check the core strength for all the edges. This can also be done in $\mathcal{O}(|E|)$. So, the running time of Algorithm 10 is $O(|E|)$.

We do not need to store the input graph while calculating the core strength of all the nodes – that is $\mathcal{O}(|E|)$. We need $\mathcal{O}(|V|)$ to store the core strengths of all the nodes. So, the space complexity of Algorithm 10 is also $\mathcal{O}(|E|)$. □

Theorem 7.3 (Correctness of Algorithm 10)**.** *Algorithm 10 always returns **True** for a valid skeletal core graph, and **False** otherwise.*

Proof. Theorem 7.1 returns **False** iff there exist an edge (u, v) such that $CS[u] > 1$ and $CS[v] > 1$. If there exists such an edge, we know from Theorem 7.1 that the graph is not a skeletal core graph.

Similarly we can show that Algorithm 10 returns **True** only if the graph is a skeletal core graph. □

7.1.1 Categorization of Skeletal Core Graphs

To better understand the effects of different type of changes to the core structure of a skeletal core graph, we need to categorize them into different types. We start by categorizing the edges based on the core numbers of its endpoints:

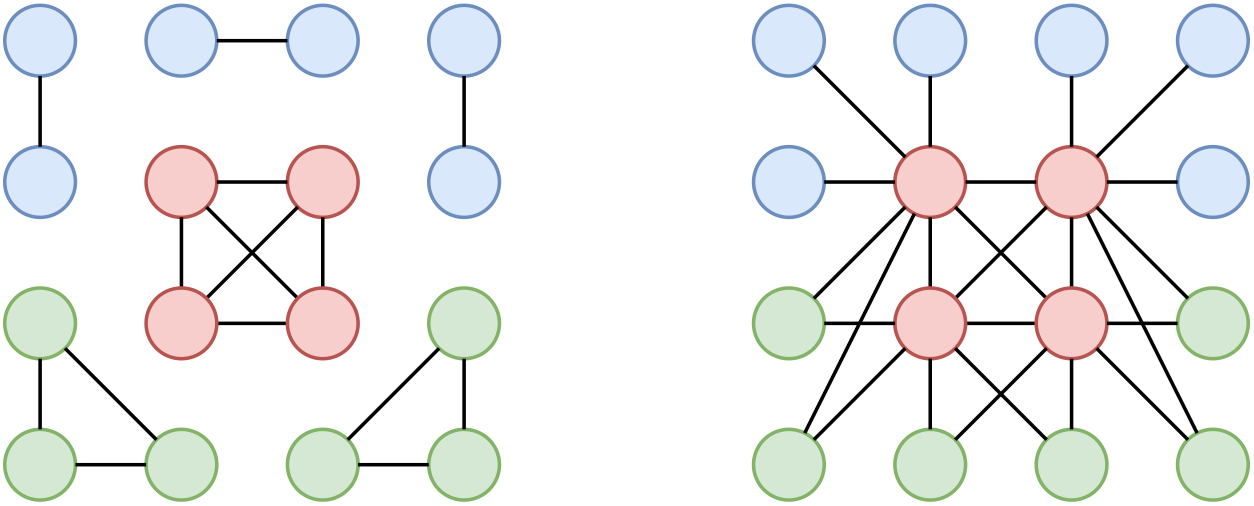
1. **Inter-Shell Edges:** These are the edges whose end vertices have the same core number.
2. **Intra-Shell Edges:** These are the edges whose end vertices have different core numbers.

Depending on the number of inter and intra shell edges, we have two extreme cases of skeletal core graphs. We call them centralized and decentralized skeletal core graphs.

1. **Decentralized Skeletal Core Graph**: There are the skeletal core graphs with no inter-shell edges.
2. **Centralized Skeletal Core Graph**: These are the skeletal core graphs with: (a) no intra-shell edges, except in the degeneracy core, and (b) all the inter-shell edges have one endpoint in the degeneracy core.

As an example consider the toy graphs shown in Figure 7.1. The color of the nodes indicates their core number – red is 3, green is 2 and blue is 1. We can see that both of the graphs are core skeletal graphs. In Figure 7.1a all the nodes connects only to another that have the same core number. So this is an example of a decentralized skeletal core graph. In Figure 7.1b, all the nodes connects to a node in the degeneracy core (red nodes). So, Figure 7.1b is an example of a centralized skeletal core graph.

In the rest of the discussion, we will used $G_D^\sigma = \langle V_D^\sigma, E_D^\sigma \rangle$ and $G_C^\sigma = \langle V_C^\sigma, E_C^\sigma \rangle$ to denote decentralized skeletal and centralized core graphs respectively.



(a) Decentralized Skeletal Core Graph *(b) Centralized Skeletal Core Graph*

Figure 7.1: Toy example showing Decentralized (Figure 7.1a and Centralized (Figure 7.1b) Skeletal Core Graphs. Here the red, green and blue nodes have core numbers of 3, 2 and 1 respectively. In Figure 7.1a, we can see that all the nodes connects to a node in the degeneracy core (red node). In Figure 7.1b all the nodes are connected to another one with the same core number.

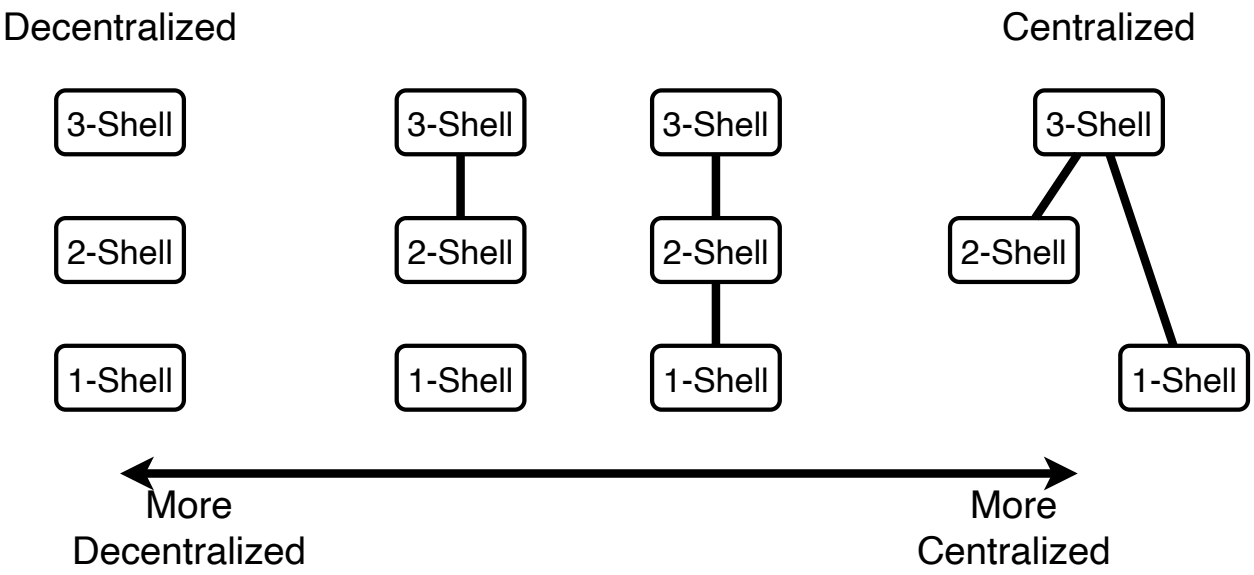


Figure 7.2: Different skeletal core graphs falls between centralized and decentralized core graphs.

Depending on the number of inter and intra-shell edges, all the core skeletal graphs will fall somewhere between decentralized and centralized core graphs Figure 7.2. To quantify where it falls within this range, we propose the *Centralized Score* measure.

The basic idea behind centralized score is that for a node u, the closer its neighbors in the $\kappa(u)$-core are to the degeneracy core, the more central the node is u. So, for a skeletal core graph $G^\sigma = \langle V^\sigma, E^\sigma \rangle$, we define the Centralized Score as,

$$CE(G) = \frac{1}{|V^\sigma \setminus V^\sigma_{k^*}|} \sum_{v \in V \setminus V_{k^*}} \frac{1}{|\Gamma^{\kappa(v)}(v)|} \sum_{u \in \Gamma^{k(v)}(v)} \frac{\kappa(u) - \kappa(v)}{k^* - \kappa(v)}. \tag{7.1}$$

Higher values of centralized score indicates that that graph is closer to a centralized skeletal core graph, and lower values indicates that it is closer to a de-centralized skeletal core graph. Decentralized skeletal core graphs have a centralized score of 0, and centralized skeletal core graphs have a centralized score of 1.

7.1.2 Skeletal Core Subgraph of a Graph

Given a graph $G = \langle V, E \rangle$, we can obtain a subgraph $G^\sigma = \langle V, E^\sigma \rangle$; $E^\sigma \subseteq E$ such that G^σ is a skeletal core graph. We call G^σ the skeletal core subgraph of G.

We can use Theorem 7.1 to find the edges to delete. At each step, all the edges that connects nodes with core strength greater than 1 are candidate for deletion. A random edge from these candidates is selected, and removed from the graph. Then, the core strengths are recomputed and the candidate sets are generated again. This continues until there are no more candidate edges to remove. This is described in Algorithm 11.

Theorem 7.4 (Complexity of Algorithm 11)**.** *The time complexity of Algorithm 11 is $\mathcal{O}(|E|)$; and the space complexity is also $\mathcal{O}(|E|)$.*

Algorithm 11 Algorithm to reduce a graph to its skeletal core subgraph

```
1: function SkeletalCoreDecomposition(G = ⟨V, E⟩)
2:     R ← ∅
3:     repeat
4:         CS ← CoreStrength(G)
5:         R ← {(u, v) ∈ E : CS[u] > 1 ∧ CS[v] > 1}
6:         X ← Random element of R
7:         E ← E \ {X}
8:     until R = ∅
9:     return G
10: end function
```

Proof. Computing the core strength of all the nodes for the first time can be done in $\mathcal{O}(|E|)$. For the subsequent steps, instead of recomputing it, we can simply calculate it for only those nodes involved in an edge deletion since we have the guarantee that the core number does not change due to the edge deletion.

Then, updating the core strength of a node can be done in constant time with proper data structure. The loop in Algorithm 11 repeats for at most $|E|$ times, and one edge deletion results in update of the core strength of two nodes. Inside each loop, the sets R and E can be found quickly through proper pruning.

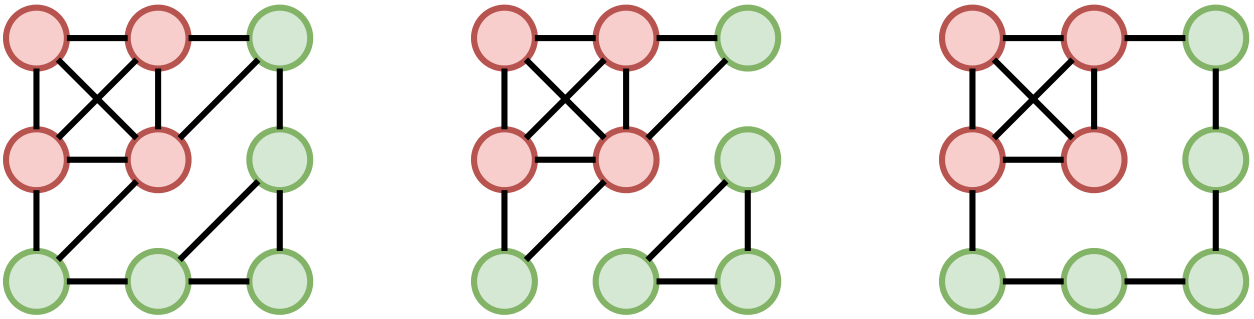
So, the overall running time of Algorithm 11 is $\mathcal{O}(|E|)$.

The space required to store the graph is $\mathcal{O}(|E|)$, the core strengths of all the nodes can be stored in $\mathcal{O}(|V|)$, and that for R is $\mathcal{O}(|E|)$.

So, the overall space complexity of Algorithm 11 is $\mathcal{O}(|E|)$. □

Theorem 7.5 (Correctness of Algorithm 11)**.** *Algorithm 11 correctly outputs a skeletal core subgraph of the input graph.*

Proof. We know that removing an edge (u, v) cannot change the core number of u or v if their core strength is greater than 1.



(a) (b) (c)

Figure 7.3: Example graph demonstrating the non-uniqueness of skeletal core subgraph.

We can see that Algorithm 11 exits the loop when $R = \emptyset$. So, by Theorem 7.1, if the loop terminates the output graph is a skeletal core graph. Because we are dealing with finite graph, it is not possible for the loop to not terminate.

So, Algorithm 11 correctly outputs the skeletal core subgraph of the input graph. □

Theorem 7.6 (Non-Uniqueness of Skeletal Core Subgraph)**.** *The skeletal core subgraph of a graph is not necessarily unique.*

Proof. Consider the graph shown in Figure 7.3a. The graphs shown in Figure 7.3b and Figure 7.3c are subgraphs, and both are skeletal core graphs. □

Core Centralized Score: To quantify how far a graph is from the centralized or decentralized skeletal core, we extend the concept of Core Centralized Score to a general graph. The core centralized score of a general graph is defined as the expected core centralized score of its skeletal core subgraphs.

For a graph $G = \langle V, E \rangle$, the likelihood of an edge (u, v) remaining in the skeletal core subgraph is dependent on the core number and number of neighbors in the same core of the node with lower

core number (both nodes if they have the same core number). That is,

$$p(u,v) = \begin{cases} \frac{\kappa(u)}{e(u)}\frac{\kappa(v)}{e(v)} & \text{if } \kappa(u) = \kappa(v) \\ \frac{\kappa(u)}{e(u)} & \text{if } \kappa(u) < \kappa(v) \\ \frac{\kappa(v)}{e(v)} & \text{if } \kappa(u) > \kappa(v) \end{cases} \tag{7.2}$$

where,

$$e(u) = |\Gamma^{\kappa(u)}(u)|. \tag{7.3}$$

For edge (u, v), $p((u, v), G')$ gives us a measure of how likely the edges are to be in the skeletal core. If $p((u, v), G') = 1$, the edge (u, v) has to be in all the skeletal core decomposed graphs of G'.

Then, we define the *Core Centralized Score* of graph G as,

$$CE(G) = \frac{1}{|V \setminus V_{k^*}|} \sum_{v \in V \setminus V_{k^*}} \frac{1}{|\Gamma^{\kappa(v)}(v)} \sum_{u \in \Gamma^{\kappa(v)}(v)} p(u,v) \frac{\kappa(u) - \kappa(v)}{\kappa^* - \kappa(v)}. \tag{7.4}$$

7.1.3 Generative Model for Random Skeletal Core Graph

When we study the k-core structure of a skeletal core graph, we ask if the observed behavior is expected for a random skeletal core graph with the same core number sequence of the nodes, or whether it is due to some other aspect of graph structure. To answer this, we need to compare the observed model with the null model – a random graph with the same core number sequence. There has been some works on generating graphs with a predefined k-core structure [12, 11]. However, these previous models needs the number of inter- and intra- shell edges in addition to the core number sequence – effectively restricting they type of graphs they can generate. So, we propose a

method to generate a random skeletal core graph with a given core number sequence.

Given a set of nodes V, assume that we have a mapping $c : V \to \mathbb{Z}_+$. We call a graph $G = \langle V, E \rangle$ is *valid* with respect to the mapping c, if $\forall v \in V, \kappa(v, G) = c(v)$. That is, suppose every node in V has an integer mapped with it. A graph is called *valid* with respect to this mapping, if the core number of all the nodes in the graph is equal to the integer that is mapped to it.

If there is a subset $V' \subset V$, such that there is a graph $G' = \langle V', E' \rangle$ where $\forall v \in V', \kappa(v, G') = c(v)$, we will refer to G' as *partially valid*. That is, if only a subset of nodes for which the core number and the integer mapped to it matches, we call it partially valid.

Now the problem is *given V and c, how can we generate a random graph $G = \langle V, E \rangle$ that is valid with respect to c.* There are some mappings c for which no valid graph exists. So, we start by considering the necessary conditions for c so that a valid graph can be generated.

Theorem 7.7 (Coreness Validity Constraint)**.** *For a given mapping c, a valid graph exists iff $\forall v \in V$, $|\{u \in V \setminus \{v\} : c(u) \geq c(v)\}| \geq c(v)$.*

That is, a valid graph can exist if and only if for every node $u \in V$, there are as many other nodes with same or greater core number than the core number of u.

Proof. We will show the proof in two steps: (1) if the coreness validity condition is not satisfied, there can be no valid graph, and (2) if the coreness validity condition is satisfied, there is always a valid graph.

Step 1: Assume that that the coreness validity condition is not satisfied. Then there exists at least one node $v \in V$ such that, $|\{u \in V \setminus \{v\} : c(u) \geq c(v)\}| < c(v)$.

Then, there are not enough nodes that v can connect to to obtain a core number of $c(v)$. That is, v cannot be in any graph valid with c. Therefore, it so not possible to obtain a valid graph if the coreness validity condition is not satisfied.

Step 2: In this step, we need to show that if the coreness validity condition is satisfied, a valid graph

exist. We will show this by induction. We begin by assuming that the coreness validity condition is satisfied.

Let $k^* = \max_{u \in V} c(u)$, and $V^* = \{u \in V : c(u) = k^*\}$. If we pick any node $v \in V^*$, by the coreness validity condition,

$$
\begin{aligned}
|\{u \in V \setminus \{v\} : c(u) = k^*\}| &\geq k^* \\
|\{v\} \cup \{u \in V \setminus \{v\} : c(u) = k^*\}| &\geq k^* + 1 \\
|V^*| &\geq k^* + 1.
\end{aligned}
$$

This means that we can construct a graph $G^* = \langle V^*, E^* \rangle$ such that every node is connected to k^* other nodes. So, G^* is partially valid. Therefore, *if the coreness validity condition holds, there is always a partially valid graph.*

Suppose that we have two nodes u, v in graph G' such that, $\kappa(u, G) < \kappa(v, G)$. By the definition of k-core, adding an edge (u, v) can never change the core number of v.

Now assume that there is a partially valid graph $G' = \langle V', E' \rangle$ such that $V^* \subseteq V'$. Consider a node $v \in V \setminus V'$, and add it to G' *without any edges*. Then, $\kappa(v, G') = 0$. Let,

$$
\begin{aligned}
S &= \{u \in V' : c(u) \geq c(v)\} \\
|S| &\geq k^* \\
|S| &> c(v).
\end{aligned}
$$

That is there are enough nodes in G' for v to connect in order to get a core number of $c(v)$. So, we can connect v to $c(v)$ other nodes in S, and the resulting graph is also partially valid.

If we keep repeating this process we will reach a point at which $V' = V$. So, if the coreness validity condition holds, a valid graph always exist.

Therefore from Step 1 and 2, Theorem 7.7 follows. □

In Step 2 of the proof for Theorem 7.7, we describe a method for constructing a k-core graph for a given distribution of core number. In the graph that is generated, if any edge is removed the core number of at least one node will change. So, it is a skeletal core graph.

Algorithm 12 describes the process of generating a random skeletal core graph from a given core number distribution.

Algorithm 12 Algorithm for generating a random skeletal core graph of given core number sequence.

function GENERATERANDOMSKELETALCOREGRAPH(c)
  if !$\mathrm{CorenessValidityCondition}(c)$ **then**
    return None
  end if
  $V \leftarrow \emptyset$
  $E \leftarrow \emptyset$
  $G \leftarrow \langle V, E \rangle$
  $k^* \leftarrow \max_{(i,j) \in c} j$
  while $k^* > 0$ **do**
    $S \leftarrow \{(i,j) \in c : j = k^*\}$
    $V \leftarrow V \cup \{i : (i,j) \in S\}$
    while $|S| > 0$ **do**
      $(i_0, j_0) \leftarrow$ Pop random element from S
      $N \leftarrow$ Select j_0 random element from $V \setminus \{i_0\}$
      $E \leftarrow E \cup \{(i_0, i_1) : i_i \in N\}$
      $S' \leftarrow \{(i_1, j_1) \in S : i_1 \in N\}$
      $S'' \leftarrow \{(i_1, j_1 - 1) \in S : j_1 - 1 > 0\}$
      $S \leftarrow (S \setminus S') \cup S''$
    end while
    $k^* \leftarrow k^* - 1$
  end while
end function
return G

Theorem 7.8 (Complexity of Algorithm 12)**.** *Both the time and space complexity of Algorithm 12 is linear with the number of nodes.*

Proof. We can see that checking the coreness validity constraint is linear with the number of nodes.

We can also see that the loops will execute for $\mathcal{O}(|V|)$. So, the running time of Algorithm 12 is $\mathcal{O}(|V|)$.

Similarly we can show that the space complexity of Algorithm 12 is $\mathcal{O}(|V|)$. □

Theorem 7.9 (Correctness of Algorithm 12)**.** *If $c : V \rightarrowtriangle \mathbb{Z}_+$ is desired the mapping from node id to core number, Algorithm 12 outputs a graph G where $\forall v \in V$, $c(v) = \kappa(v)$.*

Proof. We will divide the proof into two parts:

1. Show that all the nodes in the degeneracy core have a coreness of k^*.
2. Show that any node v, added after the degeneracy core has coreness of $c(v)$ and does not change the coreness of any previously added node.

In Algorithm 12, E is the set of edges. So, when we talk about degree we are referring to the degree w.r.t to the edges in E.

Part 1: Suppose for some $k < \max_{v \in V} c(v)$ and $V' = \{v \in V : c(j) > k\}$, we already have $G' = \langle V', E' \rangle$ such that $\forall v \in V', \kappa(v, G') = c(v)$.

Let $\overline{V} = \{v \in V : c(v) = k\}$. We need to show that after one iteration of the while loop (Steps 12-19) : (a) all nodes in $\overline{V}$ have a coreness of $c(v)$, and (b) no node in V' changed their coreness.

Be construction, $\forall v \in \overline{V}$, the node v gets connected to $c(v)$ nodes from $V' \cup \overline{V}$. So, all nodes in $V' \cup \overline{V}$ we be in the k-core after the while loop terminates. Again by construction when an edge (i_0, i_1) is added (Step 15), it is guaranteed that i_0 is not already in the k-core. So the coreness of no other can increase beyond k by this edge addition.

So, all nodes in $\overline{V}$ gets a coreness of k and the coreness of no node in V' changes after the while loop.

Part 2: We need to show that in the first iteration of the while loop (Steps 12-19), all nodes in $\overline{V} =$

$\{v \in V : c(v) = k^*\}$ gets a coreness of k^*.

In this case, $V' = \emptyset$, and like in Part 1, every node in $\overline{V}$ is guaranteed to have at least k^* neighbors in $\overline{V}$ by the end of the while loop. So, all nodes in $\overline{V}$ will be in the k^*-core.

Again, when an edge (i_0, i_1) is added, it is guaranteed that i_0 is not in the k^*-core already. So, this edge addition cannot increase the coreness of i_1 beyond k^*.

So, at the end of the while loop all nodes in $\overline{V}$ have a coreness of k^*.

From Part 1 and 2, by induction, we can see that Algorithm 12 outputs a graph G where $\forall v \in V$, $c(v) = \kappa(v)$. □

Theorem 7.10. *Let $G^* = \langle V^*, E^* \rangle$ be the degeneracy core of the skeletal core graph. Then the number of edges is,*

$$\left\lceil \frac{k^* \cdot |V^*|}{2} \right\rceil \leq |E^*| \leq \left\lceil \frac{k^* \cdot |V^*|}{2} \right\rceil + 1.$$

Proof. Recall that in a skeletal core graph, there are no edges (u, v) where the core strength of both u and v are greater than 1.

Let us consider the two sets:

$$V^\top = \{v \in V^* : CS(v) > 1\}$$
$$V^\perp = V^* \setminus V^\top.$$

For simplicity we consider the case where $k^* \cdot |V^*|$ is even. We consider two boundary cases: (1) $V^\top = \emptyset$, and (2) $\arg\max\ |V^\top|$.

Case 1: When $V^\top = \emptyset$.

In this case, all the nodes are in $V^\perp$, i.e. all nodes have k^* neighbors in V^*. So, the number of edges

is:

$$|E| \geq \frac{k^* \cdot |V^{\perp}|}{2} = \frac{k^* \cdot |V|}{2}.$$

Case 2: When $\arg\max |V^{\top}|$.

We know that every node in $V^{\top}$ connects to at (k^*+1) nodes in $V^{\perp}$, and any node in $V^{\perp}$ can have at most k^* connections to $V^{\top}$. That is,

$$\begin{aligned} k^* \cdot |V^{\perp}| &\geq (k^*+1) \cdot |V^{\top}| \\ |V^{\top}| &\leq \frac{k^* \cdot |V^{\perp}|}{k^*+1}. \end{aligned}$$

We know that,

$$\begin{aligned} |V^{\top}| + |V^{\perp}| &= |V^*| \\ \frac{2k^*+1}{k^*+1}|V^{\perp}| &\geq |V^*| \\ |V^{\perp}| &\geq \frac{k^*+1}{2k^*+1} \cdot |V^*|. \end{aligned}$$

When we have $\max |V^{\top}|$, we get $\min |V^{\perp}|$. By definition, there are no edges between any pair of node from $V^{\top}$, and every node in $V^{\perp}$ has exactly k^* connections. In this case the number of edges is,

$$|E| \leq \frac{k^* \cdot (k^*+1)}{2k^*+1}|V^*|.$$

If we generalize to cases where $k^* \cdot |V^*|$ can be odd,

$$\left\lceil \frac{k^* \cdot |V^*|}{2} \right\rceil \leq |E^*| \leq \left\lceil \frac{(k^*+1) \cdot k^* \cdot |V^*|}{2k^*+1} \right\rceil.$$

For $k^* > 0$,

$$\max\left(\left\lceil\frac{(k^*+1)\cdot k^*\cdot|V^*|}{2k^*+1}\right\rceil-\left\lceil\frac{k^*\cdot|V^*|}{2}\right\rceil\right)=1. \tag{7.5}$$

Thus,

$$\left\lceil\frac{k^*\cdot|V^*|}{2}\right\rceil\leq|E^*|\leq\left\lceil\frac{k^*\cdot|V^*|}{2}\right\rceil+1.$$

□

7.2 Skeletal Core Graph and Core Structural Change

In this section, we estimate the core resilience of a core skeletal graph. The core resilience is defined as the rank correlation between the rankings of the nodes, as ranked by the core numbers, before and after edge deletion. Once an edge is deleted from a core skeletal graph, the resulting graph is no longer a core skeletal graph. So, we start by examining the effect of one edge deletion. We also make the simplifying assumption that there are only two shells: the k-shell and the $(k-1)$-shell. This does not affect the overall validity as we can further extend the same argument by considering lower shells.

To calculate the core resilience of G, we need to compute the following in steps:

1. Probability that the deleted edge is from E_k, E_{k-1} and $E_{k,k-1}$. Represent these by $p(E_k)$, $p(E_{k-1})$ and $p(E_{k,k-1})$ respectively.

2. Core resilience as a result of the edge deletion from each set of edges. Represent these by $r(E_k)$, $r(E_{k-1})$ and $r(E_{k,k-1})$.

3. Core resilience of G, is

$$R(G) = p(E_k)r(E_k) + p(E_{k-1})r(E_{k-1}) + p(E_{k,k-1})r(E_{k,k-1}).$$

7.2.1 Core Resilience of Skeletal Core Graph

For simplicity, assume that there are only two shells – the k-shell and the $(k-1)$-shell, in a skeletal core graph, $G^\sigma = \langle V^\sigma, E^\sigma \rangle$. Let $|V_k| = n$, $|V_{k-1}| = fn$, and $|E_{k-1}| = g$. As described above, the result easily generalizes to more shells.

STEP 1:

$$|E_k| = \frac{kn}{2} \tag{7.6}$$

$$|E_{k-1}| \leq \frac{(k-1)fn}{2} \tag{7.7}$$

$$|E_{k,k-1}| \leq (k-1)fn \tag{7.8}$$

An edge in E_{k-1} is responsible for the core number of 2 nodes in the $(k-1)$-shell, and an edge in $E_{k,k-1}$ is responsible for the core number of 1 node in the $(k-1)$-shell. So,

$$|E_{k,k-1}| = 2(\frac{(k-1)fn}{2} - |E_{k-1}|) \tag{7.9}$$

$$= (k-1)fn - 2g \tag{7.10}$$

Then,

$$|E| = \frac{kn}{2} + g + (k-1)fn - 2g = \frac{n}{2}(k + 2(k-1)f - 2g). \tag{7.11}$$

The probabilities of the random edge being deleted from the different sets of edges is given by,

$$p(E_k) = \frac{kn}{k + 2(k-1)f - 2g} \tag{7.12}$$

$$p(E_{k-1}) = \frac{2g}{k + 2(k-1)f - 2g} \tag{7.13}$$

$$p(E_{k,k-1}) = \frac{2((k-1)fn - 2g)}{k + 2(k-1)f - 2g} \tag{7.14}$$

STEP 2: We need to consider three cases: (1) core resilience due to edge deletion from E_k, (2) core resilience due to edge deletion from E_{k-1}, and (3) core resilience due to edge deletion from $E_{k,k-1}$.

Case 1: Edge deletion from E_k.

Let m the the number of nodes that change core number due to the edge removal. We can easily show that these nodes will now be in the the $(k-1)$-shell.

There are two factors that contributes to the concordant pairs count[1]:

- ☐ The pairing between the nodes that do not change core number. That is, $\binom{n+fn-m}{2}$.
- ☐ The pairing between the nodes that change core number. That is, $\binom{m}{2}$.

So, the number of concordant pairs is,

$$\binom{n(f+1)-m}{2} + \binom{m}{2}. \tag{7.15}$$

There is only one factor that contributes to the discordant pairs count: the pairing between the nodes that change core number and those that did not. So, the number of discordant pairs is:

$$m(n(f+1) - m) \tag{7.16}$$

[1] Recall that we count ties as concordant in the definition of core resilience.

Then, the core resilience is,

$$r(E_k) = \frac{\binom{n(f+1)-m}{2} + \binom{m}{2} - m(n(f+1)-m)}{\binom{n(f+1)}{2}} \tag{7.17}$$

$$= \frac{1}{\binom{n(f+1)}{2}} \left(\binom{n(f+1)}{2} - 2m\left(n(f+1)-m\right) \right) \tag{7.18}$$

$$= 1 - \frac{2}{c}\left(mn(f+1) - m^2\right), \tag{7.19}$$

where $c = \binom{n(f+1)}{2}$ is a constant and does not change because the total number of nodes is constant.

We can show that $k \leq m \leq n$. The minimum value of m is when both endpoints of the deleted edge are in a clique; and the maximum value of m is when the entire k-shell collapses in a cascade.

Now,

$$\frac{\mathrm{d}}{\mathrm{d}m} r(E_k) = 0 \tag{7.20}$$

$$-n(f+1) + 2m = 0 \tag{7.21}$$

$$m = \frac{n(f+1)}{2} \tag{7.22}$$

We know that m should be within the range $[k, n]$. So

$$\min r(E_k) = 1 - \frac{n^2 f}{2c} \text{ when } m = n \tag{7.23}$$

$$\max r(E_k) = 1 - \frac{2}{c}(kn(f+1) - k^2) \text{ when } m = k \tag{7.24}$$

Case 2: Edge deleted from E_{k-1}.

Again, let m the number of nodes that changes core number. Then, there are three factors that affects the concordant pairs count:

- ☐ The pairing between the nodes that do not change core number. That is, $\binom{n(f+1)-m}{2}$.

- ☐ The pairing between the nodes that change core number. That is, $\binom{m}{2}$.
- ☐ The pairing between the nodes that change core number and the nodes in k-shell. That is, mn.

So, the number of concordant pairs is,

$$\binom{n(f+1)-m}{2} + \binom{m}{2} + mn = \binom{n(f+1)}{2} - mfn + m^2. \tag{7.25}$$

There is only one factor that affects the number of discordant pairs – the pairing between the nodes that change core number and the rest of the nodes in the $(k-1)$-shell. That is the number of discordant pairs is,

$$m(fn-m) = mfn - m^2. \tag{7.26}$$

Then, the core resilience is,

$$r(E_{k-1}) = \frac{1}{c}\left(\binom{n(f+1)}{2} - 2mfn + 2m^2\right) \tag{7.27}$$

$$= 1 - \frac{2}{c}m(fn-m). \tag{7.28}$$

When an edge in E_{k-1} is deleted, we can guaranteed that at least the two endpoints of the edge will drop core number. So,

$$2 \leq m \leq nf\,. \tag{7.29}$$

We can get a tighter bound by considering the number of edges in $|E_{k-1}|$.

Now let us calculate the minimum and maximum value of the core resilience. The maximum value

of $r(E_{k-1})$ is 1, and it happens when $m = nf$. Now,

$$\frac{\mathrm{d}}{\mathrm{d}x} r\left(E_{k-1}\right) = 0 \tag{7.30}$$

$$-\frac{2}{c}(fn - 2m) = 0 \tag{7.31}$$

$$m = \frac{fn}{2}. \tag{7.32}$$

So, we have,

$$\max r(E_{k-1}) = 1 \text{ when } m = fn \tag{7.33}$$

$$\min r(E_{k-1}) = 1 - \frac{f^2 n^2}{2c} \text{ when } m = \frac{fn}{2} \tag{7.34}$$

Case 3: Edges deleted from $E_{k,k-1}$.

Again in this case, let m be the number of nodes that changes core number; and all of them come from the $(k-1)$-shell. So, like in the Case 2, the core resilience is given by,

$$r(E_{k,k-1}) = 1 - \frac{2}{c} m(fn - m). \tag{7.35}$$

In this case, only one endpoint of the deleted edge is is the $(k-1)$-shell. So,

$$1 \leq m \leq fn. \tag{7.36}$$

7.2.2 Core Resilience of Decentralized Core Skeletal Graphs

In a decentralized core skeletal graph, $E_{k,k-1} = \emptyset$. So, the probability of edge deletion is given by,

$$p(E_k) = \frac{k}{k + f(k-1)} \tag{7.37}$$

$$p(E_{k-1}) = \frac{f(k-1)}{k + f(k-1)} \tag{7.38}$$

$$p(E_{k,k-1}) = 0 \tag{7.39}$$

Let m_0 and m_1 be the number of nodes that changes core number if an edge is deleted in E_k and E_{k-1} respectively.

Then the expected core resilience is,

$$R(G_D^\sigma) = p(E_k)r(E_k) + p(E_{k-1})r(E_{k-1}) \tag{7.40}$$

$$= \frac{k\left(1 - \frac{2}{c}(m_0 n(f+1) - m_0^2)\right)}{k + f(k-1)} + \frac{f(k-1)\left(1 - \frac{2}{c}(m_1 nf - m_1^2)\right)}{k + f(k-1)} \tag{7.41}$$

$$= 1 - \frac{2\left(k(m_0 n(f+1) - m_0^2) + f(k-1)(nm_1 f - m_1^2)\right)}{c(k + f(k-1))} \tag{7.42}$$

Random Skeletal Core Graph: Under the assumption that nk and $nf(k-1)$ are even, it is easy to see that G_k^D and G_{k-1}^D are k-regular and $(k-1)$-regular graphs respectively. We know that for 'a random r-regular graph of large size is asymptotically almost surely r-connected'[18]. We also know that when an edge is deleted in a skeletal core graph, all the nodes in the connected component that the node with lower number belongs to drops to a lower core. That is,

$$m_0 = n \tag{7.43}$$

$$m_1 = fn \tag{7.44}$$

So, resilience,

$$R(G^D) = 1 - \frac{2(k(n^2(f+1) - n^2) + f(k-1)(f^2n^2 - f^2n^2))}{c(k + f(k-1))} \tag{7.45}$$

$$= 1 - \frac{2n^2kf}{c(k + f(k-1))} \tag{7.46}$$

Skeletal Core Subgraph: If we are not dealing with a random graph, but rather, the result of skeletal core decomposition of a given graph, we need to replace m_k and m_{k-1} with the expected size of the connected component that a randomly chosen node belongs to.

7.2.3 Core Resilience of Centralized Core Skeletal Graph

In a decentralized core skeletal graph, $E_{k-1} = \emptyset$. So, the probability of edge deletion is given by,

$$p(E_k) = \frac{k}{k + 2f(k-1)} \tag{7.47}$$

$$p(E_{k-1}) = 0 \tag{7.48}$$

$$p(E_{k,k-1}) = \frac{2(k-1)f}{k + 2(k-1)f} \tag{7.49}$$

Again, let m_0 and m_1 be the number of nodes that change core number if an edge is deleted in E_k^D and E_{k-1}^D respectively. In this case, for the edge deletion from $E_{k,k-1}$, $m_1 = 1$ because the node whose core number changed is not connected to any other node in $(k-1)$-shell.

Then, the core resilience of the centralised core skeletal graph due to one edge deletion is given by,

$$R(G_C^\sigma) = p(E_k)r(E_k) + p(E_{k,k-1}r(E_{k,k-1})) \tag{7.50}$$

$$= \frac{k(1 - \frac{2}{c}(m_0n(f+1) - m_0^2))}{k + 2f(k-1)} + \frac{2f(k-1)(1 - \frac{2}{c}(fn-1))}{k + 2f(k-1)} \tag{7.51}$$

$$= 1 - \frac{2\left(k(m_0n(f+1) - m_0^2) + 2f(k-1)(fn-1)\right)}{c(k + 2f(k-1))}. \tag{7.52}$$

Random Skeletal Core Graph: Again in the case of a random graph, $m_0 = n$. So, the resilience is,

$$R(G^C) = 1 - \frac{2(kn^2f + 2f(k-1)(fn-1))}{c(k + 2f(k-1))}. \tag{7.53}$$

Skeletal Core Subgraph: Again if the skeletal core we are dealing with is derived from some other graph, we estimate m_0 as the estimated size of the connected component that a randomly selected node in the k-shell is a member of.

7.2.4 Core Resilience of Centralized vs Decentralized Skeletal Core Graphs

Over all the possible decentralized and centralized core skeletal graphs (assuming n, f, k are the same[2]), which one has the highest core resilience?

We only need to consider the resilience for the random graphs.

$$R(G_D^\sigma) - R(G_C^\sigma) = -\frac{2n^2kf}{c(k + f(k-1))} + \frac{2(kn^2f + 2f(k-1)(fn-1))}{c(k + 2f(k-1))}. \tag{7.54}$$

We can show that for $n > 2$, $R(G_D^\sigma) - R(G_C^\sigma) < 0$. That is, for a random graph, the centralized skeletal core graph has higher resilience than the decentralized skeletal core graph.

7.2.5 Experiment

From Equation 7.54, we know that skeletal core graphs that are more centralized have higher core resilience as compared to decentralized ones. So, make this hypothesis that for similar size graphs of approximately similar distribution of nodes in each shell, the graphs that has skeletal that are more centralized are more likely to have higher core resilience.

To verify this experimentally, we take 16 real world graphs of approximately similar number of nodes

[2] n is the number of nodes, f is the distribution of nodes in the different shells, k is the maximum core number.

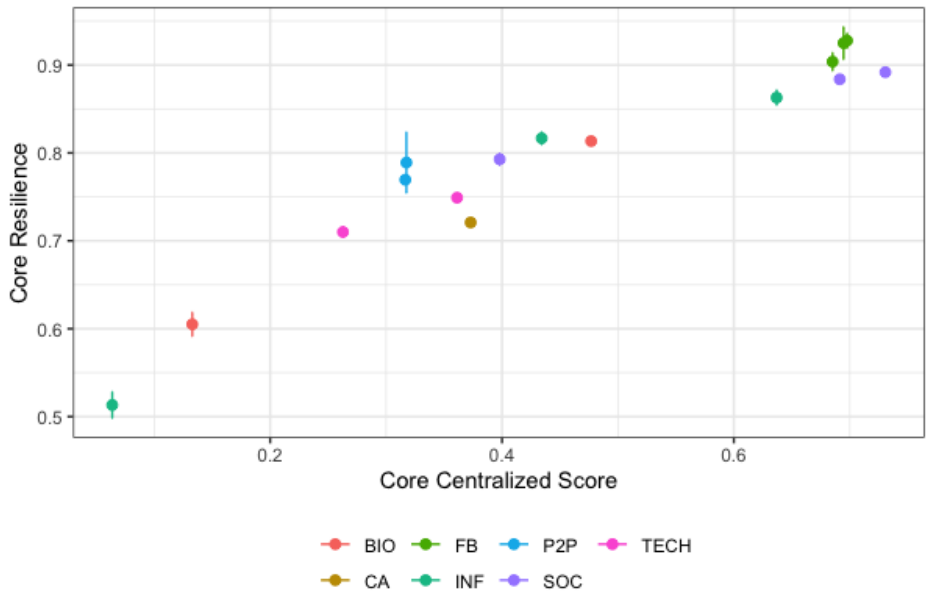


Figure 7.4: Core Centralized Score (x-axis) vs Core Resilience (y-axis) for various real-world networks.

(10^3) and similar distribution of nodes in each k-shell (Table C.1). We calculate the centralized core score for each of these graphs, and calculate the core resilience of these graphs (to 10% edge deletion for the entire structure). Figure 7.4 shows the Core Centralized Score in the x-axis and Core Resilience in the y-axis. Each point represents one graph. In the figure, we can see that graphs with higher core centralized score have higher core resilience in real-world graphs.

7.3 Skeletal Core Graph and Graph Unraveling

When we consider the anchored k-core problem, the concept of skeletal core graphs can also provide insight into why it is easier to find anchors in some graphs than others. We already know that it is easier to convert a node in $(k-1)$-shell into a follower compared to one in $(k-i)$-shell, where $i > 1$. So for this discussion we will consider only the k-core and the $k-1$-shell, i.e. we assume that all the followers will come from the $k-1$-shell.

In this section, we ask the question, *what effect does the connections between the k-core and $(k-1)$-shell have on the number of followers, given a fixed number of anchors?* Because we are talking about

why it is easier to find lots of followers in some graphs compared to others, we are not talking about the optimal anchor selection. So we will consider multiple random anchor selection.

Theorem 7.11. *All the nodes in $V \setminus V_k$ forms a skeletal core sub-graph, then the increase in the size of the anchored k-core comes purely from the anchor nodes and there will be no follower regardless of the number of anchors allowed.*

Proof. This follows directly form the concept of candidate followers. If all the nodes in $V \setminus V_k$ satisfies the condition to be in a skeletal core graph, the set of candidate followers is an empty set. □

Theorem 7.12. *The minimum number of edges such that set $V' \subset V \setminus V_k$ can still be candidate followers (with respect to the k-core) if the core strength of node $u \in V'$ is, $CS(u, G) = 1 + (k - \kappa(k, G))$. We will refer to the subgraph induced by such nodes as Nearly Skeletal Core Sub-Graph.*

Proof. This follows directly from the concept of core strength. □

To study the effect of the connections between the k-core and $(k-1)$-shell, we assume that:

1. The number of candidate nodes in in the $(k-1)$-shell is the same.
2. All the candidate nodes are in the Nearly Skeletal Core Sub-Graph.

To understand the structural difference behind why we have a lot of followers in some graphs than others for the same number of anchors, we need to study:

1. Ease of converting a node to a follower.
2. Size of cascade (of nodes becoming followers), due to anchors.
3. Ease of triggering a cascade; that is a node u when anchored leading to other nodes not directly connected to u also getting into the anchored k-core.

Ease of Converting One Node to a Follower: From [54], we know that nodes with higher *residual degree* become followers more easily than those with lower values. It is easy to show that in nearly

skeletal sub-graphs that have higher core centralized score, the residual degree will be higher compared to the ones with lower core centralized score.

Size of Cascade: The size of a cascade in dictated by the number of connected components in the nearly skeletal core sub-graph. If there is

Ease of Triggering a Cascade within a Connected Component: The ease of triggering a cascade within an connected component is determined by the ease of converting the nodes in the component to followers. That is, it directly relates to the core centralized score – cascades are more likely to be triggered in components with higher k-centralized scores.

We combine these into a score that can tell us if the graph is likely to have lot of followers or not:

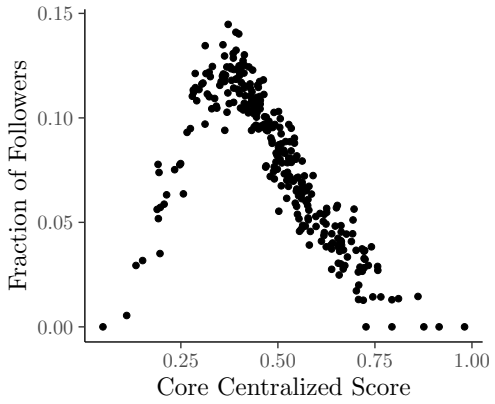
$$\alpha_R = \sum_{S' \in \mathbb{S}} \frac{|S'| - \beta}{n} \cdot CE_k(S'). \tag{7.55}$$

where S is the set of connected components in the induced sub-graph, n is the number of candidate followers, and $\beta \in \mathbb{Z}_+$ depends on the number of anchors selected.

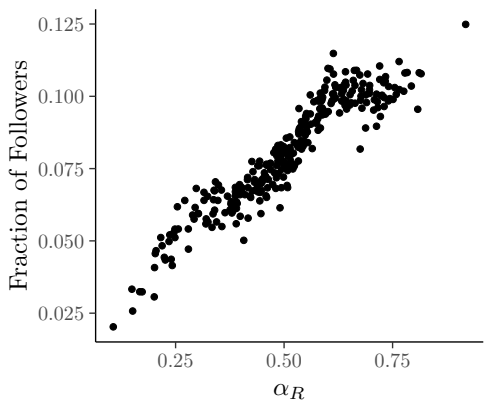
7.3.1 Number of Anchors and k-Centralized Score

From Equation (7.55) we know that in a connected component with higher k-centralized score, it is easier to find anchors with large number of followers. However, we also know from Section 7.2 that the number of edges within the shell decreases as the it gets closer to a centralized skeletal core graph. We know that the expected number of connected components increases as the number of edges decreases [2, 27, 83].

So, as the k-centralized score of a graph increases, we expect the number of followers to increase initially; but after some point it will start to drop. This point of maximum number of followers depends on the number of nodes as well as the value of k. Computing this theoretically requires estimating the number of connected components, and currently, no such techniques exist. We thus perform



(a) Core Centralized Score (x-axis) against the Fraction of followers (y-axis).



(b) α_r (x-axis) against the Fraction of followers (y-axis).

Figure 7.5: Simulation results relating the fraction of followers against the Core Centralized Score (Figure 7.5a) and α_R (Figure 7.5b). As expected we can see in Figure 7.5a that the fraction of followers increases with core centralized score initially, but decreases after reaching some peak. In Figure 7.5b, we can see that the fraction of followers increases with α_R as expected theoretically.

an experimental analysis instead.

7.3.2 Experiments

To check the relationship between α_r and the number of followers experimentally, we generate 300 graphs that consist of a 10-core and a 9-shell with 100 nodes each. The α_r value of each of these graphs are calculated, and for each graph 10 nodes from the 9-shell is selected randomly as anchor and the number of followers is calculated. This is repeated 30 times.

Figure 7.5 shows the results of the experiment. Here the x-axis is α_r, the y-axis is the fraction of followers (to the number of candidate followers), and each dot is a graph. We can clearly see that in graphs with higher values of α_r, it is easier to find good anchors.

7.4 Conclusion

In this chapter, we consider proposed the *Skeletal Core Graph* – the graph such that if any edge is removed, the core number of at least one node will change. We show how we can efficiently check if a given graph is a skeletal core graph, and to generate random skeletal core graphs of a given core number sequence.

We also propose two types of skeletal core graphs – the decentralized and centralized skeletal core graphs. Theoretically we show that graphs that are closer to the decentralized skeletal core graphs have lower resilience to core structural changes.

Finally, we relate the ease of selecting anchors for the anchored k-core problem with how close the graph is to decentralized or centralized core graphs. We show through simulations that in graphs that are very close to either of these categories, it is difficult to get anchors that results in large number of followers. However, there is some middle ground between these two were the graphs have a lot of followers to anchors.

Appendices

Appendix A

Core Structural Change

A.1 Edge Deletion and Node Deletion

We define the core resilience under two scenarios in which the ranking of the nodes by core number might change: edge deletion and node deletion. Note that node deletion can be treated as a special type of edge deletion, as when a node is deleted, all of its edges are deleted. In this section, we show the relationship between core resilience due to edge deletion and that due to node deletion.

Consider, a graph $G = \langle V, E \rangle$. The (r, p)-core resilience of G is given by $\mathcal{R}_r^{n(p)}(G)$ and $\mathcal{R}_r^{e(p)}(G)$ (by definition) for node deletion and edge deletion, respectively.

Assume that deletion of p nodes results in deletion of p' edges. It is reasonable to assume $p' > p$, since real-world networks rarely have an average degree of one. That is, $\mathcal{R}_r^{e(p')}(G) \approx \mathcal{R}_r^{n(p)}(G)$, and in general $\mathcal{R}_r^{e(p)}(G) \geq \mathcal{R}_r^{e(p')}(G)$. So, $\mathcal{R}_r^{n(p)}(G) \leq \mathcal{R}_r^{e(p)}(G)$.

Now let us consider the (r, p_l, p_u)-core resilience under edge deletion and node deletion.

$$\mathcal{R}_r^{n(p_l,p_u)}(G) - \mathcal{R}_r^{e(p_l,p_u)}(G) = \frac{\int_{p_l}^{p_u} \left(\mathcal{R}_r^{n(x)}(G) - \mathcal{R}_r^{e(x)}(G) \right) dx}{p_u - p_l}$$

$$\mathcal{R}_r^{n(p_l,p_u)}(G) \leq \mathcal{R}_r^{e(p_l,p_u)}(G) \tag{A.1}$$

A.2 Datasets

Type	Network	$\|V\|$	$\|E\|$	k^*
AS	AS_733_19971108†	3015	5196	9
	AS_733_19990309†	4759	8896	12
	Oregon1_010331†	10670	22002	17
	Oregon1_010428†	10886	22493	17
BIO	BIO_Dmela‡	7393	25569	11
	BIO_Yeast_Protein‡	1846	2203	5
CA	CA_GrQc†	5241	14484	43
	CA_HepTh†	9875	25973	31
	CA_Erdos992‡	5094	7515	7
INF	INF_OpenFlights‡	2939	15677	28
	INF_Power‡	4941	6594	5
	INF_USAir97‡	332	126	26
P2P	P2P_Gnutella08†	6301	20777	10
	P2P_Gnutella09†	8114	26013	10
	P2P_Gnutella25†	22687	54705	5
SOC	SOC_Hamsterster‡	2426	16630	4
	SOC_Advogato‡	5167	39432	5
	SOC_Wiki_Vote‡	889	2914	9
TECH	TECH_Pgp‡	10680	24316	31
	TECH_Routers_rf‡	2113	6632	15
	TECH_WHOIS‡	7476	56943	88
WEB	WEB_Spam‡	4767	37375	35
	WEB_Webbase‡	16062	25593	32

Table A.1: Real-world networks used for experiments. In this table, $|V|$ is the number of nodes, $|E|$ is the number of edges, and k^* is the degeneracy. These datasets were downloaded from SNAP (denoted by †), and Network Repository (denoted by ‡).

Appendix B

Graph Unraveling

B.1 Dataset

Network	Abbr.	$\|V\|$	$\|E\|$	k_{max}	k_{mid}	$\|C_a\|$	$\|C_f\|$	$\|E_{fa}\|$	$\|\mathcal{G}\|$
socfb-combined	FC	4.0×10^3	8.8×10^4	115	17	1289	501	7029	13
ca-CondMat	CC	2.3×10^4	9.3×10^4	25	4	2892	1179	3739	685
ca-HepPh	CH	1.2×10^4	1.1×10^5	238	4	1487	634	1901	362
loc-Brightkite	LB	5.8×10^4	2.1×10^5	52	2	3288	2365	3004	2006
socfb-Northeastern19	FN	1.4×10^4	3.8×10^5	43	33	4978	1246	18473	1
socfb-Syracuse56	FS	1.4×10^4	5.4×10^5	75	46	5522	1417	37698	2
ca-citeseer	CS	2.2×10^5	8.1×10^4	86	3	18486	8493	20187	5991
loc-Gowalla	LG	1.9×10^5	9.5×10^5	51	3	17890	10263	17706	7479
com-DBLP	KD	3.1×10^5	1.0×10^6	113	3	23182	11010	25144	8240
web-Google	WG	8.7×10^5	4.3×10^6	44	4	198014	46891	245188	20471
soc-Catster	SC	1.5×10^5	5.4×10^6	419	21	5285	2003	8428	1054
soc-Dogster	SD	4.2×10^5	8.5×10^6	248	12	20887	8750	26438	5339
soc-TwitterHiggs	ST	4.5×10^5	1.3×10^7	125	17	27146	9234	40651	3493
web-Hudong	WH	2.0×10^6	1.4×10^7	266	5	82791	40160	83886	29687
web-BaiduBaike	WB	2.0×10^6	1.7×10^7	78	3	51659	32501	50222	27735

Table B.1: Statistics of the real-world networks used in our experiments. $|V|$ and $|E|$ are the number of nodes and edges respectively; k_{max} and k_{mid} are the maximum and median values of the coreness of all the nodes. $|C_a|$ and $|C_f|$ are sizes of the candidate anchors and followers for k_{mid}. $|E_{fa}|$ is the number of edges in the subgraph induced with $C_f \cup C_a$, and $|\mathcal{G}|$ is the number of connected components.

Appendix C

Skeletal Core Graph

C.1 Dataset

Network	**Category**	$\|V\|$	k_{max}
AS_733_19971108	AS	3015	9
AS_733_19990309	AS	47509	12
Bio_Dmela	BIO	7393	11
Ca_Erdos	CA	5094	7
P2P_Gnutella08	P2P	6301	10
P2P_Gnutella09	P2P	8114	10

Table C.1: Data sets used for experiments in Chapter 6.

References

[1] Adiga, A., and Vullikanti, A. K. S. How robust is the core of a network? In *Machine Learning and Knowledge Discovery in Databases: European Conference* (2013), Springer.

[2] Aiello, W., Chung, F., and Lu, L. A random graph model for massive graphs. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing* (2000), pp. 171–180.

[3] Al-garadi, M. A., Varathan, K. D., and Ravana, S. D. Identification of influential spreaders in online social networks using interaction weighted k-core decomposition method. *Physica A 468* (2017).

[4] Altaf-Ul-Amine, M., Nishikata, K., Korna, T., Miyasato, T., Shinbo, Y., Arifuzzaman, M., Wada, C., Maeda, M., Oshima, T., Mori, H., et al. Prediction of protein functions based on k-cores of protein-protein interaction networks and amino acid sequences. *Genome Informatics 14* (2003).

[5] Alvarez-Hamelin, J. I., Barrat, A., and Vespignani, A. Large scale networks fingerprinting and visualization using the k-core decomposition. In *Advances in Neural Information Processing Systems* (2006).

[6] Alvarez-Hamelin, J. I., Dall'Asta, L., Barrat, A., and Vespignani, A. K-core decomposition of internet graphs: hierarchies, self-similarity and measurement biases. *arXiv preprint cs/0511007* (2005).

[7] Alvarez-Hamelin, J. I., Dall'Asta, L., Barrat, A., and Vespignani, A. Large scale networks fingerprinting and visualization using the k-core decomposition. In *Advances in Neural Information Processing Systems* (2006).

[8] Alvarez-Hamelin, J. I., Dall'Asta, L., Barrat, A., and Vespignani, A. K-core decomposition of Internet graphs: hierarchies, self-similarity and measurement biases. *Networks and Heterogeneous Media 3*, 2 (2008).

[9] Areekijseree, K., Laishram, R., and Soundarajan, S. Guidelines for online network crawling: A study of data collection approaches and network properties. In *Proceedings of the 10th ACM Conference on Web Science, WebSci* (2018).

[10] Batagelj, V., and Zaversnik, M. An o(m) algorithm for cores decomposition of networks. Tech. Rep. cs/0310049, Arxiv, 2003.

[11] Bauer, R., Krug, M., and Wagner, D. Enumerating and generating labeled k-degenerate graphs. In *2010 Proceedings of the Seventh Workshop on Analytic Algorithmics and Combinatorics (ANALCO)* (2010), SIAM.

[12] Baur, M., Gaertler, M., Görke, R., Krug, M., and Wagner, D. Generating graphs with predefined k-core structure. In *Proceedings of the European Conference of Complex Systems* (2007).

[13] Bavelas, A. Communication patterns in task-oriented groups. *The Journal of the Acoustical Society of America 22*, 6 (1950).

[14] Bhawalkar, K., Kleinberg, J., Lewi, K., Roughgarden, T., and Sharma, A. Preventing unraveling in social networks: the anchored k-core problem. In *International Colloquium on Automata, Languages and Programming* (2012).

[15] Bhawalkar, K., Kleinberg, J., Lewi, K., Roughgarden, T., and Sharma, A. Preventing unraveling in social networks: the anchored k-core problem. *SIAM Journal on Discrete Mathematics 29*, 3 (2015).

[16] Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, É. The louvain method for community detection in large networks. *Journal of Statistical Mechanics: Theory and Experiment 10* (2011).

[17] Bohlin, L., Edler, D., Lancichinetti, A., and Rosvall, M. Community detection and visualization of networks with the map equation framework. In *Measuring Scholarly Impact: Methods and Practice*. Springer, 2014.

[18] Bollobás, B. *Modern graph theory*, vol. 184. Springer Science & Business Media, 2013.

[19] Bonchi, F., Bordino, I., Gullo, F., and Stilo, G. Identifying buzzing stories via anomalous temporal subgraph discovery. In *2016 IEEE/WIC/ACM International Conference on Web Intelligence (WI)* (2016), IEEE.

[20] Bonchi, F., Gullo, F., Kaltenbrunner, A., and Volkovich, Y. Core decomposition of uncertain graphs. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2014).

[21] Brandes, U., and Pich, C. Centrality estimation in large networks. *International Journal of Bifurcation and Chaos 17*, 07 (2007).

[22] Burleson-Lesser, K., Morone, F., Tomassone, M. S., and Makse, H. A. K-core robustness in ecological and financial networks. *Scientific reports 10*, 1 (2020).

[23] Cai, Y., Huang, F., Wang, S., Zhang, H., and Du, C. Research hotspots mining and visualized analysis based on linking cluster and k-core decomposition. In *Proceedings of the International Conference on Data Processing and Applications* (2018).

[24] Carmi, S., Havlin, S., Kirkpatrick, S., Shavitt, Y., and Shir, E. A model of internet topology using k-shell decomposition. *Proceedings of the National Academy of Sciences 104*, 27 (2007).

[25] Cheng, J., Ke, Y., Chu, S., and Özsu, M. T. Efficient core decomposition in massive networks.

In *IEEE International Conference on Data Engineering* (2011).

[26] Chitnis, R. H., Fomin, F. V., and Golovach, P. A. Preventing unraveling in social networks gets harder. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence* (2013).

[27] Chung, F., and Lu, L. Connected components in random graphs with given expected degree sequences. *Annals of combinatorics 6*, 2 (2002).

[28] Clauset, A., Newman, M. E. J., and Moore, C. Finding community structure in very large networks. *Phys. Rev. E 70* (Dec 2004).

[29] Cohen, J. Trusses: Cohesive subgraphs for social network analysis.

[30] Cohen, J. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report 16* (2008).

[31] Coja-Oghlan, A., Feige, U., Krivelevich, M., and Reichman, D. Contagious sets in expanders. In *Proceedings of the twenty-sixth annual ACM-SIAM Symposium on Discrete Algorithms* (2014), SIAM.

[32] Dorogovtsev, S. N., Goltsev, A. V., and Mendes, J. F. F. K-core organization of complex networks. *Physical Review Letters 96*, 4 (2006).

[33] Elsharkawy, S., Hassan, G., Nabhan, T., and Roushdy, M. Studying the dissemination of the k-core influence in twitter cascades. In *IFIP International Conference on Artificial Intelligence Applications and Innovations* (2018), Springer.

[34] Erdos, P., and Hajnal, A. On chromatic number of graphs and set-systems. *Acta Mathematica Hungarica 17* (1966).

[35] Esfandiari, H., Lattanzi, S., and Mirrokni, V. S. Parallel and streaming algorithms for k-core decomposition. In *Proceedings of the 35th International Conference on Machine Learning* (2018).

[36] Feige, U., Krivelevich, M., Reichman, D., et al. Contagious sets in random graphs. *The Annals of Applied Probability 27*, 5 (2017).

[37] Freeman, L. C. A set of measures of centrality based on betweenness. *Sociometry* (1977).

[38] Garcia, D., Mavrodiev, P., and Schweitzer, F. Social resilience in online communities: The autopsy of friendster. In *Conference on Online Social Networks* (2013).

[39] Giatsidis, C., Thilikos, D. M., and Vazirgiannis, M. Evaluating cooperation in communities with the k-core structure. In *International Conference on Advances in Social Networks Analysis and Mining* (2011).

[40] Giatsidis, C., Thilikos, D. M., and Vazirgiannis, M. D-cores: measuring collaboration of directed graphs based on degeneracy. *Knowledge and Information Systems 35*, 2 (2013).

[41] Girvan, M., and Newman, M. E. Community structure in social and biological networks. *Proceedings of the national academy of sciences 99*, 12 (2002).

[42] Goldschmidt, O., Nehme, D., and Yu, G. Note: On the set-union knapsack problem. *Naval Research Logistics (NRL) 41*, 6 (1994).

[43] Gong, K., and Kang, L. A new k-shell decomposition method for identifying influential spreaders of epidemics on community networks. *Journal of Systems Science and Information 6*, 4 (2018).

[44] Govindan, P., Wang, C., Xu, C., Duan, H., and Soundarajan, S. The k-peak decomposition: Mapping the global structure of graphs. In *Proceedings of the 26th International Conference on World Wide Web* (2017).

[45] Guggiola, A., and Semerjian, G. Minimal contagious sets in random regular graphs. *Journal of Statistical Physics 158*, 2 (2015).

[46] Jun, W., Barahona, M., Yue-Jin, T., and Hong-Zhong, D. Natural connectivity of complex

networks. *Chinese physics letters 27*, 7 (2010).

[47] Karrer, B., and Newman, M. E. Stochastic blockmodels and community structure in networks. *Physical Review E 83* (Jan 2011).

[48] Kermack, W. O., and McKendrick, A. G. A contribution to the mathematical theory of epidemics. *Proceedings of the royal society of london. Series A, Containing papers of a mathematical and physical character 115*, 772 (1927).

[49] Khaouid, W., Barsky, M., Venkatesh, S., and Thomo, A. K-core decomposition of large networks on a single PC. *Proceedings of the VLDB Endowment 9*, 1 (2015).

[50] Kitsak, M., Gallos, L. K., Havlin, S., Liljeros, F., Muchnik, L., Stanley, H. E., and Makse, H. A. Identification of influential spreaders in complex networks. *Nature physics 6*, 11 (2010).

[51] Kramer, S. Anomaly detection in extremist web forums using a dynamical systems approach. In *ACM SIGKDD Workshop on Intelligence and Security Informatics* (2010), pp. 1–10.

[52] Laishram, R., Areekijseree, K., and Soundarajan, S. Predicted max degree sampling: Sampling in directed networks to maximize node coverage through crawling. In *Proceedings IEEE INFOCOM Workshops* (2017), no. 2017.

[53] Laishram, R., Sariyüce, A. E., Eliassi-Rad, T., Pinar, A., and Soundarajan, S. Measuring and improving the core resilience of networks. In *Proceedings of the 2018 World Wide Web Conference* (2018).

[54] Laishram, R., Sariyüce, A. E., Eliassi-Rad, T., Pinar, A., and Soundarajan, S. Residual core maximization: An efficient algorithm for maximizing the size of the k-core. In *SIAM Data Mining* (2020).

[55] Laishram, R., Wendt, J. D., and Soundarajan, S. Crawling the community structure of multiplex networks. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI* (2019).

[56] Lancichinetti, A., and Fortunato, S. Limits of modularity maximization in community detection. *Phys. Rev. E 84* (Dec 2011), 066122.

[57] Leon-Suematsu, Y. I., Inui, K., Kurohashi, S., and Kidawara, Y. Web spam detection by exploring densely connected subgraphs. In *2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology* (2011), vol. 1, IEEE.

[58] Li, R., Yu, J. X., and Mao, R. Efficient core maintenance in large dynamic graphs. *IEEE Transactions on Knowledge and Data Engineering 26*, 10 (2014).

[59] Liu, Y., Tang, M., Zhou, T., and Do, Y. Core-like groups result in invalidation of identifying super-spreader by k-shell decomposition. *Scientific reports 5* (2015).

[60] Long, X., Yin, W., An, L., Ni, H., Huang, L., Luo, Q., and Chen, Y. Churn analysis of online social network users using data mining techniques. In *Proceedings of the international MultiConference of Engineers and Conputer Scientists* (2012), vol. 1.

[61] Luo, S.-L., Gong, K., and Kang, L. Identifying influential spreaders of epidemics on community networks. *arXiv preprint arXiv:1601.07700* (2016).

[62] Maiya, A. S., and Berger-Wolf, T. Y. Online sampling of high centrality individuals in social networks. In *Advances in Knowledge Discovery and Data Mining, 14th Pacific-Asia Conference. Proceedings. Part I* (2010).

[63] Maiya, A. S., and Berger-Wolf, T. Y. Sampling community structure. In *Proceedings of the 19th International Conference on World Wide Web, WWW* (2010).

[64] Maiya, A. S., and Berger-Wolf, T. Y. Benefits of bias: Towards better characterization of network sampling. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining* (2011).

[65] Mantegna, R. N. Hierarchical structure in financial markets. *The European Physical Journal*

B-Condensed Matter and Complex Systems 11, 1 (1999).

[66] Matula, D., and Beck, L. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM 30*, 3 (1983).

[67] Matula, D. W. A min-max theorem for graphs with application to graph coloring. *SIAM Review 10*, 4 (1968).

[68] Medya, S., Ma, T., Silva, A., and Singh, A. K-core minimization: A game theoretic approach. *arXiv preprint arXiv:1901.02166* (2019).

[69] Moriano, P., Iyer, S., and Camp, L. J. Characterization of internet routing anomalies through graph mining. Tech. rep., 2017.

[70] Morone, F., Burleson-Lesser, K., Vinutha, H., Sastry, S., and Makse, H. A. The jamming transition is a k-core percolation transition. *Physica A 516* (2019).

[71] Morone, F., Del Ferraro, G., and Makse, H. A. The k-core as a predictor of structural collapse in mutualistic ecosystems. *Nature Physics 15*, 1 (2019).

[72] Newman, M. E. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences 103*, 23 (2006).

[73] Newman, M. E. The mathematics of networks. *The new palgrave encyclopedia of economics 2*, 2008 (2008).

[74] Newman, M. E., and Girvan, M. Finding and evaluating community structure in networks. *Phys. Rev. E 69* (Feb 2004).

[75] Nguyen, A., and Hong, S.-H. K-core based multi-level graph visualization for scale-free networks. In *2017 IEEE Pacific Visualization Symposium (PacificVis)* (2017), IEEE.

[76] O'Brien, M. P., and Sullivan, B. D. Locally estimating core numbers. In *IEEE International Conference on Data Mining* (2014).

[77] Ovelgönne, M., and Geyer-Schulz, A. An ensemble learning strategy for graph clustering. In *DIMACS Workshop: Graph Partitioning and Graph Clustering* (2012).

[78] Peixoto, T. P. Hierarchical block structures and high-resolution model selection in large networks. *Phys. Rev. X 4* (Mar 2014).

[79] Peng, C., Kolda, T. G., and Pinar, A. Accelerating community detection by using k-core subgraphs. *arXiv preprint arXiv:1403.2226* (2014).

[80] Petrov, M. Identification of unusual wallets on ethereum platform.

[81] Reichman, D. New bounds for contagious sets. *Discrete Mathematics 312*, 10 (2012).

[82] Ren, B., Deng, Y. H., He, P., and Tsang, K.-T. The comparison of four methods in finding influential spreader in social network. In *2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)* (2017), IEEE.

[83] Ross, S. M. A random graph. *Journal of applied probability 18*, 1 (1981).

[84] Rosvall, M., and Bergstrom, C. T. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences 105*, 4 (2008).

[85] Sariyüce, A. E., Gedik, B., Jacques-Silva, G., Wu, K.-L., and Çatalyürek, Ü. V. Streaming algorithms for k-core decomposition. *Proc. VLDB Endow. 6*, 6 (Apr. 2013), 433–444.

[86] Sariyuce, A. E., Seshadhri, C., Pinar, A., and Catalyurek, U. V. Finding the hierarchy of dense subgraphs using nucleus decompositions. In *WWW* (2015).

[87] Schmidt, C., Pfister, H. D., and Zdeborová, L. Minimal sets to destroy the k-core in random networks. *Physical Review E 99*, 2 (2019).

[88] Seidman, S. B. Network structure and minimum degree. *Social Networks 5*, 3 (1983).

[89] Shin, K., Eliassi-Rad, T., and Faloutsos, C. Corescope: Graph mining using k-core analysisâĂŤ

patterns, anomalies and algorithms. In *IEEE nternational Conference on Data Mining* (2016), IEEE, pp. 469–478.

[90] Talebi, M. Improving genetic algorithm operators for analyzing anomalous behavior of online customers. *International Journal of New Technology and Research 1*, 6 (2015).

[91] Tuğrul, M., and Kabakçıoğlu, A. Anomalies in the transcriptional regulatory network of the yeast saccharomyces cerevisiae. *Journal of theoretical biology 263*, 3 (2010).

[92] Vazirani, V. V. *Approximation algorithms*. Springer Science & Business Media, 2013.

[93] Wejnert, C., and Heckathorn, D. D. Web-based network sampling: efficiency and efficacy of respondent-driven sampling for online research. *Sociological Methods & Research 37*, 1 (2008).

[94] Wen, D., Qin, L., Zhang, Y., Lin, X., and Yu, J. I/o efficient core graph decomposition at web scale. In *IEEE International Conference on Data Engineering* (2016).

[95] Wendt, J. D., Wells, R., Field, R. V., and Soundarajan, S. On data collection, graph construction, and sampling in twitter. In *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining* (2016), IEEE.

[96] Yang, X., Huang, D., and Zhang, Z.-K. Neighborhood coreness algorithm for identifying a set of influential spreaders in complex networks. *TIIS 11*, 6 (2017).

[97] Zachary, W. W. An information flow model for conflict and fission in small groups. *Journal of anthropological research 33*, 4 (1977).

[98] Zhang, F., Zhang, W., Zhang, Y., Qin, L., and Lin, X. Olak: an efficient algorithm to prevent unraveling in social networks. *Proceedings of the VLDB Endowment 10*, 6 (2017).

[99] Zhang, F., Zhang, Y., Qin, L., Zhang, W., and Lin, X. Finding critical users for social network engagement: The collapsed k-core problem. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence* (2017).

[100] Zhang, Y., and Parthasarathy, S. Extracting analyzing and visualizing triangle k-core motifs within networks. In *IEEE International Conference on Data Engineering* (2012), IEEE.

[101] Zhang, Y., Yu, J. X., Zhang, Y., and Qin, L. A fast order-based approach for core maintenance. In *IEEE International Conference on Data Engineering* (2017).

[102] Zhao, F., and Tung, A. K. Large scale cohesive subgraphs discovery for social network visual analysis. *Proceedings of the VLDB Endowment 6*, 2 (2012).

[103] Zhou, Z., Zhang, F., Lin, X., Zhang, W., and Chen, C. K-core maximization: An edge addition approach. In *AAAI* (2019).

[104] Zhu, W., Chen, C., Wang, X., and Lin, X. K-core minimization: An edge manipulation approach. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (2018).

Vita

Ricky Laishram

Education

1. Master of Science in Computer Science.

   Syracuse University, Syracuse, NY.

   Master's Thesis: *Link Prediction in Dynamic Weighted and Directed Social Network using Supervised Learning.*

   Thesis Advisors: Dr Kishan Mehrotra and Dr Chilukuri Mohan.

   Graduation: May, 2015.

2. Bachelor of Engineering in Electronics and Communication Engineering.

   Birla Institute of Technology, Mesra, India.

   Graduation: December, 2010.

Experience

1. Research Assistant. Syracuse University. (2017-Present).
2. Teaching Assistant. Syracuse University. (2015-2017).
3. Web Developer. ITS Syracuse University. (2013-2015).
4. Technical Co-Founder. Digitizor Web & Media. (2011-2013).

Publications

1. **R. Laishram**, A.E. Sarıyüce, T. Eliassi-Rad, A. Pinar, S. Soundarajan. "Residual Core Maximization: An Efficient Algorithm for Maximizing the Size of the k-Core." SDM. 2020.

2. **R. Laishram**, J.D. Wendt, S. Soundarajan. "Sampling the Community Structure of Multiplex Networks." AAAI. 2019.

3. **R. Laishram**, A.E. Sarıyüce, T. Eliassi-Rad, A. Pinar, S Soundarajan. "Measuring and Improving the Core Resilience of Networks." WWW. 2018.

4. K. Areekijseree, **R. Laishram**, S. Soundarajan. "Guidelines for Online Network Crawling: A Study of Data Collection Approaches and Network Properties." WebSci. 2018.

5. **R. Laishram**, K. Areekijseree, S. Soundarajan. ”Predicted Max Degree Sampling: Sampling in directed networks to maximize node coverage through crawling.” INFOCOM. 2017.

6. K. Areekijseree, **R. Laishram**, S. Soundarajan. ”Max-node sampling: An expansion-densification algorithm for data collection.” Big Data. 2016.

7. **R. Laishram**, K. Areekijseree, S. Soundarajan. ”Predicted max degree sampling: Sampling in directed networks to maximize node coverage through crawling”. Big Data. 2016.

8. **R. Laishram**, K. Mehrotra, C.K. Mohan. ”Link Prediction in Social Networks with Edge Aging”. ICTAI. 2016.

9. **R. Laishram**, V.V. Phoha. "Curie: A method for protecting SVM Classifier from Poisoning Attack." 2016.

Extended Abstracts/Posters

1. X. Wang, **R. Laishram**, J.B. Brask, C. Marcelo, "Understanding Music with Higher Order Networks." *SFI CSSS Proceedings*. 2018.

2. S.J. Berkemer, S.L. Cheong, J. Edgerton, M. Kogan, **R. Laishram**, A.R. Pacheco, A. Shannon, M.D. Sweitzer, X. Wang, "A Study on Public Transport Mobility Flows in Singapore." *SFI CSSS Proceedings*. 2018.

3. S.L. Cheong, **R. Laishram**, S. Ojanpera, A. Shannon, X. Wang, "Industrial Economic and Spatial Clustering in Singapore." *SFI CSSS Proceedings*. 2018.

4. **R. Laishram**, A.E. Sarıyüce, T. Eliassi-Rad, A. Pinar, S. Soundarajan. "Improving Core Resilience of Networks under Random Edge Deletion." *CompleNet*. 2018.

5. **R. Laishram**, S. Soundarajan. "Evaluation of Community Similarity based on Hierarchical Distance." *NERCCS*. 2018.

Under Review

1. **R. Laishram**, J.D. Wendt, S. Soundarajan. "MCS+: An Efficient Algorithm for Crawling the Community Structure in Multiplex Networks".

2. X. Wang, **R. Laishram**. "Representing and Understanding Music with Higher-Order Networks".

3. **R. Laishram**, J.D. Wendt, S. Soundarajan. "NetProtect: Network Perturbation to Hide Target Nodes".

4. **R. Laishram**, A.E. Sarıyüce, T. Eliassi-Rad, A. Pinar, S. Soundarajan. "The Resilience of k-Core Structure to Missing Data".

Invited Talks/Presentations

1. Chesapeake Large-Scale Analytics Conference (CLSAC). "Sampling the Community Structure of Multiplex Networks". 2019.

2. Graph Exploitation Symposium (GraphEx). "Sampling the Community Structure of Multiplex Networks". 2019.

3. SIAM Conference on Computational Science and Engineering (SIAM CSE). "Measuring and Improving the Core Resilience of Networks". 2019.

Miscellaneous

1. **Education:** Complex Systems Summer School, Santa Fe Institute (2018).
2. **Reviewer:** TKDE (2018,2019); CIKM (2019), IEEE Access (2019), NetSciX (2018).