CORE STRUCTURE AND INFLUENCE IN SOCIAL NETWORKS

BY

PRIYA GOVINDAN

A dissertation submitted to the

School of Graduate Studies

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

For the degree of

Doctor of Philosophy

Graduate Program in Computer Science

Written under the direction of

S. Muthukrishnan

and approved by

New Brunswick, New Jersey

January, 2018

ABSTRACT OF THE DISSERTATION

Core Structure and Influence in Social Networks

by PRIYA GOVINDAN

Dissertation Director:

S. Muthukrishnan

Structure of social connections and interpersonal dynamics based on user behavior shapes the culture, politics and economics of the world. Building consumer products and services, require a good understanding of the human social behavior, computational tools to analyze it and efficient techniques to model and predict it.

In this dissertation we study the graph structure of interactions at the community level and the social influence between users at an individual level in a social network. Our contributions are in the form of efficient algorithms, theoretical and experimental analysis and visualizations.

At the community level, k-core decomposition has been used in many applications to find dense regions in a graph. We present a space efficient approximate algorithm to compute k-core decomposition using $O(n \log d)$ space approximate algorithm to estimate k-cores in graphs, where n is the number of nodes, and d is the maximum degree. Our experimental study shows space savings up to 60X with average relative error less than 2.3% as compared to the in-memory method. Analogous to k-core, k-peak decomposition finds centers of dense regions in a graph. We present analysis of the k-peak decomposition and show that it outperforms k-core decomposition in applications such as community detection. In addition to that, we present a graph visualization technique, that uses both k-core and k-peak decomposition to give a global mapping of the graph.

At the individual level, we study the influence of users on one another based on the posts they share and the feedback received. H-index, used as a measure of impact in academic settings, can also be used as a measure of influence in online social interactions. The high volume and rate of online social interactions make in-memory computations challenging. We present algorithms to compute h-index in various streaming settings and get a $(1 \pm \epsilon)$ estimate of the h-index with sublinear, ie, polylog or even $O(1)$ space.

Thus, we present efficient algorithms, analysis and visualizations in an effort to study the group level structure of interactions and individual level influence, among users in social networks.

Acknowledgements

I am incredibly grateful to my advisor, Prof. S. Muthukrishnan, for his guidance and support. He encouraged me to ask hard questions and think from first principles. He inspires excellence in research. I am very fortunate to have an advisor who is supportive and believed in me through the highs and lows. Muthu, a huge thank you!

I'd like to thank my committee Prof. Badri Nath (Rutgers University), Prof. Laks Lakshmanan (University of British Columbia) and Prof. William Steiger (Rutgers University), for their advice and feedback on my dissertation. A special thank you to Prof. Steiger for his support as a graduate director. I am also grateful to the Computer Science department at Rutgers for providing a supportive graduate school environment. I have been fortunate to work with, and learn from, many brilliant and wonderful co-authors. Special thanks to Sucheta Soundarajan for being a true collaborator and friend. Thanks also to Ruobing Chen for her mentorship at my internship at Bosch.

Prof. Tanya Berger-Wolf at University of Illinois at Chicago was my masters thesis advisor. She introduced me to research and encouraged me to pursue a PhD. Over the years, she has continued to be a source of inspiration and strength. Tanya, thank you!

I am fortunate to have made great friends at Rutgers, who shared the graduate school experience with me. Amruta Gokhale, Chathra Hendahewa, Darja Krushevskaja, Fatma Durak, Long Le, Morteza Monemizadeh, Parul Pandey, Qiang Ma, Rezwana Karim, Sucheta Soundarajan, Vaibhavi Kulkarni - graduate school would not have been the same without you. Parul, thank you for your consistent support and encouragement through the final stretch of my PhD. Darja, thank you for always being there. Qiang, thank you for your constant support and your infectious positive outlook. Many thanks to Dan Macheret, Devshri Bhangaonkar, Habiba, Mayank Lahiri, Paul Varkey, Prashi Jain, Samira Hozeifi, Sudhir Raghavan, for your support and encouragement through the many years.

I am deeply grateful for a loving, supportive family. Needless to say, I owe them more than I can fathom. As a woman with a successful career, my mother set an early, strong and positive precedent for me. My father instilled in me the courage to take on challenges. Prashant, my brother, has played a significant role in shaping the way I think. This PhD, and much of my career, is due to his rock-solid support and faith in me. Amini chechi, thank you for the many roles you have played and countless ways in which you have shaped me. Thank you Suman, for always being there for me.

To the many teachers, friends and family, who have guided, stood by and cheered me on, that I may have failed to mention here, thank you!

Dedication

To my family.

Table of Contents

Chapter 1

Introduction

1.1 Introduction

Around 130,000 years ago human ancestors lived in groups and with resources shared among neighboring groups [52]. This was one of the earliest evidence of social behavior among hominids. Fast forward to current day, our social behavior has certainly evolved since our hunter-gatherer days. We still form and rely on our communities, but the structure and dynamics of our social networks have become highly complex. Our social structures and dynamics shape everything from our daily personal experiences to culture, politics and human progress .

Going back in time once again, but to a more recent past, consider the world we were in, in early 2000s. We communicated mostly by phone, email, postal mail or in person. The phone companies would have known who, presumably, knows whom, from the call logs. The email service providers could have studied how information was passed on. But, apart from conducting surveys, it would have been difficult to study how people formed groups and the structure of their communities. Besides, without the knowledge of what information people shared with others and how people responded to each other, it would also have been difficult to understand the interpersonal dynamics such as that of influence, trust and authority [66], [15].

The online social networks available today, such as Facebook, Twitter and LinkedIn, allow us to study the structure of social connections and the interpersonal behavior. The key aspects that differentiate these services from the communication services available in the past, are as follows: (1) Users can make explicit connections with other users (2) They can share content such as opinions, photos or videos. (3) Other users can respond to the content by 'liking', commenting or passing it on to their contacts. Given such data, we can try to understand the structure of communities based on social connections. The dynamics of the relationship between users can be further studied based on content sharing and responses received.

As of 2017, more than 3 billion people, i.e. 40% of the world's population, use one of the online social networks [40]. As of January 2017, roughly 500 million tweets are posted on Twitter each day [6]. Thus, the size of data and the complexity of the computations, presents algorithmic challenges in the study of social networks. In this dissertation, we present algorithms, analysis, experiments and visualizations, in an effort to understand the structure of communities and measure influence among users in social networks.

In this chapter, we discuss the current online social networks, give an overview of the interesting research questions in this area and conclude with an outline of the dissertation.

1.2 Social networks

Currently, there are several social network services [1] available, for different social interactions. For instance Facebook interactions tend to be casual and meant for personal relationships. On the other hand, LinkedIn is a platform for making professional connections. Blogging and microblogging services share many features of social networks. Some of the popular blogging and microblogging services include Twitter, Snapchat, Sina Weibo and Tumblr. Services such as Snapchat, Instagram, Pinterest and Flickr can also be considered as social networks, although such services encourage sharing of a specific type of content, such as photos or videos. Although, slightly different from social networks, messengers and discussion websites could also help in studying social interactions. Popular messenger services include Whatsapp, Skype, Wechat and Viber. Reddit, and Quora are popular discussion websites.

Most of the social network services allow users to make explicit connections to other users. The connection may be unidirectional (Example: Twitter) or reciprocated by default (Example: LinkedIn). Unidirectional connections often don't require approval by the other person, where as connections reciprocated by default require the other party to approve the connection.

The type of content allowed to be shared depends on the service provider and the purpose of the platform. Some services restrict the type of content allowed to be posted. For example: Only photos and videos can be posted on Flickr, Where as, some services allow a variety of content to be posted. For Example: On Facebook, one may post text, photos, videos or URLs.

[1]Note: In the rest of the this work, we use 'social networks' and 'online social networks' interchangeably, and networks and graphs interchangeably as well.

Social networks also vary in the type of response that another user might make. One of the most common responses is ‘Like’or ‘Favorite’. Some services also allow comments or replies, where another user might respond with their own content. The third kind of response is where user A posts something and user B shares or passes on user A’s post to other users, with or without adding, appending or modifying the content of user A’s post. An example of this type of response is ‘Retweet’ on Twitter or ‘Share’on LinkedIn.

Regardless of the differences in the kind of interactions on the above mentioned services, they share the common goal of encouraging users to share information and build communities.

1.3 Interesting questions

There are several perspectives and granularities at which one could study social networks. The content shared by the users is certainly a rich source of information to study user behavior. But for the purpose of this work, we will ignore the content and focus on the metadata of the activities. Further, we can classify the observations into two categories : (1) Social connections such as ‘Friends’ in Facebook, ‘Followers’ in Twitter and (2) Content sharing and responses from others.

Exploring the connections between users could reveal the structure of relationships among groups of people. Communities and cohorts are often hard to define and they change constantly. A question that has been of interest has been to identify communities and track them over time [23]. Other topics of interest include identifying hierarchies, understanding core-periphery structures and finding hidden groups. The structure of connections can also help understand some of the global phenomenon, such as information propagation and evolution of a social network over time [13], [11]. Evidently, the simple information of connectivity between users has potential answers to many interesting questions. In this work, will address the problem of finding cohesive groups of users, given the connections between them.

The second category of observations listed above, heps us understand the dynamics of interpersonal relationships. Based on the feedback and attention a user gets, we could explore concepts such as user influence, trust or authority. In this work, we will present methods to measure influence between users.

1.4 Overview of of the dissertation

This dissertation is divided into two parts:

1. Community level study of social networks
2. Individual level study of social networks

Chapters 2 through 4 cover the first part and Chapters 5 and 6 cover the second part of our work.

In the first part, we study the community level structure resulting from interactions between users. We choose to represent the interactions as graphs and explore the structures in the graphs. In Chapter 2, we give an overview of the problems and techniques in social graph analysis. In doing so, we present the context for graph decomposition methods and the research problems addressed in Chapter 3 and 4. In Chapter 3, we present our space efficient approximate algorithm to compute k-core graph decomposition. We support the theoretical results with experiments on real graphs. In Chapter 4, we analyze an alternate graph decomposition technique called the k-peak decomposition and show how it could be applied to solve problems such as community detection and maximizing information propagation. In this chapter, we also present *Mountain plots*, a new visualization technique that takes into account both k-core and k-peak decomposition to give a global mapping of graphs.

In the second part of this dissertation, we shift our focus to individual users in a social network. Chapter 5 gives an overview of user behavior in online social networks and discusses the interesting research questions in this context. This chapter also gives an overview of the results covered in Chapter 6. In Chapter 6, we focus on social influence of users and study how to measure this efficiently. We present algorithms under different settings, that uses as little as sublinear or even $O(1)$ space to compute influence of a user.

The last chapter, Chapter 7, concludes the dissertation.

Chapter 2

Community level study of social networks: Structure of social interactions

2.1 Introduction

As discussed in Chapter 1, this dissertation is in two parts. Chapter 2, 3 and 4, constitute the first part based on community level study of social interactions between users in a social network.

In this chapter, we discuss the representation of social interactions in the form of graphs and the interesting questions one might ask based on such a representation. We conclude this chapter with an overview of remaining chapters in this part of the dissertation.

2.2 Graph representation

In this work, we represent the interactions between users as a graph, $G(V, E)$, where V is set of vertices or nodes and E is the set of edges. A graph representation allows us to capture both the users and the relationships between them. The algorithms and analysis presented in Chapter 3 and 4 further assume an undirected and unweighted graph. Note that most of the work presented here can be extended to include directionality and weights on edges.

Depending on the nature of the data and the problem, observations of connections between users of a social network can be represented using other data structures as well. For instance, Foursquare allows users to post their current geo-locations. The relationship between users can be inferred from the physical proximity recorded by the service. However, in this case, a pair-wise representation such as a graph, might not be an ideal choice.

Since our goal is to study pair-wise user interactions in social networks, where users can explicitly interact and connect with other users, we choose to analyze and build algorithms based on a graph representation.

2.3 Interesting questions based on graph structure

By representing the interactions between users as a graph, we ignore the content of these interactions and focus on structure of interactions among the given set of users. This view of interactions allow us to look for patterns of interactions of groups of people. For example, we know that a group of closely-knit people, such as a family or a group of classmates, might know most of the other members in the group. In a graph representation, this translates to looking for densely connected subgraphs within the larger graph.

The types of research questions in this area include trying to understand patterns in real graphs, identifying real phenomenon such as communities, predicting future events such as new connections and building realistic synthetic models of social connections.

Motivated by trying to understand patterns in real graphs, graph decomposition is an approach to partition the nodes in a graph based of certain connectivity properties of the nodes. K-core graph decomposition is one of the most popular graph decompositions that finds dense regions in a graph and thus has many applications. In Chapter 3, we take a closer look at k-core decomposition and present a new space-efficient approximate k-core decomposition algorithm. In Chapter 4, we study another graph decomposition called k-peak decomposition and present its analysis, applications and a new visualization technique.

Some of the other important research questions based on graph representation of social connections include community detection [23], link prediction [30], synthetic graph models [2] [44], graph evolution [48], [11], [46] and sampling [72].

2.4 Overview of the results in this part

In Chapter 3, we present *NimbleCore* a space-efficient approximate algorithm to compute k-core in graphs. We support this contribution with theoretical analysis and experiments on real social graphs. In Chapter 4, we analyze the k-peak decomposition algorithm and define k-mountain based on the k-peak structures. We also present a novel graph visualization technique using k-mountains, study the global structure of real graphs using this visualization and show

how various applications could benefit from using k-peak decomposition.

Chapter 3

Space-efficient k-core decomposition: *NimbleCore*

3.1 Introduction

As discussed in Chapter 2, graphs are used to represent social connections, interactions, co-occurrences, and relationships between entities. Understanding the structure of graphs is key to building robust and efficient mining algorithms. Questions such as '*how central is a node?*', '*what is the most densely connected set of nodes?*', or '*which nodes lie on the periphery of the graph?*', are relevant in various graph-mining applications. Central to such applications is the concept of a k-*core* [65], which is defined to be the maximal subgraph such that all the nodes in the subgraph have degree at least k in the subgraph. Every node may be part of several k-cores, each corresponding to a different value of k. The *core number* of a node is the highest value k such that the node is a part of a k-core. Core numbers of nodes have wide applications in problems such as community detection, selecting nodes for network experiments, and modeling the spread of information. For example, in community detection it is helpful to know the set of nodes that are part of a k-core containing high degree nodes [59]. Another example is the use of k-core decomposition and core numbers to understand the structure of protein interaction networks [75].

Problem definition: Given a graph $G = (V, E)$, compute the core number C_u of each node u in V. Core numbers can be calculated by running an in-memory k-core decomposition algorithm [7], which takes $O(max(n, m))$ time and $O(m)$ space, where n and m are the number of nodes and edges in the graph respectively. Other existing algorithms perform k-core decomposition in a distributed setting [56], or estimate a node's core number from its local subgraph [58], or partition the graph and load the required partitions to compute k-cores [14]. Building the local subgraph [58] or finding the appropriate partition [14] can take large amounts of time and space, with significant numbers of I/O calls. In the worst case, these algorithms can

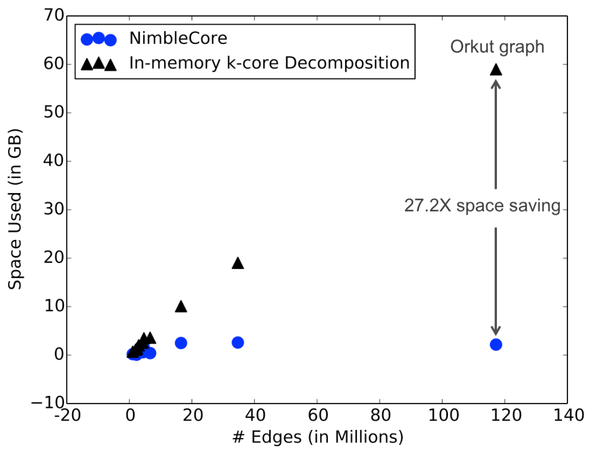


Figure 3.1: ***NimbleCore* gives significant savings (8.04X average ± 7.5) while accurately estimating core numbers (0.011 average error ± 0.018).**

require $O(m)$ space in memory. Similarly, the distributed algorithm by Montresor et al. [56] could require $O(m)$ space and up to n iterations over the data. As graph datasets become larger and larger, the need for algorithms that require much less than $O(m)$ space increases. The main **challenge** for estimating core numbers in large graphs is that the graph cannot be stored in main memory.

The major **contributions** of our work are as follows:

1. **Novel algorithm:** We present *NimbleCore*, a space-efficient external memory algorithm that accurately core numbers in a large graph without storing the graph.

2. **Theoretical analysis:** *NimbleCore* requires only $O(n)$ space for graphs with power-law degree distributions and $O(n \log d_{max})$ space for general graphs, where n is the number of nodes in the graph and d_{max} is the maximum node degree.

3. **Real-world experiments:** Experiments on large real-world large graphs from various domains show that *NimbleCore*'s achieves space savings on average of 8.04X (± 7.5) while returning core number estimates with average relative error of 0.011 (± 0.018).

The code for *NimbleCore* is available at `http://eden.rutgers.edu/~priyagn/code.html`

The remainder of this chapter is organized as follows. Section 3.2 gives the background and notations used in the chapter. In Section 3.3, we present our proposed algorithm *NimbleCore* and its theoretical analysis. Section 3.4 discusses experiments, which show the efficiency and effectiveness of *NimbleCore*. Section 3.5 reports on the related work. Section 4.8 concludes the chapter.

3.2 Background

Table 3.1 lists the notation used in this chapter. We are given a graph $G = (V, E)$. A node $u \in V$ has degree d_u; its neighboring nodes are in Γ_u (i.e., $\Gamma_u = \{v \in V \text{ s.t. } (u, v) \in E\}$).

Definition 1. k***-core****: Given graph* $G(V, E)$*, the* k*-core is the maximal subgraph such that all the nodes in the subgraph have degree at least* k *in the subgraph.*

Definition 2. ***Core number:*** *The core number of node* u *is denoted by* C_u *and is the maximum value* k *of* k*-core in which* u *participates.*

Given the core numbers of a node's neighbors, its core number can be exactly calculated as follows:

Lemma 1. *Let* $\mathcal{S}_u$ *contain the core numbers of neighbors of node* u*, sorted in descending order.*

$$C_u = \max_{1 \le i \le d_u} (\min(S_u[i], i))$$

Proof. See [56] [58] for the proof. ■

From the above lemma, we can state that the core number C_u of a node u is the largest value k such that there are at least k neighbors with degree at least k. Note that the above computation of C_u is equivalent to the definition of h-index [31] used to measure the productivity and citation impact of a scholar. The h-index of a scholar with N papers, is defined as the highest value h such that the scholar has at least h papers with h or more citations. We can formally generalize the definition of h-index on a list of values as follows:

Definition 3. *The* h*-index of a list of positive integers* L *sorted in descending order is defined as*

$$h(L) = \max_{1 \le i \le length(L)} (\min(L[i]), i))$$

where $L_i \in \mathbb{Z}_{>0}$.

Given Lemma 1 and Definition 3, we can state that node u's core number is equal to the h-index of the core numbers of u's neighbors (see Figure 3.2a for a toy example). We will refer to the computation in Lemma 1 as h-index. The following lemma shows that for a node u, the h-index of the upper bounds on the core numbers of u's neighbors is an upper bound on the core number of u.

Lemma 2. *Let ψ be a function such that $\psi(x) \geq C_x$, for $x \in V$. Let $\Gamma_u = \{v_1, v_2 ... v_{d_u}\}$ be the neighbors of node $u \in V$. Let Ψ be the list of $\psi(v) \forall v \in \Gamma_u$ in descending order.*

$$C_u \leq \max_{1 \leq i \leq d_u} \left(\min \left(\Psi(v_i), i \right) \right).$$

Proof. See [14]. ■

Because a node in a k-core must have degree at least k, the degree of a node is a naive upper-bound for the node's core number (i.e., for a node u, $C_u \leq d_u$). For graphs that can be stored on a single machine, Montresor *et al.* [56] show that we can use Lemmas 1 and 2, begin with degree as a naive upper-bound estimate for a node and iterate over the set of edges to compute the upper-bound estimate of each node's core number. They show that these values will converge to the true core number of each node. However, for graphs that cannot be stored in memory, this is not feasible.

3.3 Proposed Approach: *NimbleCore*

This section is organized as follows. We first provide an overview of *NimbleCore*. Next, we describe its binning strategy, followed by its stopping condition. We then present the entire algorithm, and wrap-up the section by discussing a theoretical analysis and extensions of *NimbleCore*.

3.3.1 Overview of NimbleCore

We present an iterative external-memory algorithm, *NimbleCore*, that estimates each node's core number by binning its neighbors' core-number estimates and then estimating the h-index using these binned values. In other words, rather than storing a complete vector of core-number

Notation	**Description**
V	set of nodes
E	set of edges
n	number of nodes
m	number of edges
d_u	degree of node u
d_{max}	maximum degree in the graph
C_u	core number of node u
$\hat{C_u}$	upper-bound on the core number of node u
$\bar{C_u}$	lower-bound on the core number of node u
C_u^i	estimate of node u's core number in i^{th} iteration
Γ_u	list containing neighbors of node u
S_u	sorted list (in descending order) containing core numbers of neighbors of node u
k_{max}	maximum core number (a.k.a. degeneracy) in G
$h(L)$	h-index of list L
$L[i]$	the i^{th} element of the list L

Table 3.1: Notations used in the chapter.

estimates for each node's neighbors, we divide these neighbor estimates into separate core-number bins. This allows for a large reduction in space requirements.

NimbleCore iterates over the graph's edges. In the first iteration, it computes the degree of each node. In subsequent iterations, while the stopping condition is not satisfied (see Section 3.3.3), *NimbleCore* uses binning to keep approximate counts of the core numbers of neighbors of each node. At the end of each iterations, *NimbleCore* updates the core number estimates of all the nodes.

The two main aspects of *NimbleCore* are (1) the binning strategy and (2) the stopping condition. Bins are vectors that are maintained to count the frequency of items. For example, while creating a histogram, one would be required to count items that fall in each range of values. Similarly, *NimbleCore* uses bins to approximately count the frequency of core numbers of neighbors of a node. The binning strategy plays an important role because the number of bins determines the space requirement of *NimbleCore* and the assignment of bin sizes affects the error on the estimated core numbers. Lastly, a good stopping condition results in fewer number iterations, thus saving runtime.

3.3.2 Binning Strategy

In this section, we describe *NimbleCore*'s binning strategy, which requires $O(n \log d_{max})$ space. The error resulting from the binning strategy of *NimbleCore* is discussed in Section 3.3.5.

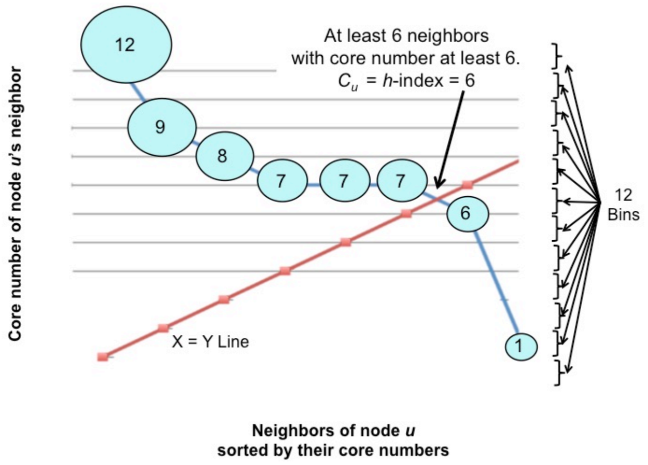
Lets first consider a naive binning algorithm, as shown in Figure 3.2a, where a bin is maintained for each distinct value of a neighbor's core number estimate, for each node. This naive binning strategy essentially build a histogram and thus requires $O(nd_{max})$ space, which is equivalent to $O(m)$; and hence not feasible for large graphs. One could use fewer counters (or bins) to build an approximate histogram. ***But, how many bins do we choose, and of what sizes, to get the best estimates of core numbers?***

Given the core numbers of a node's neighbors, *NimbleCore*'s binning, called *reverse log-binning*, selects larger bins for smaller values and smaller bins for larger values. In this way, *NimbleCore* allows for more granularity in the core-number estimates of u's neighbors that are closer to the estimated upper-bound core number $\hat{C_u}$.

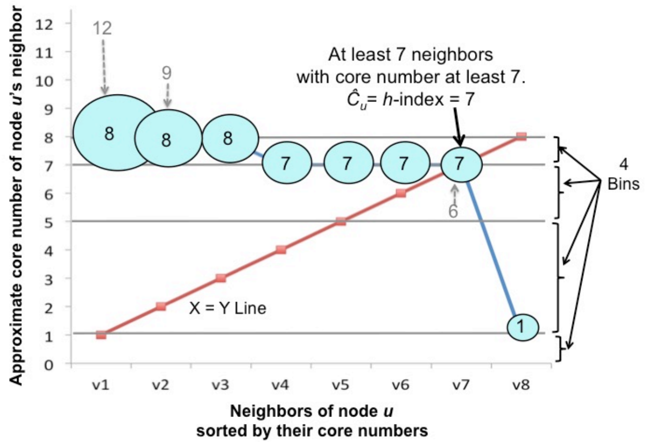
Reverse log-binning sets the number and sizes of bins (for the computation of approximate h-index) as follows:

- Let g be an upper bound on the estimate of node u's core number. g is initially set to the degree of u.
- Let B be the number of bins: $B = \log(g) + 1$.
- Set $BinCount$ and $BinValue$ of length B such that $\forall 0 \leq i < B$, $BinValue\ (i) = g - 2^{\log(g)-i} + 1$.
- A value $S_u[j]$ is counted in $BinCount\ (i)$ if $BinValue\ (i-1) < S_u[j] \leq BinValue\ (i)$.

Figure 3.2b demonstrates how *NimbleCore*'s reverse log-binning would estimate $\hat{C_u}$ using $BinCount$ and $BinValue$. As shown in the figure, there are two kinds of approximations done depending on whether the values are higher or lower than the current estimate. In Figure 3.2b, the current estimate, $\hat{C_u}$, is the degree, which is 8. Values higher than $\hat{C_u}$ are approximated to the last bin (i.e., the current upper-bound estimate). For example, 12 being higher than 8, is approximated to 8 and counted in the last bin corresponding to 8. Values that are at most



(a) Let the core numbers of the 8 neighbors of node u be [12, 9, 8, 7, 7, 7, 6, 1]. A node's core number is the h-index of the core numbers of its neighbors. The h-index is the value at which the descending order of a set of values intersect with the 'X=Y' line. A node's exact core number can be computed by storing the exact core numbers of its neighbors in arrays $\mathtt{BinValue}$ = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12] and $\mathtt{BinCount}$ = [1, 0, 0, 0, 0, 1, 3, 1, 1, 0, 0, 1]. This requires $O(d_u)$ space per node.



(b) Proposed idea: Reverse Log-binnng. A node's approximate core number can be computed by storing approximate values and counts of its neighbors's core numbers in arrays, $\mathtt{BinValue}$ = [1, 5, 7, 8] and $\mathtt{BinCount}$ = [1, 0, 4, 3]. This requires $O(\log(d_u))$ space per node.

Figure 3.2: Example: Calculating node u's core number using the core numbers of u's neighbors

the current estimate $\hat{C_u}$, are assigned to an appropriate bin as described above. For example, 6 is assigned to third bin and its value is approximated to 7.

Next, we show that the above approximation of core numbers of a node's neighbors and the subsequent h-index computation result in an upper-bound estimate of the node's core number. For values in S_u that are greater than the given upper bound of the h-index, we count them in the last bin (i.e., in $\text{BinCount}(B)$). Lemma 3 shows that this procedure does not change the

h-index result.

Lemma 3. *Let* L *be an ordered list (in descending order). Let* j *be the highest index such that* $L[j] > h(L)$*, where* $h(L)$ *is the* h*-index of the list L (see Definition 3). Furthermore, let* L' *be an ordered list (in descending order) such that* $\forall i : 0 \leq i < \mathtt{length}(L)$*,* $L'[i]$ *is set to* $h(L)$ *if* $i \leq j$*; otherwise,* $L'[i]$ *is set to* $L[i]$*. Then,* $h(L') = h(L)$*.*

Proof. If $\nexists j$ such that $L[j] > h(L)$, then L' and L are equivalent. Hence $h(L') = h(L)$. On the other hand, if $\exists j$ such that $L[j] > h(L)$, then the maximum value in L' is $h(L)$. So, $h(L') \leq h(L)$. The values at indices $i \leq j$ are replaced with $h(L)$ because j is the highest index (i.e., $L[j] > h(L)$). Thus, there are at least $h(L)$ values in L' that are at least $h(L)$; i.e. $h(L') \geq h(L)$. Hence $h(L') = h(L)$. ■

For values in S_u that are less than the given upper bound of the h-index, reverse log-binning assigns them to one of the bins and approximates their values to the upper bound of the range of values held by the bin. Hence for values in S_u that are less than the current estimate, their approximation is always an upper bound of their exact value. By Lemma 2, we know that the upper bounds on the core numbers of u's neighbors give the upper bound of u's core number.

Having small bin sizes for values that are close to but not larger than the current upper-bound estimate gives lower error in the h-index computation.[1] For example, for a node with degree five to have a core number of five, five of its neighbors must have core numbers equal to or greater than five. Thus, having smaller bins close to and not greater than five, would result in lower error in the h-index for our example node.

3.3.3 Stopping Condition

In every iteration, *NimbleCore* calculates better estimates for the core numbers of nodes than that in the previous iteration. Although more iterations give better estimates and lower error, this increases the runtime of *NimbleCore*. In this section, we discuss how *NimbleCore* decides to stop iterating over the list of edges, making a good trade-off between number of iterations and error.

[1]Recall that values larger than the current estimate are all placed in the last bin.

Our stopping condition is based on estimating the average relative error at every iteration and stopping when the estimated error drops below a threshold. The relative error (also referred to as 'error' in the chapter) of these estimates can be calculated as follows,

$$\text{Relative Error} = \frac{\hat{C_u} - C_u}{C_u} \tag{3.1}$$

where C_u is u's exact core number and $\hat{C_u}$ is the upper-bound estimate for u's exact core number. Since C_u is not known, we estimate the lower bound $\bar{C_u}$ and thus estimate the error.

$$\text{Estimated Relative Error} = \frac{\hat{C_u} - \bar{C_u}}{\bar{C_u}} \tag{3.2}$$

Since $\bar{C_u} \leq C_u$, Estimated Relative Error $\geq$ Relative Error. We compute the lower bound as follows: *Step 1:* Sample a set of wedges for each node. *Step 2:* Count the number of closed wedges i.e. triangles. *Step 3:* Given the number of triangles of node u, using Turan's theorem,[2] compute the size of the lower bound of the size of largest clique, r, that the node u participates in. If a node participates in a clique of size r, it is a part of the r-core and hence the lower bound of core number of u is r.

Similar to tightening the upper bound, we can tighten the lower bound estimate for a node's core number in each iteration. Better lower bounds give a better estimate of the error. In every iteration, for all $u \in V$, the lower-bound $\bar{C_u}$ and the upper-bound $\hat{C_u}$ of u's core number are updated. $\bar{C_u}$ helps estimate the error, as described above. Since the upper bound monotonically decreases and the lower bound monotonically increases, the estimated error also monotonically decreases. As the number of the iterations increases, the estimates of core numbers converge (i.e., stop changing). The goal of the stopping condition is to terminate *NimbleCore* when the change in the estimates falls below a negligible value. Specifically, when the difference in the estimated error, averaged over all nodes, between consecutive iterations falls below a threshold, *NimbleCore* terminates.[3]

[2]Turan's theorem [68] about a graph without a clique of size $r+1$ is as follows: Let G be any graph with n vertices and m edges, such that G does not have a clique of size $r+1$. Then $m \leq \left(\frac{r-1}{r} . \frac{n^2}{2}\right)$

[3]The experiments in Section 3.4 use a threshold of 0.01.

3.3.4 The Complete Algorithm

In the first iteration, the degree of each node u is calculated by incrementing a counter for u when an edge containing u is read. The second and the third iterations are used to count triangles, which are used in calculating a lower-bound estimate on core numbers. Specifically, in the second iteration, the neighbors of a node are sampled with probability p. A wedge is comprised of a pair of sampled neighbors. In the third iteration, each edge is checked to see if it closes any of the sampled wedges. A closed wedge is a triangle. From the triangles observed in the third iteration, *NimbleCore* estimates the lower bound on each node's core number and calculates an estimated error (described in Section 3.3.3). The subsequent iterations estimate and update the upper and lower-bounds of core numbers of the nodes. The method terminates when the drop in the estimated error falls below a threshold. The upper-bound estimates from the last iteration are then returned as the final estimates.

3.3.5 Performance Analysis of NimbleCore

The main challenge that *NimbleCore* is trying to address is that of space. By cleverly choosing bin sizes, *NimbleCore* is able to achieve a dramatic reduction in required space, as compared to existing methods. In this section, we provide bounds on *NimbleCore*'s space and time requirements.

Space

NimbleCore requires $O(n \log d_{max})$ space, which is less than the $O(m)$ space required for a k-core decomposition. Many real graphs have power-law degree distributions [22]. For these graphs, *NimbleCore* requires only $O(n)$ space.

Theorem 1. *NimbleCore requires* $O(n \log d_{max})$ *space.*

Proof. NimbleCore requires $\log d_u$ bins per node, where d_u is the degree of node u in the graph. So, it requires $O(\log d_{max})$ space per node, where d_{max} is the maximum degree. Given n nodes in the graph, the total space required is then $O(n \log d_{max})$. ■

Theorem 2. *NimbleCore requires* $O(n)$ *space for any graph with power-law degree distribution.*

Proof. Suppose a graph $G = (V, E)$ has $n = |V|$ nodes, such that the degrees of the nodes, arranged in decreasing order, follow a Zipf distribution. The i^{th} node with degree d_i is given as $d_i = i^R/n^R$ for all $i = 1, \ldots, n$, where $R < 0$. Space required by *NimbleCore* per node is $\log_\epsilon d_i = \log(i^R/n^R)$, where $\epsilon > 1$. The base of the log is 2 (i.e., $\epsilon = 2$). Total space used by the algorithm for computation of core numbers (in each iteration) is then:

$$\sum_{i=1}^{n} \log(d_i) = \sum_{i=1}^{n} \log(i^R/n^R)$$

Approximating the summation with integral we get,

$$\int_{x=1}^{n} \log(x^R/n^R)dx = -nR + R + R\log n$$

Since $R < 0$, $\forall n > 1$, $R \log n < 0$. Thus, the total space required by *NimbleCore* is less than $-nR$, i.e. $O(n)$. ■

Note that in Chapter 6, we present an algorithm [27] that further reduces the space requirement, to compute h-index, by only maintaining $O(\frac{1}{\epsilon})$ bins per node. Since, the space requirement for the whole graph is lower bounded at $O(n)$, in real graphs with power-law degree distribution, the further improvement might not matter.

Runtime

The runtime required by *NimbleCore* depends on the number of iterations. Each iteration takes $O(m + n \log d_{max})$ time.

Theorem 3. *NimbleCore requires* $O(m + n \log d_{max})$ *time per iteration.*

Proof. Each iteration requires $O(m)$ time to read the edges. After an iteration, *NimbleCore* takes $O(\log d_{max})$ time per node to estimate its core number. Hence, the time taken per iteration by *NimbleCore* is $O(m + n \log d_{max})$. ■

The number of iterations depends on the stopping condition (described in Section 3.3.2), which in turn depends on the estimates of the lower bounds on core numbers. Thus, the number of iterations, and hence the runtime for *NimbleCore*, depends on the quality of the lower-bound estimates.

3.3.6 Extensions

NimbleCore easily extends to directed graphs, by considering either the in-degree or out-degree to find the corresponding k-cores. *NimbleCore* can also be extended to weighted graphs $G(V, E, W)$, where $W(u, v)$ is the weight on edge (u, v) as described by Giatsidis et.al [25], by replacing the h-index function in Lemma 1 with

$$C_u = \max(\min_{1 \le i \le d_u} (S_u[i], \sum_{j=1}^{i} W(u, \Lambda_u[j])))$$

where Λ_u is the list of neighbors corresponding to the sorted list of core numbers of neighbors S_u.

3.4 Experiments

This section is organized as follows: datasets, baseline and competing approaches, and results. All experiments were run on a CentOS machine with 2.4 GHz (x 80) and 1024 GB memory, running Linux 2.6 and using Python 2.7.

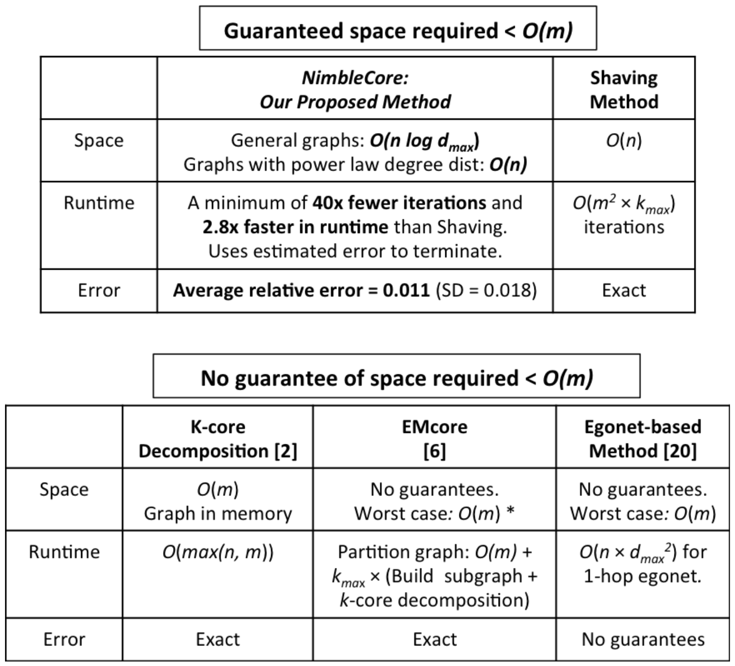
3.4.1 Datasets

Graph	**# Nodes**	**# Edges**	k_{max}	**Graph type**
Web-NotreDame	326K	1.1M	155	Unipartite
Web-Stanford	282K	2.M	71	Unipartite
Flickr	106K	2.3M	573	Unipartite
Amazon	403K	2.4M	10	Projected Bipartite
YouTube	1M	3.M	51	Unipartite
Web-Google	876K	4.3M	44	Unipartite
Wiki-Talk	2M	4.7M	131	Unipartite
Web-BerkStan	685K	6.6M	201	Unipartite
Cit-Patents	4M	16.5M	64	Unipartite
LiveJournal	4M	34.7M	360	Unipartite
Orkut	3M	117.2M	253	Unipartite

Table 3.2: Graphs used in our experiments. The graph's degeneracy is denoted by k_{max}.

Tables 3.2 lists the graphs from the Stanford Large Network Dataset Collection [49], used in our experiments. We consider a variety of graph types: *social networks* (Flickr, LiveJournal, Orkut, and YouTube), *Web graphs* (Web-BerkStan, Web-Google, Web-NotreDame and Web-Stanford), *citation graph* (Cit-Patent), *co-purchasing graphs* (Amazon), and *communication networks* (Wiki-Talk).

3.4.2 Baseline and Competing Methods



Guaranteed space required < *O(m)*

	NimbleCore: ***Our Proposed Method***	**Shaving Method**
Space	General graphs: ***$O(n \log d_{max})$*** Graphs with power law degree dist: ***$O(n)$***	$O(n)$
Runtime	A minimum of **40x fewer iterations** and **2.8x faster in runtime** than Shaving. Uses estimated error to terminate.	$O(m^2 \times k_{max})$ iterations
Error	**Average relative error = 0.011** (SD = 0.018)	Exact

No guarantee of space required < *O(m)*

	K-core Decomposition [2]	**EMcore [6]**	**Egonet-based Method [20]**
Space	$O(m)$ Graph in memory	No guarantees. Worst case: $O(m)$ *	No guarantees. Worst case: $O(m)$
Runtime	$O(max(n, m))$	Partition graph: $O(m)$ + k_{max} × (Build subgraph + k-core decomposition)	$O(n \times d_{max}^2)$ for 1-hop egonet.
Error	Exact	Exact	No guarantees

Figure 3.3: Comparison of other methods to *NimbleCore*. Unlike *NimbleCore*, several of the existing methods (listed in the right-hand side table) do not guarantee space less than $O(m)$. Shaving is the only method that provides a guaranteed space of less than $O(m)$, but it requires many more iterations than *NimbleCore* and has slower runtime than *NimbleCore*. ** Note that the lower space guarantee in EMcore [14] was shown to be incorrect by Khaouid et al. [41] and by Goodrich and Pszona [26].*

Figure 3.3 provides a summary of *NimbleCore* and the baseline and competing methods. Specifically, we compare *NimbleCore* against the following four methods.

(1) k-core decomposition: Batagelj and Zaversnik [7] propose an in-memory algorithm to find the k-cores of a graph, and thus the core numbers of nodes. To find a k-core, their method recursively deletes all nodes with degree less than k, until there are no nodes of degree k left. This method requires $O(m)$ space and $O(max(n, m))$ runtime.

(2) Shaving Method: k-core decomposition as described above can be implemented in an out-of-memory fashion by maintaining the current degree of each node in a vector of length n and by reading edges from the disk. To find a k-core, Shaving deletes nodes that have degree less than k and re-computes the degree of the remaining nodes, requiring two iterations on the edge list. As this step is performed repeatedly, a node that is removed when finding a k-core will have a core number equal to $k - 1$. This exact method requires $O(n)$ space and could require up to $O(m \times k_{max})$ iterations, resulting in a worst case time complexity of $O(m^2 \times k_{max})$.

(3) EMcore: Cheng et al. [14] propose an out-of-memory exact k-core decomposition method that partitions the graph into blocks and loads the required blocks of the graph into memory for the k-core decomposition. Unfortunately the space guarantees given in their paper was shown to be incorrect by Goodrich and Pszona [26] and by Khaouid et al. [41]. In worst case, EMcore could require $O(m)$ space.

(4) Egonet-based core number estimation: O'Brien and Sullivan [58] propose an approximate method to estimate a node's core number by using the induced subgraph up to h hops away from the node. Specifically, for each such subgraph, they run k-core decomposition to estimate the node's the core number. When the number of hops is set to the diameter of the graph, the core number of a node is exactly calculated. The performance of their algorithm in terms of space, runtime, and error depends on the number of hops considered for each node. In our comparisons, we use the 1-hop induced subgraph (i.e., the egonet) of a node.

3.4.3 Results

This section contains two parts. In part 1, we demonstrate the performance of *NimbleCore* in terms of space, error, runtime, and stopping condition. In part 2, we discuss observations about *NimbleCore* in terms of the drop in error on core-number estimates and error on core-number estimates of high-degree nodes.

Performance of NimbleCore

Q1: Space: How much space does *NimbleCore* save? Figure 3.1 shows that on average, *NimbleCore* saves 8.04X ($\pm$ 7.5) of the space as compared to the k-core decomposition [7] on the graphs in Table3.2. These observations support the theoretical results that real graphs, with

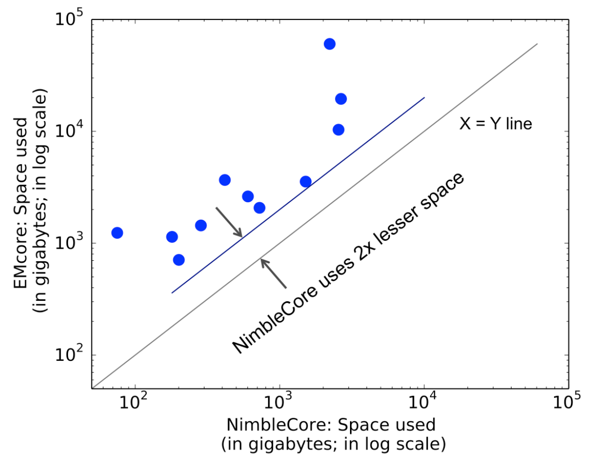


Figure 3.4: ***NimbleCore* uses $2\times$ lesser space than EMcore [14].** The space required by *NimbleCore* for a given graph is $O(n \log d_{max})$, while the space requirement for EMcore depends on the k-core being identified.

power-law degree distributions, the *NimbleCore*'s space complexity is $O(n)$ and for general graphs is $O(n \log(d_{max}))$ while the space requirement for k-core decomposition is $O(m)$ For example, for Orkut graph with 3 million nodes and 117.2 million edges, *NimbleCore* requires only 2.1 GB space, where as the k-core decomposition requires 59 GB space.

EMcore [14] is an out-of-memory exact k-core decomposition method, based on graph-partitioning that requires $O(m)$ space in worst case. Figure 3.4 shows that the space required by EMcore is at least $2\times$ higher than *NimbleCore* (on average $8\times$ higher, with $\pm$ 7.49). In contrast to EMcore, *NimbleCore* guarantees the space requirement to be $O(n \log d_{max})$, which is smaller than $O(m)$.

Q2: Error: How accurate are *NimbleCore*'s estimated core numbers?

The average relative error of core number estimates of the graphs in Table 3.2 is 0.011 ($\pm$0.018). To demonstrate the quality of the estimates returned by *NimbleCore*, we ran experiments on additional 48 graphs[4] from KONECT [47][5] with more than a million nodes and for which the *competition* (namely, the in-memory k-core decomposition) took up to 36 hours to terminate (in contrast *NimbleCore* took less than 2.5 hours to terminate, for all 48 graphs). Figure 3.5 shows that up to 60X space savings (14.6X average $\pm$10.8) with relative error less

[4]List of the additional 48 graphs: `http://eden.rutgers.edu/~priyagn/code.html`

[5]`http://konect.uni-koblenz.de/networks/`

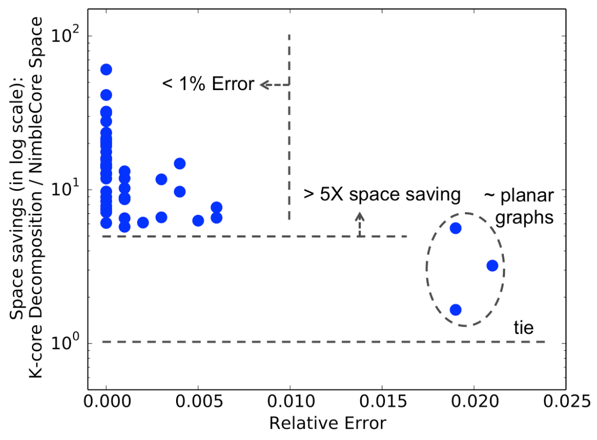


Figure 3.5: ***NimbleCore*gives upto 60.5X space savings for error less than 2.3%** Each point corresponds to one of the 48 large graphs from KONECT [47]. (0.002 average error ± 0.005 and 14.6X average space saving ± 10.8X).

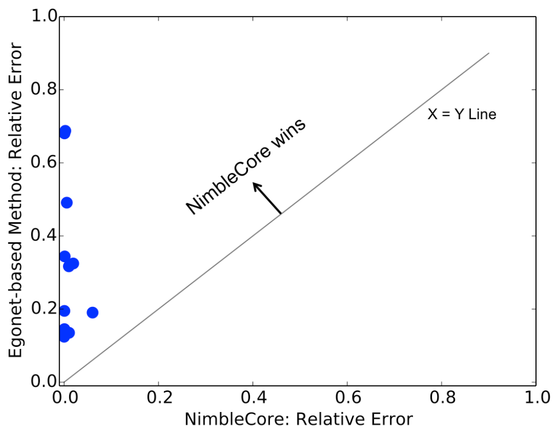


Figure 3.6: ***NimbleCore*'s error is less than the 1-hop Egonet-based method [58].** Each point on the plot represents a graph. The average ratio of error on *NimbleCore* to error on the Egonet-based method is 0.051 (± 0.097). (A tie would have an average ratio of error = 1).

than 2.3% (0.002 average error ± 0.005) was achieved. The three points to the right represent road networks (with maximum degree of 12), which are similar to planar graphs.

O'Brien and Sullivan's egonet-based method [58] extracts a h-hop egonet for every node and computes its core number based on the egonet. The value for h has to be set by the user. Constructing egonets for every node can be time-consuming depending on the value of h. Figure 3.6 compares the relative error of the 1-hop Egonet-based method with *NimbleCore* on graphs in Table 3.2. We observe that the average error of the Egonet-based method is 0.29 (with ± of 0.18), while that of *NimbleCore*'s is 0.011 (with ± of 0.018).

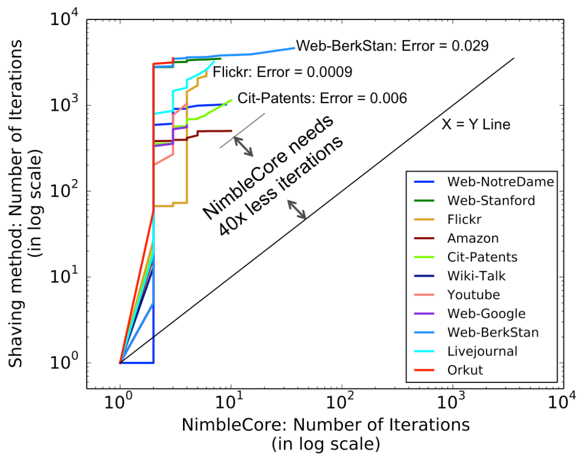


Figure 3.7: ***NimbleCore* requires fewer iterations than Shaving.** Each point on the line in the plot corresponds to the number of iterations required by *NimbleCore* and the number of iterations required by Shaving to obtain the same relative error. *NimbleCore* has 2.8× faster runtime than Shaving.

Q3: Runtime: How much faster is *NimbleCore* than the Shaving method? The Shaving method is an out-of-memory baseline that uses $O(n)$ space and iterates over the edges by reading them off the disk. Since *NimbleCore* also requires $O(n)$ space for graphs with power-law degree distributions and iterates over the edges from the disk, we compare the number of iterations required by the two methods. Figure 3.7 shows the number of iterations required by *NimbleCore* and the Shaving method to achieve the same error, on graphs in Table3.2. It shows that the Shaving method requires at least 40X more iterations over the edges than *NimbleCore*, to achieve the same error. In terms of wall-clock runtime, *NimbleCore* was found to be at least 2.8× faster than the Shaving method.

Q4: Stopping Condition: How effective is *NimbleCore*'s stopping condition? The stopping condition helps *NimbleCore* terminate in a small number of iterations, making a trade-off between error and runtime. Recall that the stopping condition depends on the estimated error. Figure 3.8 presents the number of iterations vs. relative error under two scenarios: (1) using *NimbleCore*'s stopping condition (See Section 3.3.3), and (2) using no stopping condition (i.e. terminating *NimbleCore* when there is no change in the core number of all nodes between consequent iterations). *NimbleCore*'s stopping condition provides an average speedup of 11.5× (±10.7) with an average difference in error of 0.012 (± 0.019). Note that the error for 'Web-NotreDame' graph is higher than 0 because *NimbleCore* without a stopping condition is still an

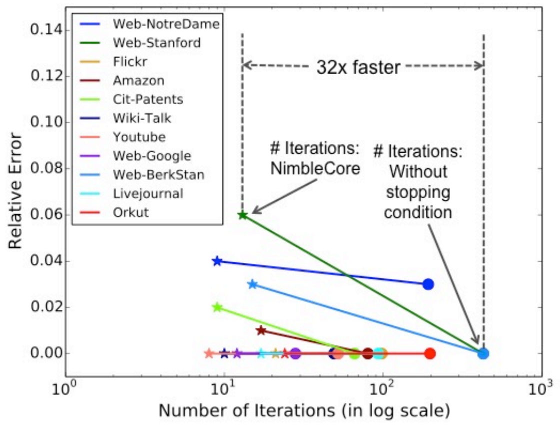


Figure 3.8: ***NimbleCore*'s stopping condition produces significant speed-up.** The stopping condition described in 3.3.3 helps terminate *NimbleCore* in fewer iterations. The average speed-up is 11.5× (± 10.7). The average difference (i.e. sacrifice) in error is 0.012 (± 0.019).

approximate algorithm.

Observations about NimbleCore

Observation 1: Error drops quickly over a small number of iterations. The first, fifth, tenth and twentieth iterations, respectively, give an error of 0.49 (± 0.19), 0.04 (± 0.03), 0.017 (±0.021) and 0.01 (± 0.017), on graphs in Table3.2. This demonstrates that even without a stopping condition, in real graphs, low error can be obtained by running *NimbleCore* for as low as 5 iterations.

Observation 2: The error of high degree nodes is low. In many applications [69] [59], it is useful to know the core number of high degree nodes. We found that the average error of *NimbleCore* on the top 1% of the high-degree nodes in the graphs in Table3.2, is 0.012 (± 0.008), as compared to an average of 0.011 (± 0.018) for the whole graph.

3.5 Related Work

Core numbers and k-cores of a graph, introduced by Seidman [65], have been shown to have many applications in problems such as community detection [25], finding dense subgraphs [60], designing network experiments [69], modeling spread in networks [43], predicting protein function [75] and graph coloring problems [53].

Batagelj and Zaversnik [7] proposed a k-core decomposition algorithm that requires $O(max(n, m))$ runtime and $O(m)$ space to perform k-core decomposition. Cheng et al. [14] presented a method to partition the graph and compute core number in a distributed fashion. O'Brien and Sullivan [58] introduced a recursive algorithm that computes core numbers for each node based on a h hop subgraph around the node. We compare *NimbleCore* against these three methods. See Figure 3.3 for a summary comparison and Section 3.4.3 for empirical comparisons.

Montresor et al. [56] proposed a distributed k-core decomposition algorithm, which is similar to *NimbleCore*. In [63], Saríyüce et al. propose algorithms for updating a k-core decomposition of a graph, as edge insertions and deletions are given as a stream, by storing certain subgraphs in memory. *NimbleCore* does not store any subgraph or list of neighbors of any node. In recent work by Khaouid et al. [41], they implement the algorithms in [7], in [14] and in a distributed setting. Goodrich and Pszona [26] present a method that returns an approximate degeneracy-ordering of nodes. Given the core numbers of nodes, a degeneracy-ordering is simply an ordering of nodes by their core number, but the core number of nodes cannot be obtained from a degeneracy-ordering.

3.6 Conclusion and Extensions

We have presented *NimbleCore*, a novel space-efficient external-memory algorithm, which estimates core numbers of nodes in graphs too large to be stored in main memory. Some of *NimbleCore*'s properties are as follows:

1. **Novel algorithm:** *NimbleCore* iterates over the edges and computes accurate estimates of core numbers in a large graph without storing the graph in memory.

2. **Theoretical analysis:** *NimbleCore* requires $O(n)$ space for graphs with power-law degree distributions and $O(n \log d_{max})$ for general graphs.

3. **Real-world experiments:** In real graphs, *NimbleCore* gave significant space savings (average 8.04X $\pm$7.5) and consistently low error (0.011 average error $\pm$ 0.018).

The code for *NimbleCore* is available at `http://eden.rutgers.edu/~priyagn/code.html`

There are several open questions and extensions to this work. Below, we list a few interesting open problems.

1. We have shown that *NimbleCore* gives low error in a small number of passes (See Section 3.4.3 in Chapter 3). Can the error over k iterations in the approximate *NimbleCore* or even in Montresor et. al's [56] exact method, be estimated?

2. For a given subset of nodes and an input of stream of edges, can we estimate their core numbers, better than $(1 \pm \epsilon)$ approximation, in one or maybe constant number of iterations?

3. Can k-core computation be distributed with known theoretical guarantees? In a given iteration, the computations at each node in*NimbleCore* , can be carried out in parallel. We could use a distributed model such as *MapReduce*, where in the *Map* phase creates two key-value pairs for each edge (u, v). The key would be the ID of one of the nodes and the value would the current core number estimate of the other node. In the *Map* phase we can collect the keys for a node, estimate h-index, and in the *Reduce* phase the updated upper bound for core number of each node could be returned. However, in a *MapReduce* framework, it is important to be able to estimate the number of passes it would take until convergence or to get a good estimate. Recently, Sariyuce et. al. [64] presented upper bounds to the number of iterations until convergence, in the above described approach. Can find a tighter bound, assuming certain graph properties, or modify the algorithm such that would give tighter bounds on the number of iterations? Note that, the challenges faced by this approach to using *MapReduce* to compute k-cores is related closely to the previous problem. However, another approach is to partition the graph and distribute the computations. Khaouid et al. [41] and Aridhi et al. [5] present alternate approaches to distributed k-core computation by demonstrating the performance of their methods through experimental results.

Chapter 4

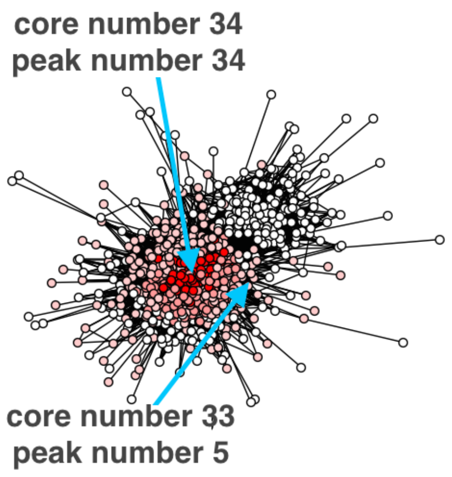
The k-peak decomposition: Mapping the Global Structure of Graphs

4.1 Introduction

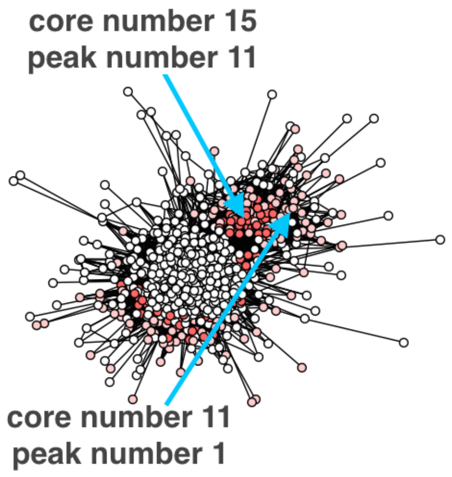
In recent years, there have been significant efforts to characterize the structure of real-world graphs, which requires an understanding of both global as well as local structure. It is thus interesting and useful to characterize the globally central parts of the graph as well as the local inter-dependence of nodes. The k-core graph decomposition has become a common way to describe the global structure of a graph. A k-core is the maximal subgraph with minimum degree k in the subgraph. This decomposition has well-understood properties [7, 65], and several efficient algorithms have been proposed for finding it, including algorithms for large graphs or streaming settings [28, 58, 63]. The k-core decomposition has achieved widespread success for applications that require finding a central, well-connected subgraph that, in a sense, captures the most important structure of the graph.

Weakness of the *k*-core decomposition: While useful, the k-core decomposition suffers from one important weakness, illustrated as follows: Suppose that a graph contains distinct and independent regions of different edge densities. In such cases, the k-core decomposition will fail to find a good global representation of the graph, instead focusing on the densest of these regions. If one wishes to find the central regions across the entire graph, the k-core would not be effective.

For example, consider the Facebook friendship graph of all the users in the world. In Iceland, 100% of the population has Internet access, while in Eritrea, only 1.1% of the population has Internet access. If one finds the k-core of the graph for some large value of k, this core will likely capture the important aspects of the Icelandic region, but none of the Eritrean region. Naturally, any method that finds a core or other type of dense subgraph will exclude many



(a) First mountain of Reed Facebook network.



(b) Second mountain of Reed Facebook network.

Figure 4.1: Mountains (distinct regions) obtained as a result of k-peak decomposition. The most central (highest peak number) nodes, peripheral nodes and nodes outside of the mountain are shown in dark red, in lighter shades of red and in white, respectively. Pruning the graph by core number would have found the peripheral nodes of the first mountain before finding the central nodes of the second mountain. Note that the second mountain consists of 2 components. (Best viewed in color.)

nodes, but the critical observation in this case is that *the structure of the dense region is independent of the structure of the sparser, excluded region.* If one lowers the value of k to capture the sparser region, too much of the dense region will be captured, negating the gains obtained by finding the core. Note that this observation is in contrast to the traditional notion of a network with strong core-periphery structure, in which a high k-core, while excluding peripheral nodes, would capture the essential structure of the entire graph. In this example, the k-core would be much more informative if one could set different values of k for different regions.

Moreover, the k-core decomposition does not capture the local structure in graphs. To understand the local structure, one might ask the following questions: *How does the core number of a node depend on those of its neighbors? What are the patterns of inter-dependence between nodes?*

The k-peak decomposition: To address this weakness, we re-state and analyze the k-peak decomposition, originally presented by Abello and Queyroi [1] as 'Vertex Core decomposition'. Intuitively, the k-peak is similar to the k-core decomposition, except that different values of k are used in different regions of the graph. The k-peak decomposition divides the graph into separate 'mountains', each of which represents a different region of the graph. Alternatively, one can view the the k-peak decomposition as a notion of centrality that gives equal consideration to nodes in sparser regions of the networks. For example, Figure 4.1 shows the two highest mountains in the Reed University Facebook network. Each mountain contains a central region (dark red), as well as nodes dependent on this region (lighter shades of red). We show that the k-peak decomposition algorithm requires $O((\sqrt{N})(M+N))$ running time for N nodes and M edges and present a novel visualization method that allows one to obtain a global picture of the network's structure with respect to the k-peak decomposition.

Additionally, we show how the k-peak decomposition can be used to gain a nuanced understanding of the relationships between a node and its region. To illustrate its value, we perform this decomposition on a multitude of graphs from different domains. Finally, we show how the k-peak can replace or supplement the k-core in a variety of applications.

Distinctions from related concepts: The k-peak decomposition is related to the concepts of community structure and core-periphery structure, but differs from these notions in important ways. A 'community' is generally thought of as a well-connected group of nodes, while a mountain in the k-peak decomposition is a region that may contain multiple communities. A mountain is thus something coarser than, but also fundamentally different from, a community, as it lacks the expectation of well-connectedness.

The k-peak decomposition is more closely related to the notion of core-periphery structure; indeed, the regions found by this decomposition can sometimes be thought of as core-periphery structures, and others have observed the presence of multiple core-periphery structures in a network [62]. However, the k-peak decomposition and corresponding mountain plot offer significantly more nuance and insight into graph regions than simply identifying them as core-periphery structures. In our descriptive study in Section 4.6, we show how the mountain plot can be used to identify narrow, 'column'-like structures, classical core-periphery structures, as well as many other patterns.

Our contributions are summarized as follows: **(1)** We define and analyze novel structures, such as 'Mountains' using the **k-peak graph decomposition** to understand the global structure of distinct regions within a graph, and present an **efficient algorithm** to find it. **(2)** We perform a **rigorous theoretical analysis** of the properties of the k-peak decomposition. **(3)** We create the 'mountain plot', a **new visualization technique** to understand a graph's global structure.

(4) We use the k-peak decomposition to **empirically analyze** the structures of 22 diverse real graphs. **(5)** We show how the k-peak decomposition can be used in various k-core-based **applications** to achieve improved results (e.g., up to an average of 53% improvement in spreading a contagion through a network).

4.2 Previous Work

The k-core was introduced by Seidman [65] as a concept to find densely connected subgraphs. Batagelj and Zaversnik [7] proposed an in-memory algorithm to compute k-cores in a graph. The wide interest in the use of k-core decomposition led to the development of efficient algorithms in distributed [56], streaming [63] and external memory [28] [58] settings. The well-understood properties of k-core and the efficient algorithms to compute it have encouraged its use in solving several problems. The k-core decomposition has been used in applications such as community detection [59], improving contagion [43], in choosing nodes for network experiments [69] and finding essential proteins in a protein-protein interaction network [76]. Liu et al. [50] proposed a similar modification to the k-core algorithm as our algorithm but their goal was to perform clustering.

There has been considerable research on studying the k-core decomposition, its process, and how to generalize it. Abello and Queyroi [1] originally proposed the 'Vertex Core decomposition', algorithm that we refer to as k-peak decomposition. We present an analysis of the decomposition, by further defining k-contours, k-peaks, k-mountains and analyzing their properties. We also present a new visualization technique called 'Mountain Plots', that use both k-core and k-peak decompositions. The k-core decomposition process has been shown to reveal an ordering of nodes based on the iteration at which it was 'peeled' from the graph [37]. There has been considerable research on generalizing and extending the definition of k-core to higher structures. A k-truss defined on an edge was proposed by Cohen [16]. More recently, Sariyuce et al. [21] further extended the generalization of k-core by defining it on a subgraph. The goal of all the previous graph decompositions based on k-cores was to find dense subgraphs, rather than globally distinct regions. The graph decomposition studied in this chapter finds distinct regions in a graph, that have properties similar to k-core. In [64], Sariyuce et al present an iterative approach to nucleus decomposition and give an upper bound to the number of iterations.

The global structure of a graph as a core-periphery structure was formalized by Borgatti and Everett [8] as a single core-periphery component. Rombach et. al [62] extended the definition to a more general model that allows for multiple core-periphery structures. An alternate model based on clustering co-efficient was proposed by Holme [34]. Zhang et. al [77] proposed a stochastic block model based approximate algorithm to find core-peripheries in graphs. The goal of these methods is to clearly define a core-periphery structure and design algorithms to find them. In contrast, the goal of our work is to *reveal distinct regions (mountains) in a graph*, using a known efficient algorithm (k-core). Although some of the mountains returned by k-peak decomposition are similar to core-periphery structures, the mountains also reveal interesting structures quite different from an ideal core-periphery. Moreover, rather than simply labeling or quantifying a region as a core-periphery structure, the k-peak decomposition allows an understanding of the specific shape and dependencies within that region.

Wang et al [71] presented a visualization technique to maps dense subgraphs. The mountain plots presented in this chapter are similar to their work with respect to the goal of plotting groups of nodes in a 2-dimensional plot, with the nodes ordered on the x-axis. But, our method differs in both the subgraphs (mountains) found and the ordering of the nodes in the plot. Specifically, the mountains found, depend on the k-peak decomposition algorithm and the ordering of nodes in a mountain in the plot depends on both the core and peak numbers. The plotting of both the core and peak numbers in the mountain plot gives information about the internal structure of the subgraphs that form the mountains.

4.3 k-peak Decomposition

Because the k-peak decomposition is based on the k-core decomposition, we first provide a review of key k-core definitions. We then introduce the k-peak decomposition and related concepts.

Recall that our goal behind the k-peak decomposition is twofold: first, to identify the distinct regions of the network and second, to obtain a deeper understanding of the relationships and dependencies between nodes.

Note that Abello and Queyroi [1] originally proposed the 'Vertex Core decomposition', which is equivalent to the k-peak decomposition that we discuss here. Our contribution is that we present the decomposition through newly defined structures of k-contours, and k-peaks, and further analyze the output of the algorithm by defining and studying k-mountains. Hence, note that the Algorithm 1, is equivalent to Algorithm 1 [1].

4.3.1 Preliminaries

A k-core is defined as follows:

Definition 4. ***k-core:*** *Given a graph G, a k-core is a maximal subgraph G_{k-core} such that each node in G_{k-core} has degree at least k in G_{k-core}.*

The *core number* of a node is the highest value k such that the node is a part of a k-core. The k-*core decomposition* is the assignment of core numbers to nodes. To find a k-core, the k-core decomposition algorithm recursively removes nodes with degree less than k. The graph *degeneracy* is the highest k such that there exists a k-core in the graph and the *degeneracy core* is the corresponding k-core. The set of all nodes with core number k is the k-*shell*. Alternatively:

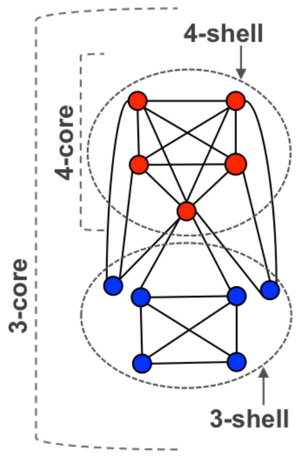
Definition 5. ***k-shell:*** *Given a graph G, a k-shell is the induced subgraph over the maximal set of nodes such that (1) the k-shell does not include nodes from any existing higher shells (i.e., p-shells where $p > k$), and (2) each node in the k-shell has at least k connections to nodes in the k-shell or higher shells.*

Note that the k-core is the induced subgraph of the union of j-shells for $j \geq k$.

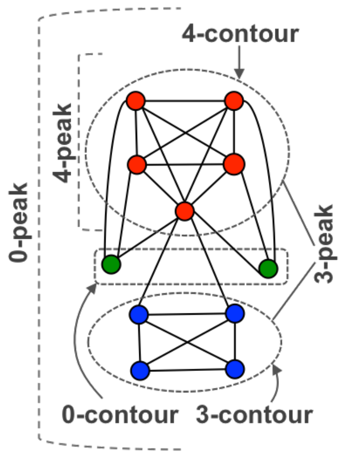
4.3.2 *k*-Peak Graph Decomposition

We posit that the global structure of a network can be viewed as a set of regions, each of which resembles a mountain with a central 'peak'. Mountains are drawn on topographical maps, with contours representing different elevations. Our terminology is based on this analogy.

First, we define a k-contour in a manner analogously to the k-shell:



(a) k-shells and k-cores



(b) k-contours and k-peaks

Figure 4.2: k-core and k-peak decomposition in a toy graph. The peak number of a node (the k-contour it belongs to) is at most its core number (the k-shell it belongs to).

Definition 6. ***k-contour:*** *Given a graph* G*, a* k*-contour is the induced subgraph over the maximal set of nodes such that (1) the* k*-contour does not include nodes from a higher contour (i.e., a* p*-contour where* $p > k$*), and (2) each node in the* k*-contour has at least* k *connections to other nodes in the* k*-contour .*

The difference between the k-contour and k-shell definitions is that in a k-shell, we count a node's connections to higher shells, whereas in a k-contour, we only count connections within the same contour.

The definition of a k-contour can be restated as follows:

Definition 7. ***k-contour:*** *Given a graph* G*, let* d *be the degeneracy of* G*. A* k*-contour of* G *is a maximal induced subgraph* $G_{k-contour}(V_{k-contour}, E_{k-contour})$ *such that*

$$G_{k-contour} = \begin{cases} G_{k-core} & \textit{if}\ k = d \\ (G \setminus \bigcup_{j=k+1}^{d} G_{j-contour})_{k-core} & \textit{if}\ k < d \end{cases}$$

If the degeneracy of the graph is d, then the d-contour is equivalent to the d-shell and the d-core is the induced subgraph of the d-contour.

The ***peak number*** of a node is the value k such that the node belongs to a k-contour.[1]

[1]Note that each node can belong to only one contour.

Definition 8. ***k-peak decomposition:*** *Given a graph* $G(V, E)$*, a* k*-peak decomposition is defined as the assignment of each node to exactly one* k*-contour.*

See Figure 4.2 for a toy example. Notice that the peak number of the two nodes with degree 3 (shown as green nodes in Fig 4.2b), is less than their core number.

Similar to the k-core, we define the k-peak as the induced subgraph of the union of j-contours, where $j \geq k$.

Definition 9. ***k-peak:*** *Given a graph* G*, a* k*-peak is the induced subgraph of the union of* j*-contours,* $\forall j \geq k$*.*

For a given graph G, the k-core decomposition is unique- i.e., each node has a single unique core number. In Theorem 4 below, we show that the k-peak decomposition is also unique- i.e., each node has a single unique peak number.

Theorem 4. *Given a graph* G *and* $k \in \mathbb{Z}_+$*, the* k*-peak is unique for all* $k > 0$*.*

Proof. Let d be the degeneracy of graph G. First, consider values of $k > d$. It is clear that there is no non-empty k-peak, because if such a k-peak existed, there would be a non-empty j-contour for some $j \geq k$. But then there would be a set of nodes, each with at least j connections to other nodes in the set; i.e., a j-core. But we assumed the d-core was the highest core of the graph. Contradiction, so the k-peak is empty, and so is unique.

Next, consider $k = d$. Then the d-core satisfies Definition 6 of the d-contour: (1) There are no higher contours, so no node in the d-core belongs to a higher contour, and (2) By Definition 4 of a k-core, each node in the d-contour has at least d edges to other nodes in the d-contour. Thus, the k-peak is simply the d-core, and so is unique.

Finally, consider $k < d$. Suppose for a contradiction that there exists some $k < d$ such that the k-contour is not unique. Select j to be the largest such value, so the j-contour is not unique but the $j + 1$-contour and higher contours are unique. Let S_1 and S_2, $S_1 \neq S_2$, be two sets that are both j-contours. Define $S = S_1 \cup S_2$. Since S_1 and S_2 satisfy Definition 6, $\forall u \in S$, u does not belong to a higher contour. Additionally, u must have at least j neighbors in either S_1 or S_2 (depending on which of these two sets it belongs to), and so it certainly has at least j neighbors in S. Thus, S satisfies conditions (1) and (2) of Definition 6 of a j-contour. However, because

S is a proper superset of S_1 and S_2, both S_1 and S_2 violated the maximality requirement in Definition 6. This is a contradiction, so the k-contour must be unique. Thus, the k-peaks are unique for all integers $k > 0$. ■

4.3.3 k-Mountains

Using the k-peaks, one can identify the k-mountains of the network, each of which can be thought of as a distinct regions within the larger network.

A k-mountain is defined as follows:

Definition 10. ***k-mountain:*** *A k-mountain is the induced subgraph of G containing all nodes $v \in G$ such that $C_v^k < C_v^{k+1}$, where C_v^k is the core number of node v after the nodes in the k-peak are removed (if a node itself is removed, define its core number after removal as 0).*

Intuitively, the k-mountain contains the k-contour as well as nodes whose core numbers are affected by its removal (after higher contours have been removed). Next, we prove that for every node in a k-mountain, there is a path from that node to the k-contour using only other nodes in the k-mountain. If the k-contour is a connected subgraph, then the k-mountain is also a connected subgraph. If the k-contour consists of multiple connected subgraphs, then the k-mountain may also contain multiple connected subgraphs.

The essence of the following proof is the observation that if edges are removed from a graph and the core number of some node changes as a result, then either that node was directly modified, or its core number changed as a result of a neighbor's core number changing.

Theorem 5. *Given a graph G, for every node u in a k-mountain of G, there is a path from u to a node in the k-contour of G that only uses other nodes in the k-mountain of G.*

Proof. The determination of whether a node belongs to the k-mountain depends only on its core number after the $k+1$-peak is removed vs. its core number after the k-peak is removed.

Define H to be the graph after the $k+1$-peak is removed. If G contains a k-contour, that means that the degeneracy of H must be equal to k. It thus suffices to consider only H, and show the theorem for the d-mountain of H, where d is the degeneracy of H.

Let H' be the resulting graph after nodes in the d-contour (degeneracy core) are removed from H. Define an *affected node* to be a node with a lower core number in H' than in H (i.e., a node in the d-mountain). Suppose for a contradiction that there is some affected node u, but there is no path from u to the d-contour using only other affected nodes.

Let S be the maximal connected (in H) set of affected nodes that contains u. Because there is no path along affected nodes from u to the d-contour, and S is connected in H, then there is no path from any node in S to the degeneracy core. S also clearly cannot contain any nodes in the d-contour itself.

Let T be the set of nodes that are adjacent to a node in S, but are themselves not in S. Because a node's core number cannot increase after the degeneracy core is removed, every node in T must have the same core number in both H and H', because otherwise S would not be maximal.

Suppose the core number of u in H is k_u, and define K be the k_u-core in H. Let $S_k = S \cap K$ and $T_k = T \cap K$. In H, each node in S_k had at least k_u connections to other nodes in $S_k \cup T_k$ (because it has at least k_u connections to K, and all of its neighbors are in either S or T). In H', the nodes in S_k still have at least k_u connections to $S_k \cup T_k$. This is because only edges within the degeneracy core were removed, and we have assumed that S (and thus also S_k) are disjoint from the degeneracy core. Thus, none of the edges incident to S were affected.

Let K' be the k_u-core in H'. $K' \subseteq K$ since $H' \subseteq H$. Note that none of the nodes in S_k can be part of K', because these are the nodes that had core number k_u in H and some lower core number in H'. We know that the core numbers of all nodes in T are unchanged. Hence the core number of the nodes in T_k is still k_u. Thus $T_k \subset K'$. But then $S_k \cup K'$ is a k_u-core. This is a contradiction, because this means that K' was not maximal, and thus was not the k_u-core of H'. Contradiction, so the claim is proved. ■

4.4 Computing the *k*-Peak Decomposition

The pseudocode for finding the k-peak decomposition of a graph G is presented in Section 4.4.1 (which is equivalent to equivalent to Algorithm 1 [1]) and proof of its correctness and analysis of its performance is given in Section 4.4.2.

4.4.1 Algorithm

The k-peak decomposition algorithm, originally presented by Abello and Queyroi [1], is shown in Algorithm 1. This algorithm iteratively removes the highest k-core of the graph and computes the core number of the remaining nodes at each iteration. The peak number of a node is the degeneracy of the graph before the node was removed. This process is carried on until the graph is empty.

Algorithm 1 k-peak decomposition

```
Require: The original graph $G(V, E)$
Ensure: $P$ {Peak Numbers of nodes}
1: while $V \neq \varnothing$ do
2:     $d$-contour $\leftarrow$ $degeneracy$ core of $G$ {Via $k$-core decomposition of $G$}
3:     $\forall u \in d$-contour, $P_u \leftarrow d$
4:     $V \leftarrow V \setminus d$-contour {Removing $d$-contour from $G$}
5: end while
```

4.4.2 Theoretical Analysis

Lemma 4. *Given graph $G(V, E)$, Algorithm 1 finds the k-peak decomposition.*

Proof. To show the correctness of Algorithm 1, we will show that it correctly assigns peak numbers to each node.

First consider all nodes u with peak number equal to the degeneracy d of the input graph G. Each of these nodes u is in the d-core, which is also the d-shell. Since there is no k-contour for $k > d$, by Definition 7, Algorithm 1 correctly sets the peak numbers of these nodes.

Next consider nodes with peak number less than d. Suppose for a contradiction that for some peak number $k < d$, the algorithm does not work correctly (i.e., a node with true peak number k is assigned an incorrect peak number). Consider the largest such value k, so that the algorithm correctly assigns peak numbers to all nodes with peak number above k. Let G_k be the graph remaining after the $k+1$-peak (which by our assumption is correct) is removed. Let d_k be the degeneracy of graph G_k. There are two possibilities: $k > d_k$ or $k = d_k$ (if $k < d_k$, nodes in the degeneracy core of G_k would have been removed as part of the $k+1$-peak, so would not be in G_k).

Suppose $k > d_k$. Then no nodes are assigned peak number k (because the nodes in the degeneracy core of G_k are assigned peak number d_k.). If this is incorrect, then the true k-contour of G should be non-empty. By Definition 7, the k-contour is a k-core in G_k. This is a contradiction, because we assumed that the degeneracy of G_k is less than k.

Now suppose $k = d_k$. After removing the $k + 1$-peak, the algorithm finds the degeneracy d_k-core of G_k, and these nodes are assigned a peak number of $d_k = k$. We assumed that this set is not equal to the true k-contour of G. All nodes in the true k-contour of G must be contained in G_k, because otherwise they would belong to a higher contour, which is assumed to be correct. The true k-contour of G is a maximal set of nodes in G_k that have at least k connections to one another; but this is simply the k-core of G_k. Thus, the d_k-core of G_k is equal to the k-contour of G. This proves the correctness of Algorithm 1. ■

The next two Lemmas discuss the running time and space requirements of Algorithm 1. Let N and M represent, respectively, the number of nodes and edges in G.

Lemma 5. *Algorithm 1 requires* $O(\sqrt{N}(M + N))$ *time.*

Proof. Suppose that a graph has b non-empty k-contours with peak numbers $k = \{k_1, ...k_b\}$, of sizes $s_1, ...s_b$. Suppose this list of contours is ordered in descending order, so k_1 is equal to the degeneracy of the graph. These k_i values must be unique, non-negative integers.

Because there are b non-empty contours, we must have $k_1 \geq b-1$, so $s_1 \geq b$ (because each node in this contour has at least $b-1$ connections within the contour). Similarly, $k_i \geq b-i$ and $s_i \geq b-i$, for each i. Let N be the number of nodes in the graph. Then, $N = s_1+s_2+...+s_b \geq b + (b - 1) + ... + 1 = (b)(b + 1)/2 \geq (b^2)/2$. Rewriting, we get $b \leq \sqrt{2N}$, so there are at most $\sqrt{2N}$ contours. Finding each contour requires performing a core decomposition on the graph, requiring $O(N + M)$ time. Thus, finding the k-peak decomposition requires $O(\sqrt{N}(M + N))$. ■

Lemma 6. *The algorithm 1 requires* $O(M + N)$ *space*

Proof. Each iteration of the while loop in Algorithm 1 computes a k-contour by computing the k-core decomposition of the current graph. Finding a k-core decomposition of G requires

$O(M+N)$ space, which can be reused in subsequent iterations. The vector P requires $O(N)$ space. Thus, the space requirement at any iteration is at most $O(M+N)$. ■

Theorem 6. *Algorithm 1 returns correctly finds the* k*-peak decomposition of a given graph* G *in* $O(\sqrt{N}(N+M))$ *time using* $O(M+N)$ *space.*

Proof. From Lemmas 4, 5, 6, thus proved. ■

4.5 Proposed Visualization

By using k-peaks, one can gain a deeper understanding of the network structure. In this section, we present the *mountain plot*, a novel visualization technique that uses the k-peak decomposition to understand both the global and local structure of the network.

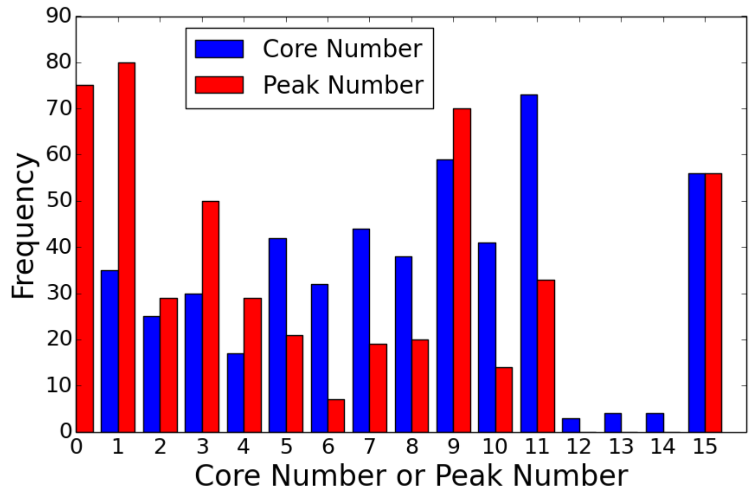
4.5.1 Mountain Plot

A major motivation behind the k-peak decomposition was understanding the dependencies between the core numbers of nodes. An obvious way to visualize these dependencies would be to plot histograms of the core numbers and peak numbers of the nodes in a graph G. An example of such a plot on the FB-Grad network is presented in Figure 4.3a. As expected, there are more nodes with higher core numbers than peak numbers. However, this simple plot suffers from two major problems: (1) It does not give us information on individual nodes, and (2) It fails to show the effect the removal of a k-contour on the rest of the graph.

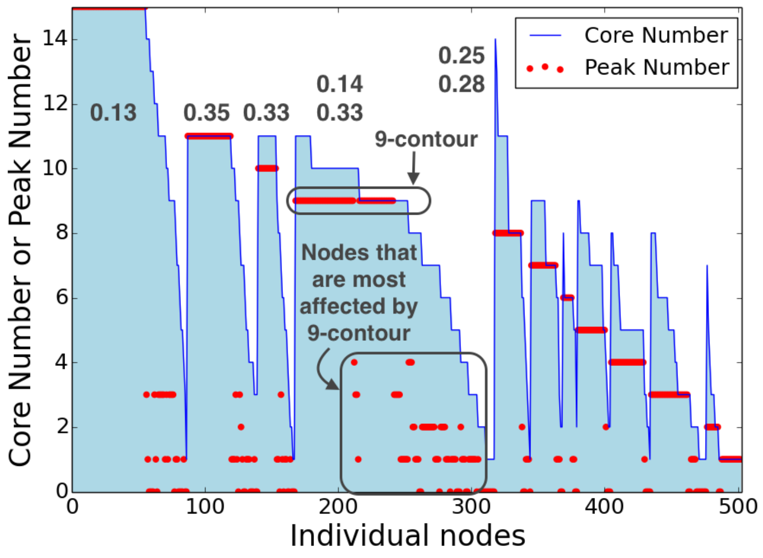
Our proposed visualization method, the *mountain plot*, provides a concise summary of the global peak structure of the network, depicts how the core and peak numbers of individual nodes differ, and allows for an understanding of node dependencies on k-contours.

To generate a mountain plot, we find mountains as in Section 4.3.3. Note that these mountains are overlapping. However, to keep the mountain plot concise, we associate each node with only one mountain.[2] We find k-contours as in Algorithm 1, where each contour forms the

[2]Note that we are not redefining the notion of a mountain, but are distinguishing between mountains that exist in the graph structure vs. mountains as they are plotted.



(a) Histogram of core and peak numbers, showing the difference in distribution of core and peak numbers.



(b) Mountain plot annotated with the core-periphery measure for each component in each mountain.

Figure 4.3: Facebook Grad network: The mountain plot gives more insight than the histogram. The mountains represent underlying distinct regions of the graph; e.g., the fourth mountain identified by the 9-contour, has a component with core-periphery structure (ρ=0.33, see Section 4.6.2) that is explained by the many nodes with low peak number (high dependency) in the mountain.

center of a mountain. We associate each node in the graph with the contour whose removal produced the greatest drop in its core number: these nodes constitute a plot-mountain. (Note that a node in the contour itself may not be assigned to that plot-mountain if it were more affected by an earlier plot-mountain.) The resulting plot-mountains are thus non-overlapping.

We sort these plot-mountains in descending order of the peak numbers of their associated contours. Within each mountain, we sort the member nodes in descending order of core number. For nodes with the same core number in a plot-mountain, we further sort them by their peak-number. This gives us an ordering of the nodes. On the x-axis of the mountain plot, we show the nodes in this order. Each integer along the x axis corresponds to the permuted ID of a node. We plot the core number of this node in blue, and the peak number in red. We then draw a line connecting all blue markers to outline the mountains.

For example, Figure 4.3b is a mountain plot of the FB-grad network. This plot shows us that the FB-grad network contains 12 mountains. The highest red markers within each mountain show the peak number of the contour corresponding to that mountain. The height of the mountain, in blue, shows the core numbers of the node within the mountain.

4.5.2 Interpreting a Mountain Plot

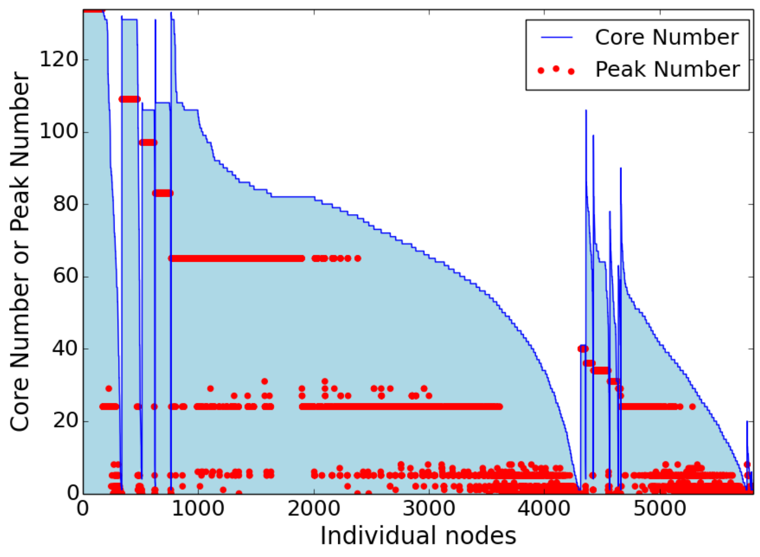
The mountain plot can give us the following information:

- The **number of mountains** in the mountain plot tells us the number of distinct regions in the graph
- The **mountains in the left of the plot** represent the most well-connected regions of the graph. The mountains towards the right corresponds to the sparser and more disconnected parts of the graph.
- The **height and width**, respectively, of a mountain increase with the core number and the number of nodes in the mountain. In the graphs that we observed, the nodes with the highest peak number in a mountain are those in the k-contour giving rise to the mountain.[3]
- The **difference between the core numbers and peak numbers** in a mountain gives information about the connectedness and dependencies of the subgraph represented by a mountain (See Section 4.6).

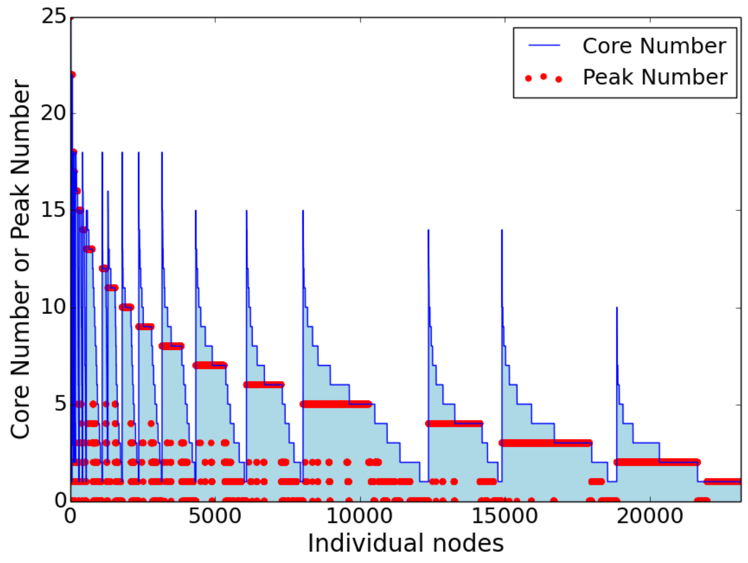
From the mountain plot, we can immediately get a sense of the global structure of the graph.

[3]Because we assign each node to the mountain that affected its core number most, it may be that the nodes in the k-contour giving rise to the mountain are not actually assigned to that mountain: however, in practice, this rarely happens, and mostly for the weaker, less well-connected mountains.

4.6 Descriptive Study



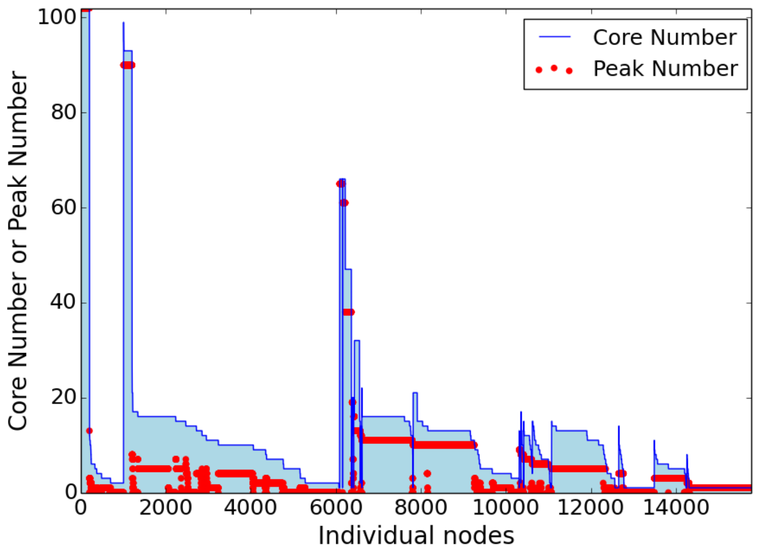
(a) Bio - yeast



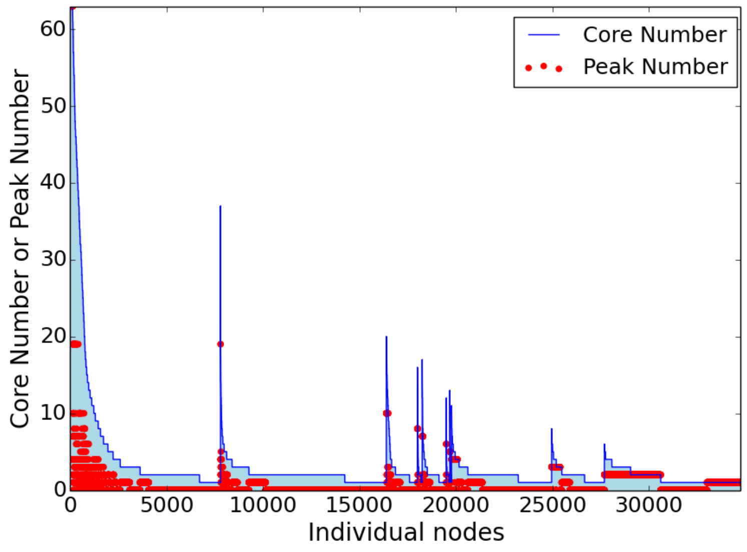
(b) Co-auth CondMat

Figure 4.4: Mountain plots for Protein3 and CondMat graphs. Observe the number, shapes, and sizes of mountains (i.e. regions).

We consider the 22 real graphs listed in Table 4.1, including social, co-authorship, email, web, and biological graphs. We begin in Section 4.6.1 by discussing structures observed in the mountain plots. In Section 4.6.2, we analyze the structures found in the detected mountains. We also consider 4 synthetic graphs, including random graphs and graph models meant to capture



(a) Web - google



(b) Internet topology

Figure 4.5: Mountain plots for Google Web and Internet topology graphs. Observe the number, shapes, and sizes of mountains (i.e. regions).

real-graph properties. In Section 4.6.3, we present the mountain plots of the synthetic graphs and discuss k-peak structures found in them.

The mountains found by the k-peak decomposition sometimes exhibit core-periphery structure [8]; however, the mountain plot allows for substantially more nuance in the way that we understand these graph regions (i.e., by examining the shapes of the mountains), rather than simply labeling them as core-periphery structures. Depending on the network structure, the

	Graph	$\|V\|$	$\|E\|$	d	ρ_G	ρ_5	ρ_{10}
Bio	Protein1 [47]	1,702	3,155	7	0.06	0.15	0.15
	Protein2 [47]	2,239	6,432	10	0.03	0.20	0.20
	Protein3 [42]	5,808	362,421	134	0.09	0.22	0.17
Co-auth	GrQC [49]	5,241	14,484	43	0.03	0.24	0.25
	HepTH [49]	9,875	25,973	31	0.02	0.25	0.24
	HepPH [49]	12,006	118,489	238	0.07	0.29	0.28
	AstroPh [49]	17,903	196,972	56	0.03	0.29	0.29
	CondMat [49]	23,133	93,439	25	0.02	0.24	0.23
Msg	Digg [47]	30,360	85,155	9	0.03	0.07	0.07
	Enron [49]	33,696	180,811	43	0.06	0.16	0.24
Social	FB Grad [54]	503	3,256	15	0.05	0.26	0.24
	FB Caltech [54]	769	16,656	35	0.14	0.05	0.05
	FB Reed [54]	962	18,812	34	0.14	0.10	0.08
	FB Ugrad [54]	1,220	43,208	47	0.09	0.05	0.03
	FB Baylor [54]	12,803	679,817	96	0.05	0.12	0.10
	Brightkite [49]	58,228	214,078	52	0.04	0.27	0.28
Tech	Gnutella [49]	26,498	65,359	5	0.01	0.19	0.19
	Internet [47]	34,761	107,720	63	0.08	0.18	0.26
Web	Foldoc [47]	13,356	91,471	12	0.02	0.02	0.06
	Google [47]	15,763	148,585	102	0.06	0.13	0.12
	Stanford [49]	281,903	2,312,497	71	0.04	0.28	0.40

Table 4.1: Statistics of the real graphs, varying in size and type. Here d is the degeneracy of G and ρ_G, ρ_5 and ρ_{10} are the core-periphery measures for G, average of the first 5 and 10 mountains respectively.

detected regions may *not* be core-periphery structures (e.g., as we will see, the CondMat network in Figure 4.4b with 'column' mountains). Additionally, although in practice the k-peak decomposition sometimes finds core-periphery-like structures, it is important to note that this is not its purpose: rather, the goal is to find distinct regions and understand patterns of node dependencies.

4.6.1 Mountain Plots of Real Graphs

In Figures 4.3b, 4.4 and 4.5, we present a mountain plot for representative social, biological, co-authorship, Internet and web graphs.

We make several observations from the mountain plots.

Number and Sizes of Mountains: First, the number of mountains identified by the k-peak decomposition corresponds to the number of distinct regions within the network, and the width of the mountains in the mountain plot shows the size of these regions. For example, Figure 4.4a shows us that the yeast biological network contains one large mountain, one medium-sized mountain, and several very small mountains. In contrast, Figure 4.4b shows us that the CondMat co-authorship network contains many medium-sized mountains; but these mountains appear on the right side of the plot, indicating that they are not associated with the highest k-contours. These mountains, while large, are not as well-connected as the smaller mountains on the left side of the plot.

Many of the social graphs we consider tend to have a small number of meaningful mountains. In contrast, the co-authorship graphs all had the same basic pattern as the CondMat graph in Fig. 4.4b: small, well-connected mountains on the left side, followed by weaker, larger mountains.

Shapes of Mountains: We observe two extreme types of mountains, with gradations in between: the 'column' type mountains (e.g., at the beginning of Figures 4.4a, and 4.4b), and the 'wide' mountains (e.g., the largest mountain in Figures 4.4a and 4.5a). A column mountain is typically formed of a large clique on which very few other nodes depend for their core numbers. In contrast, the wide mountains are the well-formed core-periphery structures, with a small core and a large, highly-dependent periphery.

Co-authorship graphs, such as in Figure 4.4b, tend to have column mountains. These mountains likely correspond to papers with many authors who tend to work together most of the time, thus forming a clique that is not well-connected to the rest of the network. These narrow column mountains very rarely exist in social graphs. Web graphs and Internet graphs as shown in Figure 4.5a and 4.5b, contain several mountains that begin with a column, but suddenly expand into a wide base at the bottom (e.g., the first two mountains in 4.5b). These represent tightly-connected structures on which many low-core-number nodes depend: these mountains likely represent cliques with low-degree nodes on the periphery, as opposed to a gradually expanding core-periphery structure like we see in Figure 4.4a.

Social networks, as in Figure 4.3b, tend to have one or two large, meaningful core-periphery structures, and a collection of mountains that are closer to column-like. This latter type of mountain (e.g., the first mountain in Figure 4.3b represents a tightly-knit group of individuals with a small number of additional high-core-number individuals (contrast this with the Web graph mountains, which are tightly-knit groups with a large number of additional low-core-number nodes).

By looking at the mountain plot, we get a sense of the overall connectedness of the network as a whole: does it contain distinct regions or is it one cohesive structure?

Difference between a node's core and peak number: For each node, the mountain plot tells us the difference between that node's core number and peak number. This difference is simply the distance between the appropriate blue line and red points. This difference tells us the extent to which a node's core number is dependent on higher k-contours: the larger the drop, the more dependent the node is. For example, in the first mountain in Figure 4.3b, we see nodes with core numbers 9-13, whose peak numbers dropped to 3 and 2, due to the removal of the degeneracy core of the graph. On the other hand, in the fourth mountain, while there are nodes with low peak number, these nodes tended to have a low core number to start with: thus, this mountain contains fewer high-core-number nodes that are highly dependent on the mountain's k-contour. For another example across graphs, compare the core and peak numbers of nodes in the the first mountain in Fig 4.5b (Internet topology graph) with that of the second mountain in Fig 4.5a (web-graph). In this example, although both mountains have a small tightly-knit core, the periphery (nodes in the mountain excluding the nodes in the k-contour) of the web-graph is more dependent on the k-contour for its core number than that in the Internet topology graph.

4.6.2 Core-Periphery Structure in a Mountain

Although finding core-periphery structures is not our goal, it is interesting to measure how well the detected regions match the ideal core-periphery structure. According to Borgatti and Everett's formulation, the core-periphery structure is represented by an idealized adjacency matrix in which the first r nodes form the core and the remaining s nodes form the periphery ($r + s = N$). Every node in the core is connected to every node in the graph, but nodes in the periphery are only connected to nodes in the core. To measure how well a graph exhibits core-periphery structure, one reorders the adjacency matrix so that the core nodes appear first, and then measures ρ, the Pearson's r correlation between the true adjacency matrix and the idealized adjacency matrix (in our mountains, the 'core' contains nodes in the associated contour, while the 'periphery' contains everything else). $\rho \in [-1, 1]$, with values above 0 indicating positive core-periphery structure.

Because the k-peak decomposition is premised on the notion that sparser graph regions can exhibit meaningful structure, we modify this measure by dividing the values in the matrix by $2M$, so that the sum of (weighted) edges in the idealized and true adjacency matrices are the same. We measure significance using a QAP test, by repeatedly randomly permuting the set of nodes in the core and periphery (while maintaining their sizes), and calculating the correlation between the permuted adjacency matrix and the idealized adjacency matrix: p is simply the fraction of times that the permuted adjacency matrix exhibits a greater or equal correlation than the true adjacency matrix.

The first five mountains in Figure 4.3b are annotated with their core-periphery ρ values. Note that the fourth and fifth mountains consist of two connected components, so two values are shown. While these values seem low as correlations, the overall ρ of the entire graph is only 0.05,[4] and so the mountains exhibit much greater core-periphery structure. These values are significant at the $p = 0.01$ level.

The average ρ values for the first 5 and first 10 mountains is given in Table 4.1 for all graphs. Note that the values vary substantially by network: although the k-mountains are almost always more core-periphery-like than the graph as a whole, some mountains exhibit strong core-periphery structure (e.g., the first 10 mountains of the Stanford graph), while others are much weaker (e.g., Google web in Fig. 4.5a).

4.6.3 Mountain plots of Synthetic Graphs

In this section we consider some of the synthetic graphs graph models and study their mountain plots.

We first consider Erdös-Rényi random graphs, where given nodes n and an edge between any two nodes may exist with probability p. In figure 4.6a, we present a mountain of a Erdös-Rényi graph with $10,000$ nodes and probability p of 0.001. As the figure shows, there is one mountain, corresponding to 7-contour. The peripheral nodes of this mountain belong to $0, 1$ and 2-contours. It is not surprising that an Erdös-Rényi graph has only one k-mountain, since

[4]To measure the ρ of an entire graph, we consider all values of k, and set nodes in the k-core to be the 'core' and nodes outside to be the periphery.

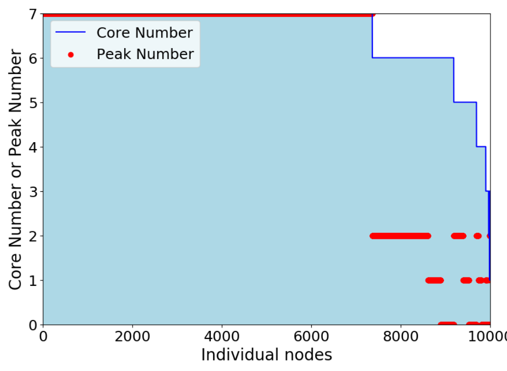
core-periphery structures are unusual in Erdös-Rényi graphs.

Next, we consider the Barābasi-Albert preferential attachment model [3], which preserves the power-law degree distribution often observed in real graphs. In this graph model, the probability of an edge from a new node to an existing node is proportional to the degree of the existing node, thus capturing the *'rich gets richer '* phenomena. Figure 4.6b shows the mountain plot of Barābasi-Albert preferential attachment graph with $10,000$ nodes, where each new node has 10 edges. Although this graph has a power-law degree distribution, it has only one mountain, composed of the 10-contour. Since the edges between high degree nodes is unlikely in Barābasi-Albert graph model, higher k-cores and k-peaks do not form. The core-periphery structure that is typical of a k-mountain in real graphs is unlikely to occur in Barābasi-Albert graphs, thus resulting in just one mountain of a 10-contour with no peripheral nodes.

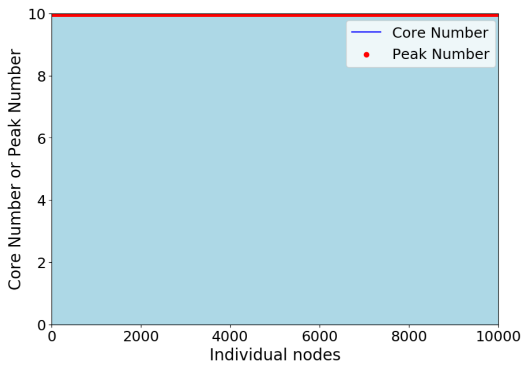
Watts-Strogatz [74] is a graph model that aims to capture the small world effect seen in real social graphs. The generative process beings by creating a regular graph where each node has degree k. It then rewires the edges with probability p. In figure 4.6c, we present the mountain plot of a Watts-Strogatz graph with $10,000$ nodes where each node has 5 edges initially, that are rewired with probability 0.8. This graph consists of exactly one mountain of 3-contour with peripheral nodes from 1 and 0-contours.

Hence, we observe that the three synthetic graph models described above do not show mountains similar to that observed in real graphs because they were not designed to capture the core-periphery structure of real graphs.

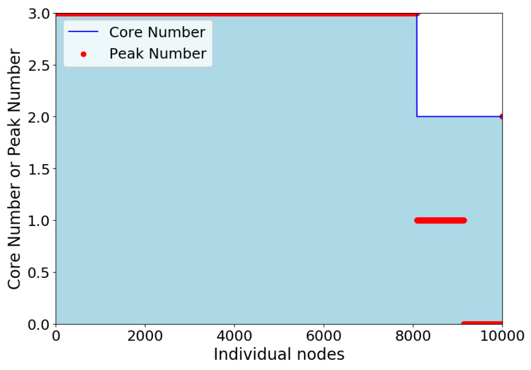
We now consider the *configuration model* [57], which is a random graph with a given degree distribution. We give as input the degree sequence of the Enron email graph, that has a power-law degree distribution. Figures 4.7a and 4.7b show the mountain plot and the degree distribution (in log -log scale) of the Enron graph. Figure 4.7b shows that the Enron graph has a power-law degree distribution and Figure 4.7a shows that the graph has several mountains. Observe that the mountains of the Enron graph in Figure Figures 4.7a has a tightly knit core (high peak numbers) and a relatively sparse periphery (lower core and peak numbers). Secondly, note that the core and peak numbers of the periphery are not too far apart. This shows that the nodes in the periphery are connected mostly to the mountain assigned to it and have very sparse connections to the rest of the mountains. Another way to interpret these mountains



(a) Erdös-Rényi random graph of 10, 000 nodes where any two nodes have an edge between them with probabiity 0.001



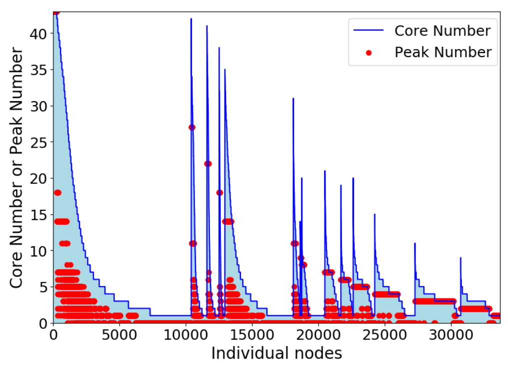
(b) Barähasi-Albert preferential attachment graph of 10, 000 nodes, where each new incoming node attaches itself to 10 existing nodes.



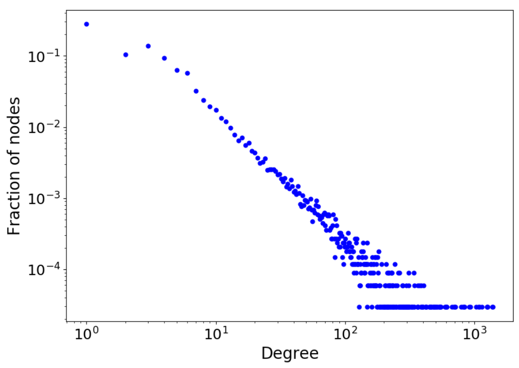
(c) Watts-Strogatz small world graph of 10, 000 nodes, where initially each node has 5 edges forming a regular graph. The edges are then rewired with probability 0.8.

Figure 4.6: Mountain plots for a Erdös-Rényi random graph, a Barähasi-Albert preferential attachment graph and a Watts-Strogatz small world graph .

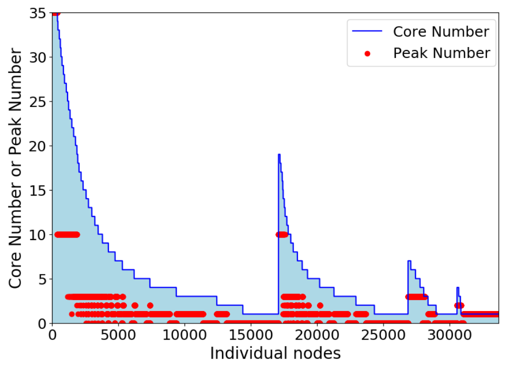
is that these core-periphery regions are fairly non-overlapping. Now, consider the configuration model graph generated using the degree distribution of the Enron graph. Figures 4.7c and 4.7d present the mountain plot and the degree distribution of such a configuration model graph. Fig 4.7d confirms that the degree distribution remains unchanged. The mountain plot of this graph in Fig 4.7c, shows that random graph was able to create similar core-periphery structures as in the original Enron graph i.e. tightly knit core (small set of nodes with high peak numbers) with a large loosely connected periphery (larger set of nodes with low peak numbers). However, as the mountain plots demonstrate, this instance of the configuration model random graph fails to capture the number of mountains, since there are four mountains in Fig 4.7c as compared to more than 10 in Fig 4.7a. Also note that the peripheral nodes in the mountains in the random graph, as shown in Fig 4.7c, has a larger gap between their core and peak numbers



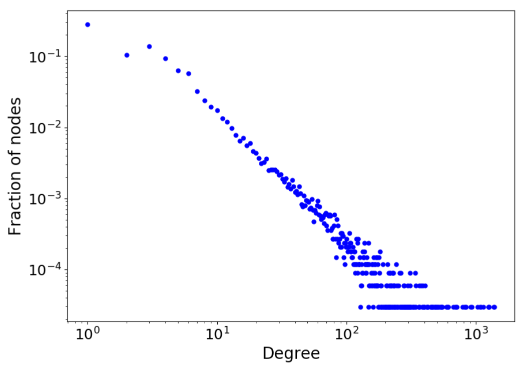
(a) Mountain plot of Enron graph



(b) Degree distribution of Enron graph



(c) Mountain plot of configuration model graph (generated from degree distribution of Enron graph)



(d) Degree distribution of the configuration model graph of Fig 4.7c

Figure 4.7: Mountain plot and degree distribution of Enron graph and a configuration model graph. The two graphs of similar degree distribution have different graph structures that is demonstrated by the different number and shape of mountains in their corresponding mountain plots.

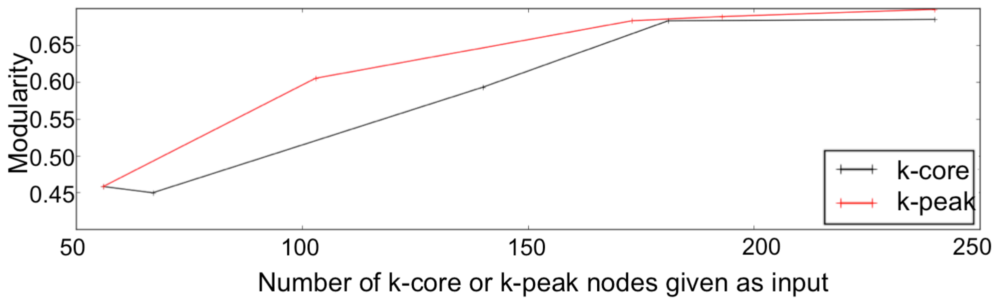
(as compared to that in Fig 4.7a), suggesting more connections to higher contours than to other peripheral nodes.

The mountain plots of the synthetic graphs discussed in this section shows distinct regions in these graphs and how they compare to the real graphs. Admittedly, this is not an extensive study of using the mountain plots to understand how the synthetic graph models might capture properties of the real graphs. This discussion aims to point out that mountain plots give a global view of the distinct regions in a graph and that may be used to further study a given synthetic graph model. An interesting problem that arises from this discussion is whether there exists a generative graph model that can capture the structure of distinct regions in real graph and thus produce mountain plots that are similar to that of real graphs.

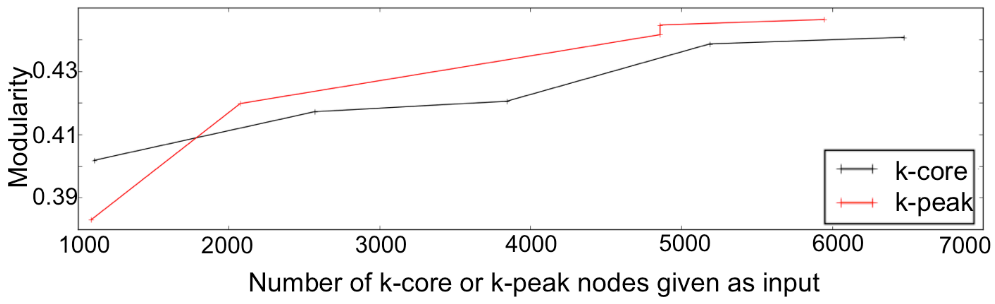
4.7 Applications: k-peak vs. k-core

We now show how the k-peak decomposition can replace or supplement the k-core decomposition in a variety of applications: community detection, contagion, and finding essential proteins in a protein-protein interaction network. Peng, et al. [59] present a three-step method using k-cores to accelerate community detection that finds communities in a k-core with $\sim 30\%$ of the nodes, and uses these community labels to infer community memberships of nodes outside the k-core. The key intuition behind this process is that the community memberships of nodes in the k-core are useful in identifying the community memberships of nodes outside of the core.

4.7.1 Accelerating Community Detection



(a) Facebook Grad network



(b) Facebook Baylor network

Figure 4.8: Community detection using k-cores and k-peaks.

This intuition is not justified in a graph with multiple distinct regions. We propose that instead of using the k-core, one should use the k-peak. To motivate this argument, consider the example Facebook network from Section 4.1: a high k-core captures the Icelandic community structure but not the Eritrean structure, and lower values of k capture too much of the Icelandic region, negating gains in speed.

To illustrate this, we show results for the Facebook Grad and Facebook Baylor networks in Fig. 4.8 (similar results observed in other cases).

We find k-cores and k-peaks containing 10% - 50% of the nodes in the graph, run the algorithm from [59] on these cores and peaks, and measure the modularity of the detected partitionings. When using the k-peak, the algorithm finds a much better partitioning than when using the k-core.

This application captures the essence of our motivation for the k-peak: when the graph contains distinct, independent regions of different densities, the k-core is less suitable than the k-peak, which finds the cores of these different regions.

4.7.2 Locating Essential Proteins

Biological researchers are interested in experimentally identifying 'essential' proteins in protein-protein interaction (PPI) networks. An essential protein is typically defined as one that is required for the survival or reproduction of an organism, as opposed to one that simply increases fitness [78]. Proteins with higher core numbers in this network are more likely to be essential [76]. Here, we show how peak number and core number can be used together.

First, we observe that peak number should not be used as a replacement for core number in this application. Suppose, for instance, that a PPI network has degeneracy d. Consider a node that is tightly connected to this core, but only has core number of $d - 1$. This node is still part of this central, well-connected region, and so is likely to be essential, but has a very low peak number. If we considered peak number alone, we would overlook nodes like this. However, if one considers only core number, essential nodes with low core numbers are excluded. We argue that to better identify these nodes, peak number is useful.

We consider the Protein3 graph listed in Table 4.1, a Yeast PPI network. Over the entire network, essential nodes are 18% of the entire set of nodes, and core numbers range from 1 to 134. Suppose we define a 'low' core number to be anything less than or equal to some value t. Identifying essential low-core-number nodes is necessary to get a complete picture of essential proteins. For each low core number $c \leq t$, let R_c be the set of nodes with highest peak number within the set of all nodes with core number c. Let R represent the union of R_c subsets over all low core numbers c.

	Infection Probability (b)					
p	0.01	0.02	0.05	0.1	0.2	0.5
0.01	0.06 (0.2)	0.04 (0.12)	0.0 (0.03)	0.0 (0.01)	0.0 (0.01)	0.0 (0.0)
0.05	0.12 (0.17)	0.09 (0.14)	0.07 (0.1)	0.04 (0.05)	0.01 (0.02)	0.0 (0.0)
0.1	0.17 (0.28)	0.17 (0.25)	0.11 (0.15)	0.08 (0.09)	0.04 (0.05)	0.01 (0.01)
0.5	0.55 (0.59)	0.53 (0.55)	0.46 (0.47)	0.38 (0.37)	0.27 (0.27)	0.13 (0.14)

Table 4.2: Average fraction improvement (std. deviation) in number of infected nodes when initially infected nodes p are chosen according to peak number vs. core number.

At $t = 30$, R contains 12% of the low-core-number nodes, but 34% of the essential nodes with low core numbers. Similarly, at $t = 10$, R consists of 8% of the nodes with low core number, but 22% of the essential nodes; and if $t = 20$, R consists of 12% of the nodes, but 28% of the essential nodes. In other words, by finding nodes with high peak numbers within a given k-shell, one can improve accuracy.

Effectively, this experiment is identifying nodes that have low core numbers, but are central within their regions: if two nodes have the same core number, but one exists in the periphery of a mountain and the other is in the core of a mountain, the second is much more likely to be essential.

4.7.3 Identifying Influential Nodes

It is well known that nodes with high core number tend to be more important in spreading disease or information through a network. Kitsak, et al. [43] show that in an SIR contagion, the core number of the initially infected node is better correlated with the final number of infected nodes than is the degree of the node.

We show that if multiple nodes are initially infected, selecting nodes with high peak number leads to a larger set of infected nodes than infecting nodes with high core number. We perform simulations as follows: Given a network G, we initially infect p fraction of top core number or top peak number nodes in G, with infection probability b using a SIR contagion. As in [43], we set the lifetime of the infection to be 1 unit of time, and run the simulation until the infection dies out. For every network listed in Table 4.1, for a range of values of p, and b, we perform

100 simulations.

Table 4.2 shows the average fraction improvement obtained by using peak numbers instead of core numbers. For example, a value of 0.5 would indicate that the contagion beginning at nodes with high peak numbers infected 50% more nodes than the contagion beginning at nodes with high core numbers. The standard deviation is sometimes high, but this variation typically occurs on the upper end: that is, using peak number almost always outperforms using core number, but the degree of improvement varies.

In almost all cases, values are positive, indicating that beginning the contagion at nodes with high peak numbers infects a greater number of nodes than beginning the contagion at nodes with high core numbers.

4.8 Conclusion and Extensions

We have studied the graph structures using the k-peak graph decomposition. We analyze the k-peak decomposition algorithm and show that it can be used to better understand the global structure of large network. Additionally, we introduce a new network visualization plot, the mountain plot, which depicts the separate regions of the network. We apply the k-peak decomposition to a variety of real-world networks, and show how it gives substantially deeper insight than the k-core decomposition alone. Finally, we show how the k-peak decomposition can replace the k-core decomposition in several applications, with large gains in performance.

This work naturally leads to some extensions and open problems, related to efficient computation and application of k-peak decomposition. They are as follows:

1. Can we compute k-peaks faster and using lesser space, with theoretical guarantees?

2. The current 2-dimensional mountain plot, can only show the membership of node to the mountain it was affected the most by. Could we build a interactive visualization of the k-peak decomposition that allows a user to pick a k-mountain and study its structure based on all the nodes affected by it?

3. Are there other applications that require identifying the centers of regions that may be varying in densities? k-peak decomposition might help find such regions better than

approaches that simply find the densest regions.

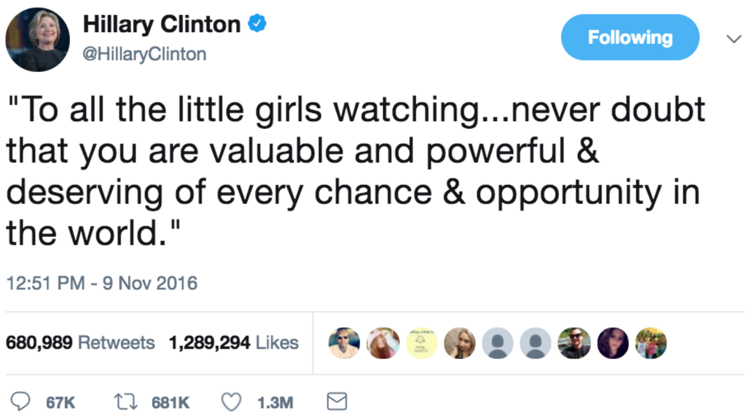
Chapter 5

Individual level study of social networks: User influence

5.1 Introduction

Chapters 5 and 6, constitute the second part of the dissertation, based on individual level study of interpersonal behavior between users in a social network.

As discussed in Section 1.2, social networks such as Facebook, LinkedIn and Twitter, encourage users to share content and to respond to the content posted by other users. For example: Figure 5.1 shows a tweet posted by the then US presidential candidate, Hilary Clinton. It also shows how others have responded to her tweet by ‘retweet’-ing it, ‘favorite’-ing it and replying to her tweet.


Hillary Clinton
@HillaryClinton
Following

"To all the little girls watching...never doubt that you are valuable and powerful & deserving of every chance & opportunity in the world."

12:51 PM - 9 Nov 2016

680,989 Retweets 1,289,294 Likes

67K 681K 1.3M

Figure 5.1: An Example of a tweet posted by Hilary Clinton on Twitter and the responses received in the form of 681K *Retweets*, 1.3M *Favorites* and 67K replies.

As of 2017, more than 3 billion people use some form of a social network [40]. As the number of users increase, we can now observe the interactions between the users and their contacts. This allows us to study the dynamics of the relationships between people, that was not possible in the past. In addition to interacting with people they already know (friends, co-workers, family), people also interact with celebrities and others they may know only through online social networks. As shown in Figure 5.1, Twitter was a medium for users to interact with a politician and for a public personality to reach out to the masses.

In this chapter, we discuss the interesting questions one might ask based on individual level user behavior and give overview of the next chapter.

5.2 Interesting research questions based on user behavior

In general, interpersonal behavior is often studied by the impact people have on others. Some of the key aspects of interpersonal behavior include influence, trust, credibility and leadership [66], [15]. These aspects play significant roles in shaping the community and over time affects the culture and politics of the society. As of September 2017, more than a quarter of the world's population (2.07 billion) use Facebook [35]. The interpersonal behavior that sociologists have studied, can now be observed, recorded and studied more extensively on the online social networks. In addition to the above mentioned aspects of user behavior, there are some issues that are specific to the setting of online interactions. For instance, spam bots may be created and assigned to interact with other users, pretending to be a real person. Or some users may create multiple accounts and give the illusion of a group interacting. Thus there may be additional challenges and questions related to online social networks, that is of interest to the research community. Some of the questions that one could ask include the following:

- Who are the most influential users?
- How can we infer trust and credibility of users?
- What role (leader, follower, go-between) does this person play?
- Which other user is a given user most similar to?
- Is this a real person or a spam bot?

In this work, we focus on measuring influence of users on one another [61]. With more than 3 billion users on social networks [40] and increased activity, massive amounts of data is being generated by these platforms. Thus there is a strong need for space-efficient algorithms to study these datasets. In this work, we address the question – Can we compute influence of users, given the responses of their contacts, in limited space?

We consider different models of input data. For instance when observing the twitter feed for a given user, we might see the total number of retweets or likes a tweet might have got. This is the first input model we consider (Sections 6.4.1, 6.4.2). Alternatively, depending on the source of data, we may have a record for each like or retweet, instead of the total number of likes or retweets, for a given post. This is the second input model we consider. (See Section 6.4.3)

5.3 Overview of the results in this part

In Chapter 7, first, we study h-index as a metric for influence and compare it to other similar metrics in real data. We show that the right choice of a metric of influence is based on the properties of the social platform and how the data is collected. We then present algorithms to compute user influence using h-index in little space. Given the user responses as an input stream, we present a $(1 \pm \epsilon)$ approximation algorithm that uses $O(1/\epsilon \log(1/\epsilon))$ space. On assuming a random permutation of the input stream, we present a $(1+\epsilon)$ approximate algorithm that further reduces the space required to $O(1)$. In the case where each response is input separately, we present an algorithm that uses
$\mathrm{poly}(1/\epsilon, \log(1/\delta), \log n)$ space.

Chapter 6

Streaming Algorithms for Measuring Social Influence

6.1 Introduction

Consider a system that lets users u_i publish items $p(i, j)$ and have other users respond to the items. These could be for example users in Twitter. Users publish their tweets or embed hashtags, and the others may respond by retweeting the publication or posting with the same hashtag handle and so on. Likewise, a user in Facebook may post a story or picture, and their friends can "like" it. In a more familiar setting, researchers publish their results and others cite the publications. In any publication setting, these responses may be positive or negative to varying extents or even just neutral or mix of these responses. Our research is on the positive impact of users, and hence, for the purposes of our research here, we will only focus on the positive responses. [1]

Our focus is on how to measure users' impact. There is a vast theory of studying influence of individual user actions on webpages, posts, etc and also the cumulative impact of all of a users' publications [61, 67, 73]. We adopt a very simple and fundamental notion of impact, well-known in academic world, called the H-index [32]. For each item $p(i, j)$ published by user i, say the *number* of positive responses is $R(i, j)$. Then, the H-index of user i is the largest k such that at least k publications $j_1, j_2, \ldots, j_k$ have k or more (positive) feedback, that is, $R(i, j_\ell) \geq k$, for $1 \leq \ell \leq k$. This is one of the most popular measures of impact in academia, it arose to popularity in the Physics community and is now widespread, with Google Scholar cited frequently by researchers. One of its attractions is that it is parameter-free, for example, it does not focus on the total responses for top 10 cited publications or the total number of

[1] Our discussion can be extended to the setting when the responses are positive to different extent or when responses are negative only or when responses can be a mix of positive and negative, etc. This will lead to notions of impact of users different from the one we study here.

publications with at least 10 citations each etc., where 10 is a parameter. It combines the number of publications and the number of responses (citations) for each in a nifty way, to eke out an impact measure. There is vast literature on extensions of H-index, for example, normalizing for areas, years of publication, different transformations like at least k publications have total of k^2 or more (positive) feedback , or applying it to groups of users, and so on [10, 17, 20]. See [9] for a discussion of the research on H-index.

We veer from prior research on H-index, in not studying its applicability or in extending its definition, but in computing it. If we have memory to store every response $r(k, i, j)$ for each publication $p(i, j)$, then it is easy to compute the H-index of user i. For example, one can calculate $R(i, j)$ by aggregating over k for each (i, j), sorting based on $R(i, j)$'s, and determining H-index by considering them in say the decreasing order. However, in applications, the publications and responses arrive *online* over time as they are composed and released. Further, the number of publications and responses maybe very large, and certainly they are prohibitive if we have a large number of users like the popular publication forums. Motivated by this, we study the *streaming versions* of the problem of computing the H-index.

6.1.1 Our Contribution

Our main results are very small space streaming algorithms for computing the H-index. We consider both aggregated and unaggregated models of streaming. In more detail, our results are as follows.

- *(Aggregate Streaming Model)* We consider a stream of numbers $R(i, j)$ in arbitrary order of i's and j's, and we design an algorithm for computing H-index of each user i. We present a streaming algorithm that uses only $O(\epsilon^{-1} \log \epsilon^{-1})$ space and returns an $1 - \epsilon$ approximation to the H-index of a user. Further if there is a mild lower bound $\Omega(\epsilon^{-3} \log \log n)$ on the ultimate H-index and the publications are considered in a random order, we present an improved algorithm that uses only $O(1)$ storage[2], independent of the number of publications j', number of feedbacks $R(i, j)$'s or the approximation factor ϵ. These are some of the most efficient streaming algorithms one could hope for, not

[2]The constant is just 6 as our proof shows.

even using $O(\log n)$ words that is standard for a lot of streaming analysis over signals of size n.

- *(Cash Register Streaming Model)* In the so called "cash register" streaming model, we get a stream of responses $r(k, i, j)$, that is, the responses arrive unaggregated. We present randomized streaming algorithms for estimating the H-index to ϵ additive error using space $\text{poly}(1/\epsilon, \log(1/\delta), \log n)$ and with success probability at least $1 - \delta$. This is the first sublinear space algorithm for this problem.

We use many of the tools that have been developed in streaming research over the past few years. This includes exponential bucketing methods, distinct or L_0 sampling, group testing with a subset of items to find the heavy hitting ones, etc. We adopt them to calculate H-index, but in the process we significantly improve the results one can get. For example, while using exponential bucketing, we prove that the bucket counters can be reused for our problem and as a result get streaming algorithms that use very little space, sub-logarithmic or constant space even, while typical use of these techniques for other streaming problems use logarithmic space or more. Our work initiates study of streaming algorithms for problems that estimate impact or identify impactful users. There are many variations of H-index and other impact measures for users, and streaming algorithms for calculating them per user, or mining anomalous users on such impact measures is a rich area for further study.

6.2 H-index as a social influence metric

Influence of a user, can be measured based on how other users respond to the activities of a user, over time. A good measure of influence would be one that takes into account the responses a user has received over multiple posts over time and one that is preferably parameter-free. Intuitively, a good influence metric must be one that is higher for a user that consistently gets high responses as compared to a user who got high responses on a few posts. There are many metrics to measure influence that can be roughly classified based on the information considered. Influence can be measured based on the connections between the users, the dissemination of information or more generally based on the responses of other users. In this work, we only consider how other users respond to a user's activities. Specifically, we only consider the counts of a particular kind of response. Arguably, considering other settings such as topic of activity, the connections between users and how information spreads, could give a different perspective of influence of a user. But, here we consider a much more general setting that could be more widely applied and one that is agnostic of the content of a user's activity.

Given the counts of responses for a set of activity of a user, the H-index [32] of the user is defined as the largest value h such that at least h activities or posts made by the user received at least h responses. This metric has been used to measure the impact of a researcher on the scientific community based on the papers she wrote and the citations received for each paper. The key advantages of h-index are that (1) it is parameter-free (2) for h-index to be high, the user has to receive a large number of responses on a large number of posts. Other naive approaches such as average, thresholding, considering top-k posts etc., do not have these advantages. However, there has been a lot of work on different variants of h-index and a discussion on the choice of the right variant in the academic setting (See [4]).

In this section we briefly discuss a few variants of h-index and study their distribution in real datasets.

6.2.1 Variants of h-index

In addition to h-index we consider three variants of it, namely, g-index , h^2-index and h_α-index. One of the drawbacks of h-index is that it ignores the number of responses higher than h. Some users may have a few posts that were very popular and received responses much higher than h. To take into account the highly popular posts along with the other moderately popular posts G-index was proposed by Egghe [19]. G-index is defined as the largest value g such that the top g posts have received a sum of g^2 responses. The author also points out that $g - index \geq h - index$.

Since citations in academic papers is sometimes difficult to track, Kosmulski [45] proposed the $h^{(2)}$-index, to counter this problem. $H^{(2)}$-index is defined as the largest value $h^{(2)}$such there are at least $h^{(2)}$ posts that received at least $[h^{(2)}]^2$ responses each. For example, a user that has 10 posts with 100 *likes* for each post, would get a $h^{(2)}$-index of 10. Clearly $h^{(2)} - index \leq h - index$. In the social setting, $h^{(2)}$, might have another advantage. In social interactions, some activities such as tweets by a celebrity might get consistently high responses. For example,

Dataset	Posts	Response	# Users	# Total Posts
Instagram	Pictures	Likes	4999	236,031
Youtube	Videos	Likes, Views	122,753	239,233
Yelp	Reviews	Upvotes	310,897	1,142,660
DBLP	Papers	Citations	737,133	3,181,744

Table 6.1: Datasets used to study individual social influence

the number of *retweets* received for a tweet by the president of the USA might usually be in hundreds or thousands. In such settings, the h-index of a user might be just the number of their posts, thus making is difficult to distinguish between the influence of different celebrities.

Eck and Waltman [70] proposed a generalization of h-index that can be tuned to be higher or lower than h-index, as required. h_α-index is defined as the largest value h_α such that of the n posts by a user, there are h_α posts with at least $h_\alpha * \alpha$ responses each and the remaining $n - h_\alpha$ posts have at most $h_\alpha * \alpha$ responses each.

6.2.2 Datasets

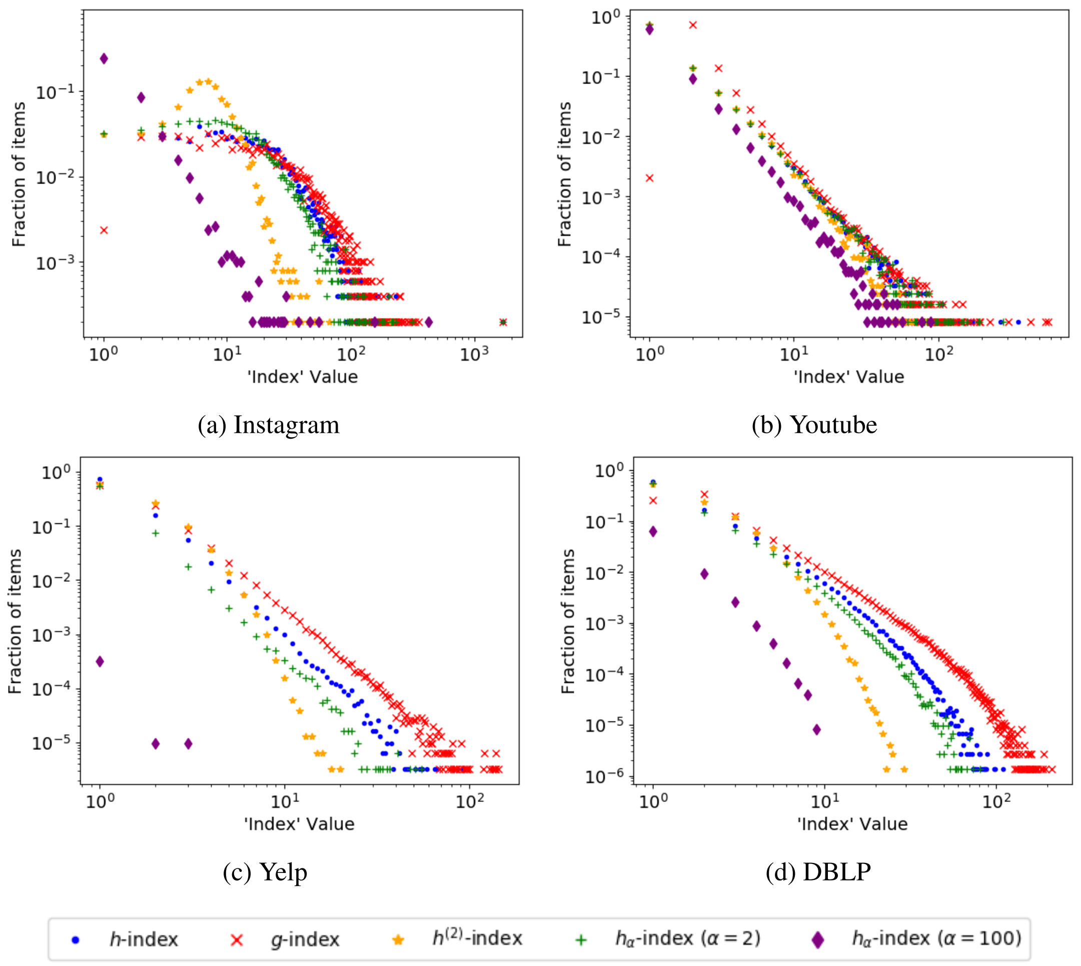
Table 6.1 shows the datasets we used for the analysis here. We consider the Instagram dataset of *'Teens from Profile '* [36], Youtube dataset of views of videos [12], Yelp dataset of *upvotes* on reviews of businesses [3] and DBLP dataset [4] of papers of authors matched with number of citations each paper received scraped from Google Scholar results [18] [51]. Note that the number of responses received in each dataset varies widely. For instance, the number of citations received by a paper is rarely more than a few thousands. On the other hand the number of *views* of videos on Youtube is often in several thousands. Note that in this work, our focus in not on the number of responses received with respect to each post; we focus on the overall influence of a user as a measure of the responses received on all their posts.

6.2.3 Observations in real data

Figure 6.1 shows the distributions of h-index and its above defined three variants, in four real datasets that are listed in Table 6.1. For h_α, we set α to 2 and 100. Note that the figures are

[3]https://www.yelp.com/dataset/challenge

[4]http://dblp.uni-trier.de/



(a) Instagram

(b) Youtube

(c) Yelp

(d) DBLP

Figure 6.1: Distribution of h-index and similar metrics in real data

plotted in log-log scale.

The first observation is that for all the datasets and all four metrics (excluding $\alpha = 100$ for h_α-index), the distributions are skewed and similar to that of a power-law distribution. Since the goal of these metrics is to find influential users, a skewed distribution identifies the small set of users with high indices i.e. high influence. Also note that the value of the indices vary across datasets. The highest h-index in the Instagram dataset is higher than 1000, where as the highest h-index in DBLP is less than 200.

The second observation is that in Figures 6.1c and 6.1d of the Yelp and DBLP datasets respectively, each metric has a distinct distribution. The relationship $h^{(2)} \leq h \leq g$ that was evident in the definitions above, can be seen clearly for the Yelp and DBLP datasets. The reason why the distributions (excluding h_α-index with $\alpha = 100$) are not distinct in Youtube and

Instagram datasets, is due to high number of views/ likes for most videos/ photos in Youtube/ Instagram. For example, consider a user who posts 5 videos that received the following number of views: 30, 35, 50, 80, 100. For this user, h-index, g-index, $h^{(2)}$-index and h_α-index ($\alpha = 2$) are all equal to 5, but h_α-index ($\alpha = 100$) is 1. We observe that for datasets where the number of responses are high for most posts, for most users, a metric like h_α-index, with α set to a high value would help identify influential users. Overall, this demonstrates that a different variant of h-index, and perhaps with different parameters may need to be picked based on the number of responses and the distribution of number of responses over the posts of a user.

In this section, we discussed the different variants of h-index, their properties and their values in real datasets. In real datasets of social interactions, such as twitter or Instagram, users can publish posts and react to other's posts in seconds. Thus the number of user activities and the responses recorded can be very large. For instance, roughly 500 million tweets are posted on Twitter each day [6]. Storing all the data and computing influence metrics offline might not be practical. Such high volume data of social interactions leads to a need for space-efficient computation of influence metrics. In the remainder of this chapter, we will focus on h-index and its computation in little space.

6.3 Preliminaries

In this section, we present the formal definition of H-index and the different data stream models.

6.3.1 *H*-Index

The H-index was suggested in 2005 by Jorge E. Hirsch as a tool for determining theoretical physicists' relative quality and is sometimes called the *Hirsch index* or *Hirsch number*.

Definition 1 (H-Index [32])**.** *Let $V \in \mathbb{N}^n$ be a vector of natural numbers of dimension n. We denote by $h^*(V)$ the maximum number $i \in [n]$ such that the number of elements of V that are greater than or equal to i is at least i, i.e.,*

$$h^*(V) = \arg\max_{i \in [n]} |\{j \in [n] : V[j] \geq i\}| \ ,$$

where we denote $[n] = \{1, 2, \cdots, n\}$. In other words, for the sorted vector V' of V in descending order, h-index of V' (and V) is defined as $h^*(V) = h^*(V') = \max_{i \in [n]}(\min(V'[i]), i)$.

We let $\mathsf{H}(\mathsf{V}) = \{\mathsf{V}[i] : \mathsf{V}[i] \geq h^*(\mathsf{V})\}$ be the *support* of the H-index of the vector V which is defined as the set of numbers $\mathsf{V}[i]$ that are greater than or equal to the H-index $h^*(\mathsf{V})$. We drop V and V' from $h^*(\mathsf{V})$ and $h^*(\mathsf{V}')$ when it is clear from the context.

Example 2. *Let* $V' = [5,5,6,5,5,5,6,5,5,6]$ *be a vector of* 10 *numbers.*
Let $V' = [6,6,6,5,5,5,5,5,5,5]$ *be the sorted variant of* V*. Then, for the* H*-index* $h^*(V)$ *of* V *which is the same as* $h^*(V')$ *we have* $h^*(V) = h^*(V') = 5 = \max_{i\in[10]} \min(V'[i], i))$*, but* $H(V) = H(V') = V'$*.*

6.3.2 Author, Paper and Citation Model

In this work, we consider publication settings that allows users to give positive feedback on other users's activities. Examples of such a setting include tweets and re-tweeting, Facebook posts and likes and academic papers and citations. We define this setting in the general terms of authors, papers and citations.

Let $\mathcal{A}$ be a set of authors and $\mathcal{P}$ be a set of papers. Each paper $p \in \mathcal{P}$ has a *citation number* c_p which is the number of papers in $\mathcal{P}$ that cite the paper p. As is common in science we assume a paper p cannot cite itself.

We denote the set of papers of an $a \in \mathcal{A}$ by $P_a \subseteq \mathcal{P}$. Similarly, we denote the set of authors of a $p \in \mathcal{P}$ by $A_p \subseteq \mathcal{A}$. For the sake of simplicity, we assume that there is a natural number x for the maximum number of authors that a paper p can have, that is, $|A_p| \leq x$. We represent a paper $p \in \mathcal{P}$ by a tuple $(p, a_1^p, \cdots, a_y^p, c_p)$ for $y \leq x$, where $a_1^p, \cdots, a_y^p$ and c_p are the authors and the citation number of the paper p, respectively. Thus, from now on, we consider a paper $p \in \mathcal{P}$ or $(p, a_1^p, \cdots, a_y^p, c_p) \in \mathcal{P}$, interchangeably.

6.3.3 Aggregate and Cash-Register Streams

Let $V \in \mathbb{N}^n$ be a vector of natural numbers of dimension n. Let $h^*(V)$ be the H-index of V. We say $\mathcal{S}$ is an *aggregate* stream of the vector V if $\mathcal{S}$ is a (possibly adversarially ordered) stream of the elements of the underlying vector V. We say $\mathcal{S}$ is a *randomly ordered* aggregate stream if $\mathcal{S}$ is a random order chosen uniformly at random from the set of all permutations (or orders) of the elements of the underlying vector V. We say $\mathcal{S}$ is a *cash register* stream if $\mathcal{S}$ is a stream of updates to the underlying vector V, where an update at time t is a pair (i_t, z) for a natural number z that replaces the entity $V[i_t]$ by $V[i_t] + z$.

For the data model described in Section 6.3.2, the aggregate streaming model would consist of one tuple $(p, a_1^p, \cdots, a_y^p, c_p)$ for each paper p in the dataset, where c_p is the number of citations of the paper. In the cash register model for the above data model, each tuple $(p, a_1^p, \cdots, a_y^p, c_p^i)$ corresponds to the i^{th} update to the number of citations of paper p, such that $c_p = \sum_i c_p^i$. Note that in Section 6.4, for the sake of simplicity we assume the case when there is only one author in the stream. This can easily be extended to papers with multiple authors and computing H-index for each author.

We say a streaming algorithm $\mathcal{A}$ is a multiplicative (ϵ, δ, s)-approximation algorithm for a stream $\mathcal{S}$ (either aggregate or cash register) if over one pass over $\mathcal{S}$, $\mathcal{A}$ uses a space s to report an estimator $h(V)$ such that

$$\mathbf{Pr}\Big[|h^*(V) - h(V)| \leq \epsilon h^*(V)\Big] \geq 1 - \delta \ .$$

Similarly, we say a streaming algorithm $\mathcal{A}$ is an additive (ϵ, δ, s)-approximation algorithm for a stream $\mathcal{S}$ (either aggregate or cash register) if one pass over $\mathcal{S}$, $\mathcal{A}$ uses a space s reports an estimator $h(V)$ such that

$$\mathbf{Pr}\Big[|h^*(V) - h(V)| \leq \epsilon n\Big] \geq 1 - \delta \ .$$

6.3.4 ℓ_0 -Sampler

We define an ℓ_0-Sampler as follows.

Definition 3. (ℓ_0-SAMPLER [24,55]) *Let $0 < \delta < 1$ be a parameter. Let $S = (a_1, t_1), \cdots, (a_i, t_i), \cdots$ be a stream of updates of an underlying vector $x \in \mathbb{R}^n$ where $a_i \in [n]$ and $t_i \in \mathbb{R}$. The i-th update (a_i, t_i) updates the a_i-th element of x using $x[a_i] = x[a_i] + t_i$. An ℓ_0-sampler algorithm for $x \neq 0$ returns FAIL with probability at most δ. Else, with probability $1 - \delta$, it returns an element $j \in [n]$ such that the probability that j-th element is returned is* $\Pr[j] = \frac{|x[i]|^0}{\ell_0(x)}$.

Here, $\ell_0(x) = (\sum_{i \in [n]} |x[i]|^0)$ is the (so-called) "0-norm" of x that counts the number of non-zero entries.

Lemma 4 ([38]). *Let $0 < \delta < 1$ be a parameter. Then, there exists a linear sketch-based algorithm for ℓ_0-sampling using $O(\log^2 n \log \delta^{-1})$ bits of space.*

6.4 *H*-index in streams

In this section we present our algorithms to compute H-index of a single user in different streaming settings, namely aggregate, random order and cash-register models. In Section 6.4.1, we consider the aggregate streaming model, where the input stream is the *number* of likes or retweets of different posts of a user. In Section 6.4.2, we assume that the stream of number of positive feedback is given in a random order. In a cash-register streaming model the input is not the total number of likes or retweets per post, but a sequence of updates when a new like or retweet occurs. In Section 6.4.3, we present our algorithm to compute H-index of a user when the input is a cash-register model.

6.4.1 Aggregate Streaming Model

In this section, we first develop a streaming algorithm that using a space $2\epsilon^{-1} \log n$ words reports an $(1 \pm \epsilon)$-estimator of the H-index of a stream. A similar approach was used by Govindan et. al. [28] to compute H-index of core number of nodes in an effort to compute k-cores in graphs. Later we improve the space of this algorithm down to $6\epsilon^{-1} \log 3\epsilon^{-1}$ words. Formally, we first prove the following theorem. The pseudocode of the algorithm and the correctness proof of the algorithm are given after it.

Theorem 5 (Exponential Histogram). *Let $0 < \epsilon < 1$ be a parameter. Let $V \in \mathbb{N}^n$ be a vector of natural numbers of dimension n whose H-index is $h^*(V)$. Let $\mathcal{S}$ be an arbitrarily (possibly adversarially ordered) stream of the elements of the underlying vector V. Then, there exists a one-pass deterministic streaming algorithm (Algorithm 2) that using space $2\epsilon^{-1} \log n$ words (where each word consists of $\log n$ bits) reports an estimator $h(V)$ such that*

$$(1 - \epsilon)h^*(V) \le h(V) \le h^*(V) \ .$$

Here for the simplicity of the exposition, we assume $n = (1 + \epsilon)^k$ is a power of $(1 + \epsilon)$ and we observe that n is a trivial upper bound for the H-index $h^*(V)$ of an underlying vector

V. We consider $\log_{1+\epsilon} n$ guesses of $(1+\epsilon)^i$ for $h^*(V)$. For each guess $(1+\epsilon)^i$, we design a counter c_i that counts the number of elements of the stream $\mathcal{S}$ that are greater than or equal to $(1+\epsilon)^i$. At the end of the stream, we find the greatest guess $(1+\epsilon)^i$ whose counter is at least $(1+\epsilon)^i$, but the counter of the subsequent guess $(1+\epsilon)^{i+1}$ is less than $(1+\epsilon)^{i+1}$.

Algorithm 2 Exponential Histogram

```
1: Input: Stream $\mathcal{S}$ of elements of $V$.
2: Initialize counters $c_i = 0$ to zero for $i \in [\log_{1+\epsilon} n]$.
3: For step $t$ that the element $\mathcal{S}_t$ arrives
4:     For $i = 1$ to $\log_{1+\epsilon} n$
5:         If $\mathcal{S}_t \geq (1+\epsilon)^i$, then $c_i = c_i + 1$
6: Let $h(V)$=$(1+\epsilon)^i$ be the greatest threshold such that
           $c_i \geq (1+\epsilon)^i$ and $c_{i+1} < (1+\epsilon)^{i+1}$ .
7: Output $h(V)$.
```

Proof. Observe that there exists an index $i \in [\log_{1+\epsilon} n]$ such that $(1+\epsilon)^i \leq h^*(V) < (1+\epsilon)^{i+1}$. Let us consider two counters c_i and c_{i+1}. The counter c_i corresponds to the number of elements of the stream $\mathcal{S}$ that are greater than or equal to $(1+\epsilon)^i$. Similarly, the counter c_{i+1} corresponds to the number of elements of the stream $\mathcal{S}$ that are greater than or equal to $(1+\epsilon)^{i+1}$. Observe that since $(1+\epsilon)^i \leq h^*(V) < (1+\epsilon)^{i+1}$, we have $c_{i+1} < (1+\epsilon)^{i+1}$ and $c_i \geq h^*(V) \geq (1+\epsilon)^i = h(V)$. Therefore,

$$(1-\epsilon)h^*(V) \leq \frac{h^*(V)}{(1+\epsilon)} \leq h(V) = (1+\epsilon)^i \leq h^*(V) \ .$$

The space of the algorithm is $\log_{1+\epsilon} n \leq 2\epsilon^{-1} \log n$ words, where each word consists of $\log n$ bits. ■

Next we improve the space of this algorithm to $6\epsilon^{-1} \log 3\epsilon^{-1}$ words by observing that we do not need to keep the counters for all the guesses at each step of the stream, but only a shifting window of $x = O(1/\epsilon)$ consecutive counters.

Theorem 6 (Shifting Window). *Let $0 < \epsilon < 1$ be a parameter. Let $V \in \mathbb{N}^n$ be a vector of natural numbers of dimension n whose H-index is $h^*(V)$. Let $\mathcal{S}$ be an arbitrarily (possibly adversarially ordered) stream of the elements of the underlying vector V. Then, there exists* *a one-pass deterministic streaming algorithm (Algorithm 3) that using space* $6\epsilon^{-1}\log 3\epsilon^{-1}$ *words (where each word consists of* $\log n$ *bits) reports an estimator* $h(V)$ *such that*

$$(1-\epsilon)h^*(V) \le h(V) \le h^*(V) \ .$$

We maintain a window of $x = O(1/\epsilon)$ consecutive counters $c_i, c_{i+1}, \cdots, c_{i+\log x}$ where the counter c_{i+j} corresponds to the revealed elements of the stream that are greater than or equal to $(1+\epsilon)^{i+j}$. At every step t of the stream if the number of elements that are greater than or equal to $(1+\epsilon)^{i+1}$ becomes at least $(1+\epsilon)^{i+1}$, we shift the window of counters by one, i.e., we delete the counter c_i, create a new counter c_{i+x+1}, and set $i = i+1$. We prove that the number of elements of the stream that are greater than or equal to $(1+\epsilon)^{i+j}$, but not counted by the counter c_{i+j} (mainly because they have been revealed before the counter c_{i+j} is created) is at most $\epsilon(1+\epsilon)^{i+j}$ which is an ϵ-fraction of the H-index of the stream if the H-index of the stream is at least $(1+\epsilon)^{i+j}$.

Algorithm 3 Shifting Window

```
1:  **Input:** Stream $\mathcal{S}$ of elements of $V$.
2:  Let $X = \{0,1,2,\cdots,r = \lceil \log_{1+\epsilon}\epsilon^{-1}\rceil\}$ and $i = 1$.
3:  Initialize counters $c_{i+j} = 0$ for $j \in X$.
4:  For step $t$ that the element $\mathcal{S}_t$ arrives
5:      For each $j \in X$
6:          If $\mathcal{S}_t \ge (1+\epsilon)^{i+j}$, then $c_{i+j} = c_{i+j} + 1$
7:          If $c_{i+1} \ge (1+\epsilon)^{i+1}$, then
8:              Delete the counter $c_i$.
9:              Let $i = i + 1$.
10:             Initialize new counter $c_{i+r} = 0$.
11:         If $\mathcal{S}_t \ge (1+\epsilon)^{i+r}$, then $c_{i+r} = c_{i+r} + 1$.
12: Let $h(V)=(1+\epsilon)^{i+j}$ be the greatest threshold for $j \in X$ such that
        $$c_{i+j} \ge (1+\epsilon)^{i+j} \text{ and } c_{i+j+1} < (1+\epsilon)^{i+j+1} \ .$$
13: Output $h(V)$.
```

Proof. For the sake of simplicity we assume $\epsilon^{-1} = (1+\epsilon)^{\ell}$ is a power of $(1+\epsilon)$. Thus, $r = \log_{1+\epsilon}\epsilon^{-1} = \ell$. Observe that for the H-index $h^*(V)$ of the vector $V \in \mathbb{N}^n$, there exists a natural number k for which we have $(1+\epsilon)^k \le h^*(V) < (1+\epsilon)^{k+1}$. Let us set $i = k+1-\ell$,

$x = (1+\epsilon)^i$ and so, $\frac{x}{\epsilon} = (1+\epsilon)^{i+\ell}$. Observe that

$$\frac{x}{\epsilon} \cdot (1+\epsilon)^{-1} = (1+\epsilon)^{i+\ell-1} \leq h^*(V) < (1+\epsilon)^{i+\ell} = \frac{x}{\epsilon} \ .$$

Let us consider the set $Y = \{0, i - s\ell, \cdots, i-2\ell, i-\ell, i, i+\ell\}$ of the powers of $(1+\epsilon)$ where $s = \lfloor i/\ell \rfloor$. The powers in Y correspond to the guesses $Z = \{1, \epsilon^s x, \cdots, \epsilon^2 x, \epsilon x, x, \frac{x}{\epsilon}\}$ for $h^*(V)$. Indeed, for each number $j \in Y$, there exists a counter c_j that

- is initialized when the counter $c_{j-\ell}$ is deleted. That in turn, occurs when $c_{j-\ell+1} \geq (1+\epsilon)^{j-\ell+1}$. Note that the case $j = 1$ is an exception which starts by seeing the first element of the stream $\mathcal{S}$.
- is deleted when for the subsequent counter c_{j+1}, we have $c_{j+1} \geq (1+\epsilon)^{j+1}$. Observe that at the same time the new counter $c_{j+\ell}$ is created and set to zero.

Observe that since $j \in Y$, we must have $j+\ell, j-\ell \in Y$. We decompose the stream $\mathcal{S}$ into $s+2$ substreams recursively as follows. Corresponding to each number $j \in Y$, we define the substream $\mathcal{S}_j$ which starts when the counter c_j is created and set to zero and ends when the counter c_j is deleted. The substream $\mathcal{S}_j$ corresponds to the sub-vector V_j of the vector V including those entries of the vector V that are revealed in the substream $\mathcal{S}_j$. In the remainder of the proof, we assume that the H-index of the sub-vector V_j is $h^*(V_j) = h^*(\mathcal{S}_j)$.

Claim 7. *Let* $j \in Y$ *be an index in* Y*. Let* c_j *be the counter corresponding to the index* j*. At the time when* c_j *is deleted, there are at least* $(1+\epsilon)^{j+1}$ *and at most* $(1+\epsilon)^{j+2}$ *numbers in the union of the substreams* $\cup_{k\leq j \textit{ and } j\in Y}\mathcal{S}_k$ *that are greater than or equal to* $(1+\epsilon)^{j+1}$*. Moreover,*

$$(1+\epsilon)^{j+1} \leq c_{j+1} \leq (1+\epsilon)^{j+2} \ .$$

Proof. For a fix number $j \in Y$, let us consider three subsequent substreams $\mathcal{S}_{j-\ell}, \mathcal{S}_j, \mathcal{S}_{j+\ell}$ that start when the counters $c_{j-\ell}, c_j, c_{j+\ell}$ are created and end when these counters are deleted, respectively.

Recall that the counters c_j and c_{j+1} (observe that $j+1 \notin Y$) are created when the counters $c_{j-\ell}$ and $c_{j-\ell+1}$ are deleted what in turn, happen when $c_{j-\ell+1} \geq (1+\epsilon)^{j-\ell+1}$ and $c_{j-\ell+2} \geq (1+\epsilon)^{j-\ell+2}$, respectively. The substream $\mathcal{S}_j$ starts when the counter c_j is created and the counter $c_{j-\ell}$ is deleted. Therefore, the counter c_{j+1} may not count all the elements of sub-vector V_j that are greater than or equal to $(1+\epsilon)^{j+1}$. Moreover, the substreams $\mathcal{S}_k$ for $k < j$ and $j \in Y$ were ended before the counter c_{j+1} was created. Thus, the counter c_{j+1} does not count the elements of the sub-vectors V_k for $k \le j$ and $j \in Y$ that are greater than or equal to $(1+\epsilon)^{j+1}$. We will bound the number of such elements in the union of the sub-vectors $\cup_{k \le j \text{ and } j \in Y} V_k$ that are not counted in c_{j+1}.

Let us first consider the elements of the sub-vector V_j that are greater than or equal to $(1+\epsilon)^{j+1}$ but are not counted in c_{j+1}. Let us consider the counter $c_{j-\ell+1}$. This counter is deleted when the counter c_{j+1} is created and is created when the counter $c_{j-2\ell+1}$ is deleted. Therefore the counter $c_{j-\ell+1}$ exists before the substream $\mathcal{S}_j$ starts. That is, the number of elements of the sub-vector V_j that are greater than or equal to $(1+\epsilon)^{j+1}$ but are not counted in the counter c_{j+1} have already been counted by the counter $c_{j-\ell+1}$. Observe that those elements of the substream $\mathcal{S}_j$ that are counted by the counter $c_{j-\ell+1}$ must be greater than or equal to $(1+\epsilon)^{j-\ell+1}$ not necessarily greater than or equal to $(1+\epsilon)^{j+1}$, but that gives an upper bound for those elements that are greater than or equal to $(1+\epsilon)^{j+1}$ which is enough for the analysis. Thus, at most $(1+\epsilon)^{j-\ell+2}$ such elements are not counted by the counter c_{j+1}. Indeed, when $c_{j-\ell+2} \ge (1+\epsilon)^{j-\ell+2}$, the counter $c_{j-\ell+1}$ is deleted and the counter c_{j+1} is created and start counting the elements of the the substream $\mathcal{S}_j$ (interchangeably, the sub-vector V_j) that are greater than or equal to $(1+\epsilon)^{j+1}$.

On the other hand, the counter c_j is deleted when $c_{j+1} \ge (1+\epsilon)^{j+1}$. Therefore, the substream $\mathcal{S}_j$ (interchangeably, the sub-vector V_j) contains at least $(1+\epsilon)^{j+1}$ elements greater than or equal to $(1+\epsilon)^{j+1}$.

Overall, the substream $\mathcal{S}_j$ (and respectively, in the sub-vector V_j) contains at least $(1+\epsilon)^{j+1}$ and at most $(1+\epsilon)^{j+1} + (1+\epsilon)^{j-\ell+2} = (1+\epsilon)^{j+1}(1+\epsilon(1+\epsilon))$ elements that are greater than or equal to $(1+\epsilon)^{j+1}$. Let T_j be the exact number of such elements in the the substream $\mathcal{S}_j$. Then,

$$(1+\epsilon)^{j+1} \le T_j \le (1+\epsilon)^{j+1}(1+\epsilon(1+\epsilon)) \ .$$

Thus,

$$c_{j+1} \le T_j \le c_{j+1} \cdot (1+\epsilon(1+\epsilon)) \ .$$

Similarly, for every substream $\mathcal{S}_{j-x\ell}$ for $1 \le x \le s = \lfloor i/\ell \rfloor$ (observe that $j - x\ell \in Y$) we can prove that the number of elements of $\mathcal{S}_{j-x\ell}$ that are greater than or equal to $(1+\epsilon)^{j-x\ell}$ (what could be greater than or equal to $(1+\epsilon)^{j+1}$) are at most $(1+\epsilon)^{j-x\ell}$. Let $T_{j-x\ell}$ be the exact number of such elements in the the substream $\mathcal{S}_{j-x\ell}$. Thus, for $\epsilon \leq 1/2$ we have

$$(1+\epsilon)^{j+1} \leq \sum_{j-x\ell \text{ and } 1\leq x\leq s} T_{j-\ell} \leq \sum_{j-x\ell \text{ and } 0\leq x\leq s} (1+\epsilon)^{j-x\ell+1} \ .$$

That is,

$$\begin{aligned} c_{j+1} \leq \sum_{j-x\ell \text{ and } 1\leq x\leq s} T_{j-\ell} &\leq c_{j+1}\cdot(1+\sum_{1\leq x\leq s}\epsilon^{x}(1+\epsilon)) \\ &\leq c_{j+1}\cdot(1+\epsilon(1+\epsilon)\sum_{0\leq x\leq s}\epsilon^{x}) \\ &\leq c_{j+1}\cdot(1+2\epsilon(1+\epsilon)) \\ &\leq c_{j+1}\cdot(1+3\epsilon) \ . \end{aligned}$$

By replacing ϵ with $\epsilon/3$ we finish the proof of this claim. ■

Similar to the proof of Claim 7 we can prove the general case.

Claim 8. *Let* $j \in Y$ *be an index in* Y. *Let* c_j *be the counter corresponding to the index* j. *At the time* t_{j+r} *of the stream* $\mathcal{S}$ *when the counter* c_{j+r} *for* $0 \leq r \leq \ell$ *is deleted, there are at least* $(1+\epsilon)^{j+r}$ *and at most* $(1+\epsilon)^{j+r+1}$ *numbers from the beginning of the stream* $\mathcal{S}$ *till that time that are greater than or equal to* $(1+\epsilon)^{j+r}$. *Moreover,* $(1+\epsilon)^{j+r} \leq c_{j+r} \leq (1+\epsilon)^{j+r+1}$.

Having the above claims, we observe that at the time when every counter c_i is deleted, the next counter c_{i+1} is a $(1\pm\epsilon)$-approximation of the H-index of the stream seen so far.

The space of the algorithm is $\log_{1+\epsilon}\epsilon^{-1} \leq 2\epsilon^{-1}\log\epsilon^{-1}$ words. Since in the proofs of Claims 7 and 8 we need to replace ϵ by $\epsilon/3$, the space of the algorithm will be $6\epsilon^{-1}\log 3\epsilon^{-1}$ words where each word consists of $\log n$ bits.

■

6.4.2 Random Order Stream

In this section we show that if a stream $\mathcal{S}$ of items is uniformly randomly ordered, we can develop an $(1\pm\epsilon)$-estimator for the H-index of the stream $\mathcal{S}$ using much smaller space. In particular, if we know a mild lower bound $O(\epsilon^{-3}\log\log(n))$ for the H-index of the stream, we only need six words (where each word consists of $\log n$ bits) to $(1 \pm \epsilon)$-estimate the H-index of the stream. Without having such a mild lower bound, the space of our new streaming algorithm is $O(\epsilon^{-1} \log \epsilon^{-1})$ words where each word consists of $O(\log\log\log n)$ bits. On the other hand, the streaming algorithm in Theorem 6 uses $O(\epsilon^{-1} \log \epsilon^{-1})$ words where each word in that algorithm consists of $\log n$ bits. Formally, we state this main result in theorem below.

Theorem 9 (Random Order Stream)**.** *Let $\mathcal{S}$ be a randomly ordered stream of elements of an underlying vector $V \in \mathbb{N}^n$ whose H-index is $h^*(V)$. Let $0 < \epsilon, \delta < 1$ and $\beta = 150\epsilon^{-2} \log\log n$. Then, there exists a one-pass streaming algorithm that reports an estimator $h(V)$ such that* $\Pr[|h^*(V) - h(V)| \leq \epsilon h^*(V)] \geq 1 - \delta$.

- *Let $\beta/\epsilon \leq h^*(V)$ be a lower bound for $h^*(V)$. Then, the space usage of this algorithm is six words, where a word is a binary vector of length $\log n$ bits.*
- *If $h^*(V)$ is upper bounded by $h^*(V) \leq \beta/\epsilon$, then the space usage of this algorithm is $6\epsilon^{-1} \log 3\epsilon^{-1}$ words each one consists of $\log(\beta/\epsilon)$ bits.*

We first give an overview of the algorithm. We runs two subroutines corresponding two different cases for $h^*(V)$ in parallel. The first case is when we have a lower bound β/ϵ for $h^*(V)$. The second case is when $h^*(V)$ is upper bounded by β/ϵ. For the latter case we simply run the streaming algorithm of Theorem 5 that needs a space of $O(\epsilon^{-1} \log \epsilon^{-1})$ words where each word consists of $\log(\beta/\epsilon)$ bits. Note that in the latter case the H-index is upper bounded by β/ϵ and hence each word requires $\log(\beta/\epsilon)$ bits. For the former case, we consider $\log_{1+\epsilon}(n/\beta)$ guesses for $h^*(V)$ such that the i-th guess corresponds to $n/(1+\epsilon)^i \leq h^*(V) \leq n/(1+\epsilon)^{i-1}$. For the i-th guess, we look at a window W_i of the stream $\mathcal{S}$ of size $\beta(1+\epsilon)^{i-1}(2+\epsilon)$ and count the number of elements of this window that are greater than or equal to $n/(1+\epsilon)^i$. We should mention that every two windows W_{i-1} and W_i corresponding to two consecutive guesses $i-1$ and i overlap. Indeed, let $Z = \{r_0 = 1, r_1, \cdots, r_i, \cdots, r_{\lceil \log_{1+\epsilon}(n/\beta) \rceil}\}$, where $r_i = r_{i-1} + \beta(1+\epsilon)^i$. The window W_i contains all the elements in the interval $(r_{i-1}, r_{i+1}]$ of the stream $\mathcal{S}$. Observe that W_i contains $\beta(1+\epsilon)^{i-1} + \beta(1+\epsilon)^i = \beta(1+\epsilon)^{i-1}(2+\epsilon)$ elements of the stream $\mathcal{S}$. Let $x = \beta(2+\epsilon)/(1+\epsilon)$. We then accept the i-th guess and report $n/(1+\epsilon)^i$ as an $(1 \pm \epsilon)$-approximation of $h^*(V)$ if $(1 \pm \epsilon)x$ elements of this window are greater than or equal to $n/(1+\epsilon)^i$.

We analyze the algorithm in three steps. Suppose that the i-th guess is correct, that is, $n/(1+\epsilon)^i \leq h^*(V) \leq n/(1+\epsilon)^{i-1}$. First, we show that if we ignore the elements in the interval $[1, r_{i-1}]$ of the stream $\mathcal{S}$ then we lose only ϵ-fraction of $h^*(V)$. Second, we observe that in expectation x elements of the window W_i are greater than or equal to $n/(1+\epsilon)^i$. Finally, we use a Chernoff concentration bound for the hypergeometric distribution (i.e., sampling without replacement) to prove that with high probability (i.e., with probability at least $1-\delta$ for $0 < \delta < 1$), $(1 \pm \epsilon)x$ elements of the window W_i are greater than or equal to $n/(1+\epsilon)^i$.

The Pseudocode of the algorithm is given in below.

Algorithm 4 Random Order Stream

```
 1: Input: Stream $\mathcal{S}$ of elements of $V$.
 2: Let $\beta = 150\epsilon^{-2}\log\log n$.
 3: Let $h_1$ be the output of Algorithm 2 with input $(\mathcal{S}, \beta)$.
        (We stop Algorithm 3 if the reported H-index of this
        algorithm is greater than $\beta$.)
 4: Let $h_2$ be the output of Algorithm 5 with input $(\mathcal{S})$.
 5: $h(V) = \max(h_1, h_2)$.
```

Algorithm 5 Sampling Without Replacement

```
 1: Input: Stream $\mathcal{S}$ of elements of $V$.
 2: Let $\beta = 150\epsilon^{-2}\log\log n$, $x = \beta(2+\epsilon)/(1+\epsilon)$, $k = 1$ and $c, c', r = 0$.
 3: For $i = 0, 1, 2, \cdots, \lceil \log_{1+\epsilon}(n/\beta) \rceil$
 4:     Let $r' = r$ and $r = r + \beta(1+\epsilon)^i$.
 5:     While $r' < k \leq r$
 6:         If $\mathcal{S}_k > n/(1+\epsilon)^i$, then $c = c + 1$.
 7:         If $\mathcal{S}_k > n/(1+\epsilon)^{i+1}$, then $c' = c' + 1$.
 8:         Let $k = k + 1$.
 9:     If $(1-\epsilon/3)x \leq c \leq (1+\epsilon)x$, then
10:         Output $n/(1+\epsilon)^i$ and quit.
11:     Else
12:         $c = c'$ and $c' = 0$.
13: Output $c = 0$.
```

Next we prove Theorem 9. First we consider the case that the H-index $h^*(V)$ is upper bounded by β/ϵ. In this case we invoke Algorithm 3 that returns $(1 \pm \epsilon)$-approximation of $h^*(V)$ using $6\epsilon^{-1}\log 3\epsilon^{-1}$ words where each word consists of $\log h^*(V) \leq \log(\beta/\epsilon) = 9\log(\epsilon^{-1}\log\log n)$ bits.

Next we consider the case that the H-index $h^*(V)$ is lower bounded by β/ϵ. We prove the following lemma.

Lemma 10 (H-index is not very small)**.** *Let us assume that* $h^*(V) \geq \beta/\epsilon$. *Then, Algorithm 5 approximates the* H*-index* $h^*(V)$ *within* $(1 \pm \epsilon)$*-approximation.*

We prove this lemma using a series of lemmas that we prove first.

Let $i \in [\log_{1+\epsilon} n]$ be the guess for $h^*(V)$ such that $\frac{n}{(1+\epsilon)^i} \leq h^*(V) \leq \frac{n}{(1+\epsilon)^{i-1}}$. The first lemma shows that if we take a random sample subset of size $t_i = \beta(1+\epsilon)^{i-1}(2+\epsilon)$, the number of elements in this set that are greater than or equal to $\frac{n}{(1+\epsilon)^i}$ is sharply concentrated its expectation.

Lemma 11 (Sampling Concentration Bound)**.** *Let* $i \in [\log_{1+\epsilon} n]$ *be a guess for* $h^*(V)$ *such that* $\frac{n}{(1+\epsilon)^i} \leq h^*(V) \leq \frac{n}{(1+\epsilon)^{i-1}}$. *Let* T_i *be a random sample subset of size* $t_i = \beta(1+\epsilon)^{i-1}(2+\epsilon)$ *from the indices of the vector* V *taken uniformly at random without replacement. Let* $T_i' = \{j \in T_i : V[j] \geq \frac{n}{(1+\epsilon)^i}\}$ *and* $c = |T_i'|$.

- *Then, for* $t_i = \beta(1+\epsilon)^{i-1}(2+\epsilon)$ *we have*

$$\Pr[(1-\epsilon/3)x \leq c \leq (1+\epsilon)x] \geq 1-\delta \ .$$

- ***Noisy Sampling:*** *Suppose that due to a noise (E.g., previously incorrect guesses of* $h^*(V)$*) up to* $\epsilon h^*(V)$ *elements of* V *that are at least* $\frac{n}{(1+\epsilon)^i}$ *have been deleted. Then, for* $t_i = \beta(1+\epsilon)^{i-1}(2+\epsilon)$ *and the same definition for* T_i' *and* c *we have*

$$\Pr[(1-\epsilon/3)x \leq c \leq (1+\epsilon)x] \geq 1-\delta \ .$$

Proof. We first prove the first claim. We define a random variable X_i which corresponds to the number of indices $j \in T_i$ for which $V[j] \geq \frac{n}{(1+\epsilon)^i}$. Observe that X_i has the hypergeometric distribution $X_i \sim H(n, h^*(V), t_i)$ with the probability mass function (pmf)

$$\Pr[X_i = k] = \frac{\binom{h^*(V)}{k}\binom{n-h^*(V)}{t_i-k}}{\binom{n}{t_i}} \ ,$$

and the expectation $\mathbf{E}[X_i] = p_i t_i$ where $p_i = h^*(V)/n$.

Then, for $0 < \epsilon < 1$ and $t_i = \frac{4\ln(2/\delta)}{\epsilon^2 p_i}$ we have the following tail bound for X_i,

$$\Pr[|X_i - \mathbf{E}[X_i]| \geq \epsilon\mathbf{E}[X_i]] \leq 2\exp(-\epsilon^2\mathbf{E}[X_i]/4) = \delta \ .$$

Recall that $\frac{1}{(1+\epsilon)^i} \leq p_i = h^*(V)/n \leq \frac{1}{(1+\epsilon)^{i-1}}$ which means that $\frac{t_i}{(1+\epsilon)^i} \leq \mathbf{E}[X_i] \leq \frac{t_i}{(1+\epsilon)^{i-1}}$. Thus, $x \leq \mathbf{E}[X_i] \leq (1+\epsilon)x$.

From the above tail bound for X_i, with probability at least $1-\delta$, we have $(1-\epsilon)\mathbf{E}[X_i] \leq X_i \leq (1+\epsilon)\mathbf{E}[X_i]$. Now we replace ϵ by $\epsilon/3$ to obtain

$$(1-\epsilon/3)x \leq X_i \leq (1+\epsilon)^2 x \leq (1+\epsilon)x \ .$$

Next, we prove the second claim. Let V' be the new vector after removing up to $\epsilon h^*(V)$ elements that are at least $\frac{n}{(1+\epsilon)^i}$. Then, since $(1-\epsilon)h^*(V) \leq h^*(V') \leq h^*(V)$, we have

$$\frac{n}{(1+\epsilon)^{i+1}} \leq (1-\epsilon) \cdot \frac{n}{(1+\epsilon)^i} \leq h^*(V') \leq \frac{n}{(1+\epsilon)^{i-1}} \ .$$

Similar to the first claim, since $\frac{1}{(1+\epsilon)^{i+1}} \leq p_i = h^*(V)/n \leq \frac{1}{(1+\epsilon)^{i-1}}$, we obtain $x \leq \mathbf{E}[X_i] \leq (1+\epsilon)^2 x$.

Recall that with probability at least $1-\delta$, we have $(1-\epsilon)\mathbf{E}[X_i] \leq X_i \leq (1+\epsilon)\mathbf{E}[X_i]$. Now we replace ϵ by $\epsilon/3$ to obtain

$$(1-\epsilon/3)x \leq X_i \leq (1+\epsilon)^3 x \leq (1+\epsilon)x \ .$$

■

Next we show that if we ignore the first r_{i-1} elements of the randomly ordered stream $\mathcal{S}$, the H-index of the remaining part of the stream is at least $(1-\epsilon)h^*(V)$.

Lemma 12 (First Sample is a Tiny Noise)**.** *Suppose that* $h^*(V) \geq \frac{n}{(1+\epsilon)^i} \geq \frac{(1+\epsilon)^2\beta}{\epsilon^2}$. *Let* V' *be the vector after sampling and removing* r_{i-1} *elements from* V *uniformly at random without replacement. Then, for* H*-index of* V' *we have* $h^*(V') \geq (1-\epsilon)h^*(V)$.

Proof. We first compute an upper bound on r_{i-1}. Recall that $Z = \{r_0 = 1, r_1, \cdots, r_i, \cdots, r_{\lceil \log_{1+\epsilon}(n/\beta) \rceil}\}$, where $r_i = r_{i-1} + \beta(1+\epsilon)^i$. The window W_i contains all the elements in the interval $(r_{i-1}, r_{i+1}]$ of the stream $\mathcal{S}$. Recall that W_i contains $\beta(1+\epsilon)^{i-1} + \beta(1+\epsilon)^i = \beta(1+\epsilon)^{i-1}(2+\epsilon)$ elements of the stream $\mathcal{S}$. Also we have

$x = \beta(2+\epsilon)/(1+\epsilon)$. Thus,

$$\begin{aligned} r_{i-1} &= \beta \sum_{j=0}^{i-1} (1+\epsilon)^j \\ &= \beta(1+\epsilon)^{i-1}(1 + \frac{1}{(1+\epsilon)} + \frac{1}{(1+\epsilon)^2} + \cdots + \frac{1}{(1+\epsilon)^{i-1}}) \\ &\leq \beta(1+\epsilon)^{i-1} \cdot (\frac{1}{1-\frac{1}{(1+\epsilon)}}) = \frac{\beta(1+\epsilon)^i}{\epsilon} \quad , \end{aligned}$$

elements of the stream, by the Taylor series $\frac{1}{1-x} = \sum_{j=0}^{\infty} x^j$ for $x \in (-1,1)$.

Let us define a random variable Y_i which corresponds to the number of indices $j \in [r_{i-1}] = \{1,2,\cdots,r_{i-1}\}$ for which $V[j] \geq \frac{n}{(1+\epsilon)^i}$. Once again, Y_i has the hypergeometric distribution $Y_i \sim H(n, h^*(V), r_{i-1})$. Therefore, similar to the random variable X_i in Lemma 11, we can prove that with probability $1-\delta$, the random variable Y_i is sharply concentrated around its expectation

$$\mathbf{E}[Y_i] = r_{i-1} \cdot h^*(V)/n \leq \frac{\beta(1+\epsilon)^i}{\epsilon} \cdot \frac{h^*(V)}{n} \quad ,$$

that is, $(1-\epsilon)\mathbf{E}[Y_i] \leq Y_i \leq (1+\epsilon)\mathbf{E}[Y_i]$.

Thus, if $\frac{n}{(1+\epsilon)^i} \leq h^*(V) \leq \frac{n}{(1+\epsilon)^{i-1}}$, we then have

$$\begin{aligned} Y_i &\leq (1+\epsilon) \cdot \frac{\beta(1+\epsilon)^i}{\epsilon} \cdot \frac{h^*(V)}{n} \\ &\leq \frac{\beta(1+\epsilon)^{i+1}}{\epsilon}(\frac{n}{n(1+\epsilon)^{i-1}})\frac{(1+\epsilon)^2\beta}{\epsilon} \quad , \end{aligned}$$

Therefore, if we ignore the first r_{i-1} elements of the uniformly randomly ordered stream $\mathcal{S}$, in the rest of the stream $\mathcal{S}$ we have at least $\frac{n}{(1+\epsilon)^i} - \frac{(1+\epsilon)^2 t}{\epsilon}$ elements that are at least $n/(1+\epsilon)^i$.

Recall that $h^*(V) \geq \frac{n}{(1+\epsilon)^i} \geq \frac{(1+\epsilon)^2\beta}{\epsilon^2}$. Let V' be the vector after seeing at most $\frac{t(1+\epsilon)^i}{\epsilon}$ elements of the stream $\mathcal{S}$ that are at least $n/(1+\epsilon)^i$. We then have

$$h^*(V') \geq h^*(V) - \frac{(1+\epsilon)^2\beta}{\epsilon} \geq (1-\epsilon) \cdot \frac{n}{(1+\epsilon)^i} \geq (1-\epsilon)h^*(V) \quad .$$

■

Now we apply the union bound to show that all the guesses that are far from $h^*(V)$ fail and a guess which is close to $h^*(V)$ will be well-approximated.

Lemma 13 (Union Bound)**.** *Let* $\beta = 36\epsilon^{-2}\ln(\frac{2\log_{1+\epsilon} n}{\delta})$*. Then, with probability at least* $1-\delta$ *we have*

- *for each guess* $i \in [\log_{1+\epsilon} n]$ *such that* $h^*(V) < \frac{n}{(1+\epsilon)^{i+5}}$, *we have* $X_i < (1-\epsilon/3)x$*;*

- *for the guess* i *such that* $\frac{n}{(1+\epsilon)^i} \le h^*(V) \le \frac{n}{(1+\epsilon)^{i-1}}$, *we have* $(1-\epsilon/3)x \le X_i \le (1+\epsilon)x$.

Proof. The second part of this lemma was proved in Lemma 11. Next, we prove the first part. Let $i \in [\log_{1+\epsilon} n]$ be a guess for which $h^*(V) < \frac{n}{(1+\epsilon)^{i+5}}$. We define the random variable X_i corresponding to the number of elements in the window W_i (i.e, which is the interval $(r_{i-1}, r_{i+1}]$) of the stream $\mathcal{S}$ that are greater than or equal to $\frac{n}{(1+\epsilon)^i}$. Recall that $p_i = \frac{h^*(V)}{n} < \frac{1}{(1+\epsilon)^{i+5}}$, $t_i = \beta(1+\epsilon)^{i-1}(2+\epsilon)$, and $x = \beta(2+\epsilon)/(1+\epsilon)$. Similar to Lemma 11, the expectation of X_i is upper-bounded by

$$\begin{aligned}
\mathbf{E}[X_i] = p_i t_i &< \frac{t_i}{(1+\epsilon)^{i+5}} \\
&= \frac{\beta(2+\epsilon)}{(1+\epsilon)^6} \le \frac{x}{(1+\epsilon)^5} \le x(1-\epsilon+\epsilon^2)^5 \\
&\le x(1-3\epsilon/4)^5 \le x(1-4\epsilon/3) \ ,
\end{aligned}$$

where $t_i = (1+\epsilon)^{i+1}\beta$, $\epsilon \le 1/4$ and the Taylor series $\frac{1}{1-x} = \sum_{j=0}^{\infty} x^j$ for $x \in (-1,1)$. Since with probability at least $1-\delta$, $(1-\epsilon)\mathbf{E}[X_i] \le X_i \le (1+\epsilon)\mathbf{E}[X_i]$, we then have

$$\begin{aligned}
X_i &\le (1+\epsilon)(1-4\epsilon/3)x \\
&= (1+\epsilon-4\epsilon/3-4\epsilon^2/3)x < (1-\epsilon/3)x \ ,
\end{aligned}$$

We have $\log_{1+\epsilon} n$ guesses $i \in [\log_{1+\epsilon} n]$ for $h^*(V)$ such that for one of them, say $j \in [\log_{1+\epsilon} n]$ we have $\frac{n}{(1+\epsilon)^j} \le h^*(V) \le \frac{n}{(1+\epsilon)^{j-1}}$. We replace δ by $\delta/\log_{1+\epsilon} n$ and apply the union bound to show that with probability at least $1-\delta$ for the guess j, we have $(1-\epsilon/3)x \le X_j \le (1+\epsilon)x$ and for the other guesses $i \in [\log_{1+\epsilon} n]$ such that $h^*(V) < \frac{n}{(1+\epsilon)^{i+5}}$, we have $X_i < (1-\epsilon/3)x$ what finishes the proof of this lemma. As for the sample size t_i we have

$$\begin{aligned}
t_i &= \frac{4\ln(2/\delta)}{\epsilon^2 p_i} \cdot (1+\epsilon) \le \frac{4(1+\epsilon)^{i+1}\ln(2/(\delta/\log_{1+\epsilon} n))}{(\epsilon/4)^2} \\
&= \beta(1+\epsilon)^{i+1} \ ,
\end{aligned}$$

for $\beta = 36\epsilon^{-2}\ln(\frac{2\log_{1+\epsilon} n}{\delta}) \le 150\epsilon^{-2}\log\log n$. ■

Now we have enough tools to prove Lemma 10.

Finishing the Proof of Lemma 10

Observe that we take the elements in the window W_i (i.e, which is the interval $(r_{i-1}, r_{i+1}]$) of the stream $\mathcal{S}$ when the tests for the guesses $j < i$ in Step 8 of Algorithm 5 fail. Since $r_{i-1} \leq \beta(1+\epsilon)^i/\epsilon$, the last guess that we can make is for $i = \log_{1+\epsilon}(\epsilon n/\beta)$ where we test if

$$\frac{n}{(1+\epsilon)^{\log_{1+\epsilon}(\epsilon n/\beta)}} \leq h^*(V) \leq \frac{n}{(1+\epsilon)^{\log_{1+\epsilon}(\epsilon n/\beta)-1}} ,$$

that is, $\beta/\epsilon \leq h^*(V) \leq (1+\epsilon)\beta/\epsilon$ since that guess r_{i-1} is the last guess such that $r_{i-1} \leq n$.

Now we put all the things together. Let $i \in [\log_{1+\epsilon} n]$ be a guess for $h^*(V)$ such that $\frac{n}{(1+\epsilon)^i} \leq h^*(V) \leq \frac{n}{(1+\epsilon)^{i-1}}$. First, we use Lemma 12 for the uniformly randomly ordered stream $\mathcal{S}$ to prove that if we ignore the first set of $r_{i-1} \leq \frac{\beta(1+\epsilon)^i}{\epsilon}$ elements of the stream $\mathcal{S}$, then the H-index of the rest of the stream $\mathcal{S}$ is at least $(1-\epsilon)h^*(V)$. Let $\mathcal{S}'$ be the remaining set of elements of the stream $\mathcal{S}$ after ignoring the first $\frac{\beta(1+\epsilon)^i}{\epsilon}$ elements of $\mathcal{S}$. Second, we use Lemma 11 that shows if we take a random set T_i of size $t_i = \beta(1+\epsilon)^{i-1}(2+\epsilon)$ from $\mathcal{S}'$ and let $T'_i = \{j \in T_i : V[j] \geq \frac{n}{(1+\epsilon)^i}\}$ and $c = |T'_i|$, then, $\Pr[(1-\epsilon/3)x \leq c \leq (1+\epsilon)x] \geq 1-\delta$. Finally, we use Lemma 13 that shows that for the big guesses $j \in [\log_{1+\epsilon} n]$ such that $h^*(V) < \frac{n}{(1+\epsilon)^{j+5}}$, we have $X_j < (1-\epsilon/3)x$. Therefore, such big guesses fail. This finishes the proof of Lemma 10.

6.4.3 Cash Register Streaming Model

In this section we prove our result for the cash register model.

Theorem 14 (Unbiased Sampling)**.** *Let $0 < \epsilon < 1$ be a parameter. Let $V \in \mathbb{N}^n$ be a vector of natural numbers of dimension $n = (1+\epsilon)^k$ whose H-index is $h^*(V)$ where $k \in \mathbb{N}$. Let $\mathcal{S}$ be a stream of updates (i_t, z) of the underlying vector V. Then, there exists a one-pass randomized streaming algorithm (Algorithm 6) that reports an estimator $h(V)$.*

- ***Multiplicative Error:*** *Let $h^*(V) \geq \beta$ be a lower bound for $h^*(V)$. If this algorithm uses x instances of ℓ_0-Sampler, where $x = \frac{n}{\beta} \cdot \frac{3}{\epsilon^2} \cdot \ln(2/\delta)$, then $(1-\epsilon)h^*(V) \leq h(V) \leq h^*(V)$.*

- ***Additive Error:*** *Suppose we do not have a lower bound for $h^*(V)$. If this algorithm* *uses* x *instances of* ℓ_0*-Sampler where* $x = \frac{3}{\epsilon^2} \cdot \ln(2/\delta)$*, then* $h^*(V) - \epsilon \cdot n \leq h(V) \leq h^*(V) + \epsilon \cdot n$.

The success probability of this algorithm is $1 - \delta$.

The outline of the algorithm is as follows. We sample entries of the vector V (almost) uniformly at random. Given a stream $\mathcal{S}$ of updates to the vector V, we keep track of changes to these samples to find an unbiased estimator for the H-index $h^*(V)$ whose expectation is $h^*(V)$. We show that with high probability the estimator is sharply concentrated around its expectation $h^*(V)$.

Algorithm 6 Unbiased Sampling

```
1: Input: Stream $\mathcal{S}$ of updates $(i_t, z)$ to an underlying $n$-dimensional vector $V$ and a lower
   bound $\beta$ for $h^*(V) \geq \beta$.
2: Let $X$ be the output set of $x$ instances of $\ell_0$-Sampler.
   $\{X$ is a uniformly sample set of non-zero indices of $V.\}$
3: Let $y$ be an $(1 \pm \epsilon)$-approximation of the number of non-zero indices of $V$ using algorithm
   of [39].
4: For $i = 0$ to $\lceil \log_{1+\epsilon} n \rceil$
5:     Let $R_i = \{j \in X : V[j] \geq (1+\epsilon)^i\}$.
6:     Let $r_i = |R_i| \cdot y/x$.
7: Let $z$ be the greatest $(1+\epsilon)^i$ s.t. $r_i \geq (1+\epsilon)^i(1-\epsilon)$.
8: Output $z$.
```

Proof. We first prove the multiplicative error bound. Suppose that $(1+\epsilon)^i \leq h^*(V) \leq (1+\epsilon)^{i+1}$. Observe that z is a random variable which depends on random variable $|R_i|$. In the following we prove that $\mathbf{E}[|R_i|] \geq x \cdot \frac{\beta}{n}$ what yields $\mathbf{E}[z] = (1+\epsilon)^i$.

Let $X = \{a_1, \cdots, a_j, \cdots, a_x\}$ be the set of indices that x instances of ℓ_0-Sampler return. Here we assume that none of x instances of ℓ_0-Sampler and the algorithm of [39] fail which can be easily guaranteed by a simple union bound argument. Recall that $R_i = \{j \in X : V[j] \geq (1+\epsilon)^i\}$ and $r_i = |R_i| \cdot y/x$. Let us define an indicator random variable I_j corresponding to the j-th instance of the ℓ_0-Sampler which is one if $V[a_j] \geq h^*(V) \geq (1+\epsilon)^i$ and zero otherwise. Let $I = \sum_{j=1}^{x} I_j$. Recall that ℓ_0-Sampler takes samples with replacement. Observe that $\mathbf{E}[I_j] = \Pr[I_j = 1] \geq \frac{h^*(V)}{y} \geq \frac{(1+\epsilon)^i}{y}$. Thus, $\mathbf{E}[|R_i|] \geq \mathbf{E}[I] \geq x \cdot \frac{h^*(V)}{y} \geq x \cdot \frac{(1+\epsilon)^i}{y} \geq x \cdot \frac{\beta}{n}$ which means that $\mathbf{E}[r_i] \geq (1+\epsilon)^i$.

We use the Chernoff bound which is stated in below.

Lemma 15 (Chernoff Inequality). *[33] Let* $Z_1, \cdots, Z_m$ *denote* m *independent random variables such that* $0 \le Z_i \le 1$ *for* $1 \le i \le m$. *For* $Z = \sum_{i=1}^{m} Z_i$ *we get*

$$\Pr[|Z - \mathbf{E}[Z]| \ge \epsilon \mathbf{E}[Z] \le 2 \exp\left(-\frac{\epsilon^2 \mathbf{E}[Z]}{3}\right).$$

Using the Chernoff bound for $x = 3\epsilon^{-2} \log(2/\delta) \cdot \frac{\beta}{n}$ we have

$$\Pr[|z - \mathbf{E}[z]| \ge \epsilon \mathbf{E}[z]] = \Pr[|r_i - \mathbf{E}[r_i]| \ge \epsilon(1+\epsilon)^i]$$
$$= \Pr[||R_i| - \mathbf{E}[|R_i|]| \ge \epsilon \mathbf{E}[|R_i|]] \le 2\exp(-\frac{\epsilon^2 \mathbf{E}[|R_i|]}{3}) \le \delta \ .$$

Now we proof the second claim of this theorem. To this end, we use the additive Hoeffding inequality which is given below.

Lemma 16 (Additive Hoeffding Inequality). *[33]*
Let $Y_1, \cdots, Y_m$ *denote* m *independent random variables such that* $0 \le Y_i \le M$ *for* $1 \le i \le n$. *For* $Y = \sum_{i=1}^{m} Y_i$ *we get*

$$\Pr[|Y - \mathbf{E}[Y]| \ge t] \le 2 \exp\left(-\frac{mt^2}{3M^2}\right).$$

Using the additive Hoeffding inequality for $x = 3\epsilon^{-2} \log(2/\delta)$ and $M = 1$ we have

$$\Pr[|z - \mathbf{E}[z]| \ge \epsilon n] = \Pr[|r_i - \mathbf{E}[r_i]| \ge \epsilon n]$$
$$= \Pr[||R_i| - \mathbf{E}[|R_i|]| \ge \epsilon n \frac{y}{x}] \le \Pr[||R_i| - \mathbf{E}[|R_i|]| \ge \epsilon n]$$
$$\le 2\exp(-\frac{x\epsilon^2 n^2}{3M^2}) = 2\exp(-\frac{x\epsilon^2 n^2}{3}) \le \delta^n \le \delta \ ,$$

for $x \le y \le n$ and $0 < \delta < 1$. ■

6.5 Extensions and Concluding Remarks

We have initiated the study of streaming algorithms for problems that estimate impact or identify impactful users, and focused on the H-index of users. There are many variations of the H-index, including those that take publication dates, response timestamps, responder's H-index, and so on into account; there are also variations based on different functions of the number of responses with respect to the number of publications like k publications with a total of k^2 responses each; heavy hitters that are users with H-index that are large in the count or in square of the counts (so called L_2 heavy hitters); etc. While our techniques will extend naturally to some of these problems, there is much that remains to be done in obtaining the best bounds for these streaming problems. New platforms for social interactions may bring different constraints and require re-defining the influence metric. For example, posts on social networks such as Snapchat, can be set to be visible to other users only for a limited time. The extent of interaction possible and the reach of the influence is possibly different than in platforms like Twitter. An interesting study would be to find a reasonable metric for influence in such a system and compute it efficiently.

Chapter 7

Conclusion and Open Problems

The culture, politics and economics of the world is shaped by structure of social connections and interpersonal dynamics based on user behavior. Building consumer products and services, require a good understanding of the human social behavior, computational tools to analyze it and efficient techniques to model and predict it.

In this dissertation, we have developed computational tools such as, mathematical definitions of patterns of social behavior, efficient algorithms to identify them and presented exploratory tools to visualize the social structures. In the first part of the dissertation, we focused on the community level graph structure of the interaction between users. Specifically, we developed a space-efficient approximate algorithm to compute k-core graph decomposition in Chapter 3, that requires only linear space in real social graphs. In Chapter 4, we defined k-mountains and k-peaks to find distinct social groups of varying density of connections, analyzed them and presented a visualization technique to explore them. In the second part of the dissertation, we shifted our focus to measuring influence between users at an individual level. In Chapter 6, we presented extremely space efficient approximate algorithms to compute user influence , that requires as little as $O(1)$ space under certain settings. The contribution of our work in this dissertation can be summarized as an effort to push the boundaries of space-efficient algorithms and exploratory tools, to study complex social behavior.

As individuals influence one another, the influence often has a propagating effect. Information and feedback received from one contact may be passed on to others. Given the graph structure of interactions and the influence between users, we can aim to measure and study the spread of influence. For a given user, we can count the number of users reached and record the responses received. In addition to this, given the graph structure of connectivity, we can also measure the reach of the influence based on graph distance. Although there has been considerable research in this area, there are broad open questions regarding the metrics of influence, their computation and analysis in real data. For example, for a specific user, given the graph structure of interactions and the responses received, it would be interesting to study a good metric of influence that takes into account the reach (how far the influence propagated), the coverage (how many users were influenced) and the impact (how strongly they responded) of the influence.

Zooming out of the individual level, we could consider the influence of a community or a group of people. Given the graph structure of interactions and the responses to individual actions, we could ask questions similar to the ones we discussed above, about the influence of a community or a group. It may be possible that certain sets of individuals are more influential as a group than their collective individual influence. An interesting direction of study would be to measure the influence of different groups, the influence of individuals within these groups and dynamics of influence between other individuals and groups.

The study of social structures and interactions has many unanswered questions, interesting patterns and challenging computational problems. This dissertation is a step in solving some of the computational problems and gaining a better understanding of the structures in real social networks.

References

[1] J. Abello and F. Queyroi. Fixed points of graph peeling. In *Advances in Social Networks Analysis and Mining (ASONAM), 2013 IEEE/ACM International Conference on*, pages 256–263. IEEE, 2013.

[2] W. Aiello, F. Chung, and L. Lu. A random graph model for power law graphs. *Experimental Mathematics*, 10(1):53–66, 2001.

[3] R. Albert and A. lszl Barabsi. Statistical mechanics of complex networks. *Rev. Mod. Phys*, page 2002.

[4] S. Alonso, F. Cabrerizo, E. Herrera-Viedma, and F. Herrera. h-index: A review focused in its variants, computation and standardization for different scientific fields. *Journal of Informetrics*, 3(4):273–289, 2009.

[5] S. Aridhi, M. Brugnara, A. Montresor, and Y. Velegrakis. Distributed k-core decomposition and maintenance in large dynamic graphs. In *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, pages 161–168, New York, NY, USA, 2016.

[6] S. Aslam. Twitter by the numbers: Stats, demographics & fun facts. `https://www.omnicoreagency.com/twitter-statistics/`, 2017. Accessed: 2017-12-06.

[7] V. Batagelj and M. Zaversnik. An O(m) algorithm for cores decomposition of networks. *Advances in Data Analysis and Classification*, 5(2):129–145, 2011.

[8] S. P. Borgatti and M. G. Everett. Models of core/periphery structures. *Social Networks*, 21(4):375–395, February 2000.

[9] L. Bornmann. h-index research in scientometrics: A summary. *Journal of Informetrics*, 8(3):749 – 750, 2014.

[10] L. Bornmann, W. Marx, A. Y. Gasparyan, and G. D. Kitas. Diversity, value and limitations of the journal impact factor and alternative metrics. *Rheumatology International*, 32(7):1861–1867, 2012.

[11] B. Bringmann, M. Berlingerio, F. Bonchi, and A. Gionis. Learning and predicting the evolution of social networks. *IEEE Intelligent Systems*, 25(4):26–35, July 2010.

[12] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon. I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System. In *ACM Internet Measurement Conference*, October 2007.

[13] W. Chen, C. Castillo, and L. V. S. Lakshmanan. *Information and Influence Propagation in Social Networks*. Morgan & Claypool, 2013.

[14] J. Cheng, Y. Ke, S. Chu, and M. T. Ozsu. Efficient core decomposition in massive networks. In *ICDE*, pages 51–62, 2011.

[15] R. B. Cialdini, , and N. J. Goldstein. Social influence: Compliance and conformity. *Annual Review of Psychology*, 55(1):591–621, 2004. PMID: 14744228.

[16] J. Cohen. Trusses: Cohesive subgraphs for social network analysis. 2008.

[17] G. Cormode, Q. Ma, S. Muthukrishnan, and B. Thompson. Socializing the h-index. *J. Informetrics*, 7(3):718–721, 2013.

[18] G. Cormode, Q. Ma, S. Muthukrishnan, and B. Thompson. Socializing the h-index. *Journal of Informetrics*, 7(3):718–721, 2013.

[19] L. Egghe. Theory and practise of the *g*-index. *Scientometrics*, 69(1):131–152, 2006.

[20] L. Egghe. Theory and practise of the g-index. *Scientometrics*, 69(1):131–152, 2006.

[21] S. A. Erdem, C. Seshadhri, A. Pinar, and U. V. Catalyurek. Finding the hierarchy of dense subgraphs using nucleus decompositions. In *WWW*, 2015.

[22] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999.

[23] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75 – 174, 2010.

[24] G. Frahling, P. Indyk, and C. Sohler. Sampling in dynamic data streams and applications. In *Proceedings of the 21st Annual Symposium on Computational Geometry (SoCG)*, pages 142–149, 2005.

[25] C. Giatsidis, D. M. Thilikos, and M. Vazirgiannis. Evaluating cooperation in communities with the k-core structure. In *ASONAM*, pages 87–93, 2011.

[26] M. T. Goodrich and P. Pszona. External-memory network analysis algorithms for naturally sparse graphs. In *ESA*, pages 664–676, 2011.

[27] P. Govindan, M. Monemizadeh, and S. Muthukrishnan. Streaming algorithms for measuring h-impact. In *PODS*, pages 337–346, 2017.

[28] P. Govindan, S. Soundarajan, T. Eliassi-Rad, and C. Faloutsos. Nimblecore: A space-efficient external memory algorithm for estimating core numbers. In *ASONAM*, 2016.

[29] P. Govindan, C. Wang, C. Xu, H. Duan, and S. Soundarajan. The k-peak decomposition: Mapping the global structure of graphs. In *WWW*, pages 1441–1450, 2017.

[30] M. A. Hasan and M. J. Zaki. *A Survey of Link Prediction in Social Networks*, pages 243–275. Springer US, Boston, MA, 2011.

[31] J. E. Hirsch. An index to quantify an individual's scientific research output. *PNAS*, 102(46):16569–16572, 2005.

[32] J. E. Hirsch. An index to quantify an individual's scientific research output. *Proceedings of the National Academy of Sciences of the United States of America*, 102(46):16569–16572, 2005.

[33] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[34] P. Holme. Core-periphery organization of complex networks. *Physical Review E*, 72(4):046111, 2005.

[35] F. Inc. Stats. `https://newsroom.fb.com/company-info/`, 2017. Accessed: 2017-12-10.

[36] J. Y. Jang, K. Han, D. Lee, H. Jia, and P. C. Shih. Teens engage more with fewer photos: Temporal and comparative analysis on behaviors in instagram. In *Proceedings of the 27th ACM Conference on Hypertext and Social Media*, pages 71–81, New York, NY, USA, 2016.

[37] J. Jiang, M. Mitzenmacher, and J. Thaler. Parallel peeling algorithms. *ACM Trans. Parallel Comput.*, 3(1):7:1–7:27, Aug. 2016.

[38] H. Jowhari, M. Saglam, and G. Tardos. Tight bounds for L_p samplers, finding duplicates in streams, and related problems. In *Proceedings of the 17th ACM SIGMOD Symposium on Principles of Database Systems (PODS)*, pages 49–58, 2011.

[39] D. M. Kane, J. Nelson, and D. P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2010, June 6-11, 2010, Indianapolis, Indiana, USA*, pages 41–52, 2010.

[40] S. Kemp. Number of social media users passes 3 billion with no signs of slowing. `https://thenextweb.com/contributors/2017/08/07/number-social-media-users-passes-3-billion-no-signs-slowing`. Accessed: 2017-12-06.

[41] W. Khaouid, M. Barsky, V. Srinivasan, and A. Thomo. K-core decomposition of large networks on a single PC. *PVLDB*, 9(1):13–23, 2015.

[42] H. Kim, J. Shin, E. Kim, H. Kim, S. Hwang, J. E. Shim, and I. Lee. Yeastnet v3: a public database of data-specific and integrated functional gene networks for saccharomyces cerevisiae. *Nucleic Acids Research*, 42:731–736, 2014.

[43] M. Kitsak, L. Gallos, S. Havlin, F. Liljeros, L. Muchnik, E. Stanley, and H. Makse. Identification of influential spreaders in complex networks. *Nature Physics*, 6(11):888–893, 2010.

[44] T. G. Kolda, A. Pinar, T. D. Plantenga, and C. Seshadhri. A scalable generative graph model with community structure. *CoRR*, abs/1302.6636, 2013.

[45] M. Kosmulski. A new Hirsch-type index saves time and works equally well as the original h-index. *ISSI Newsletter*, pages 4–6, 2006.

[46] R. Kumar, J. Novak, and A. Tomkins. *Structure and Evolution of Online Social Networks*, pages 337–357. Springer New York, New York, NY, 2010.

[47] J. Kunegis. KONECT: The koblenz network collection. In *Proc. Int. Web Observatory Workshop*, pages 1343–1350, May 2013.

[48] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data*, 1(1), Mar. 2007.

[49] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[50] Y. Liu, N. Shah, and D. Koutra. An empirical comparison of the summarization power of graph clustering methods. *CoRR*, 2015.

[51] Q. Ma, S. Muthukrishnan, B. Thompson, and G. Cormode. Modeling collaboration in academia: A game theoretic approach. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 1177–1182. ACM, 2014.

[52] B. Marwick. Pleistocene exchange networks as evidence for the evolution of language. *Cambridge Archaeological Journal*, 13(1):6781, 2003.

[53] D. W. Matula and L. L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3):417–427, 1983.

[54] A. Mislove, B. Viswanath, K. P. Gummadi, and P. Druschel. You are who you know: Inferring user profiles in online social networks. In *WSDM*, pages 251–260, 2010.

[55] M. Monemizadeh and D. Woodruff. 1-Pass Relative-Error L_p-Sampling with Applications. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1143–1160, 2010.

[56] A. Montresor, F. D. Pellegrini, and D. Miorandi. Distributed k-core decomposition. *IEEE TPDS*, 24(2):288–300, 2013.

[57] M. E. J. Newman. The structure and function of complex networks. *SIAM REVIEW*, 45:167–256, 2003.

[58] M. P. O'Brien and B. D. Sullivan. Locally estimating core numbers. In *ICDM*, pages 460–469, 2014.

[59] C. Peng, T. G. Kolda, and A. Pinar. Accelerating community detection by using k-core subgraphs. *CoRR*, abs/1403.2226, 2014.

[60] Y. Qian, G. Zhang, and K. Zhang. FACADE: A fast and effective approach to the discovery of dense clusters in noisy spatial data. In *SIGMOD*, pages 921–922, 2004.

[61] F. Riquelme. Measuring user influence on twitter: A survey. *CoRR*, abs/1508.07951, 2015.

[62] M. P. Rombach, M. A. Porter, J. H. Fowler, and P. J. Mucha. Core-periphery structure in networks. *SIAM Journal on Applied Mathematics*, pages 167–190, 2014.

[63] A. E. Saríyüce, B. Gedik, G. Jacques-Silva, K.-L. Wu, and U. V. Çatalyürek. Streaming algorithms for k-core decomposition. *PVLDB*, 6(6):433–444, 2013.

[64] A. E. Sariyüce, C. Seshadhri, and A. Pinar. Parallel local algorithms for core, truss, and nucleus decompositions. *CoRR*, abs/1704.00386, 2017.

[65] S. B. Seidman. Network structure and minimum degree. *Social Networks*, 5(3):269–287, 1983.

[66] W. Sherchan, S. Nepal, and C. Paris. A survey of trust in social networks. *ACM Comput. Surv.*, 45(4):47:1–47:33, Aug. 2013.

[67] J. Sun and J. Tang. *A Survey of Models and Algorithms for Social Influence Analysis*, pages 177–214. Springer US, Boston, MA, 2011.

[68] P. Turán. On an extremal problem in graph theory. *Matematikai és Fizikai Lapok*, 48:436–452, 1941.

[69] J. Ugander, B. Karrer, L. Backstrom, and J. M. Kleinberg. Graph cluster randomization: Network exposure to multiple universes. In *KDD*, pages 329–337, 2013.

[70] N. J. van Eck and L. Waltman. Generalizing the h- and g-indices. *Journal of Informetrics*, 2(4):263 – 271, 2008.

[71] N. Wang, S. Parthasarathy, K. Tan, and A. K. H. Tung. CSV: visualizing and mining cohesive subgraphs. In *SIGMOD*, pages 445–458, 2008.

[72] T. Wang, Y. Chen, Z. Zhang, T. Xu, L. Jin, P. Hui, B. Deng, and X. Li. Understanding graph sampling algorithms for social network analysis. In *2011 31st International Conference on Distributed Computing Systems Workshops*, pages 123–128, June 2011.

[73] S. Wasserman and K. Faust. *Social network analysis : methods and applications*. Structural analysis in the social sciences. Cambridge University Press, 1997, Cambridge, New York, Melbourne, 1994. Autres tirages : 1998, 1999, 2007, 2008.

[74] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998.

[75] S. Wuchty and E. Almaas. Peeling the yeast protein network. *PROTEOMICS*, 5(2):444–449, 2005.

[76] S. Wuchty and E. Almaas. Peeling the yeast protein network. *Proteomics*, 5(2):444–449, 2005.

[77] X. Zhang, T. Martin, and M. E. Newman. Identification of core-periphery structure in networks. *Physical Review E*, 91(3):032803, 2015.

[78] X. Zhang, J. Xu, and W. xin Xiao. A new method for the discovery of essential proteins. *PLoS One*, 8(3), 2013.

Acknowledgement of Previous Publications

Portions of this dissertation are based on work previously published or submitted for publication by the author. Chapter 3 is joint work with Sucheta Soundarajan, Tina Eliassi-Rad and Christos Faloutsos [28]. Chapter 4 is joint work with Chenghong Wang, Chumeng Xu, Hongyu Duan and Sucheta Soundarajan, [29]. Chapter 6 is joint work with Morteza Monemizadeh and S. Muthukrishnan [27]