UNIVERSITY OF CALIFORNIA
SANTA CRUZ

COUNTING CLIQUES IN REAL-WORLD GRAPHS

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Shweta Jain

March 2020

The Dissertation of Shweta Jain
is approved:

Prof. Seshadhri Comandur, Chair

Prof. Abhradeep Guha Thakurta

Prof. Yang Liu

Quentin Williams
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Counting Cliques in Real-World Graphs

by

Shweta Jain

Cliques are important structures in network science that have been used in numerous applications including spam detection, graph analysis, graph modeling, community detection among others. Obtaining the counts of k-cliques in graphs with millions of nodes and edges is a challenging problem due to combinatorial explosion. Essentially, as k increases, the number of k-cliques goes up exponentially, and it is not known how one can count them without enumerating. Most existing techniques fail to count k-cliques for $k > 5$. Obtaining global k-clique counts is challenging. Obtaining counts of k-cliques that each edge or vertex is a part of (called local k-clique counts) is even more so.

In this work, we present a set of techniques to efficiently count the number of k-cliques in large graphs that improve upon the state of the art, both in practice and in theory. Our first method is a randomized algorithm called TuránShadow that uses insights from extremal combinatorics to estimate the k-clique counts for $k \leq 10$, while being orders of magnitude faster and more accurate than state-of-the-art methods. We further employ this machinery of clique counting for counting near-cliques – cliques that are missing a few edges. In another application, we show how going beyond edges and incorporating the information of higher-order structures like k-cliques yields graph visualizations that are more human-readable than existing methods.

In a somewhat surprising result, our second method called Pivoter counts all global and local k-cliques, for all k, in a fraction of the time taken by all other

methods, including parallel/approximation methods. In addition, it improves the worst-case running time of clique counting from $O(2^n)$ to $O(3^{n/3})$ and proves that it is indeed possible to count cliques without enumerating them. Crucially, it uses a classic technique called pivoting that drastically reduces the search space for cliques. Using this algorithm, for the first time we were able to obtain the counts of k-cliques of several graphs for which clique counting was infeasible before.

With increasing data come increasing challenges. We highlight certain open problems and future directions to explore to make clique counts even more accessible on large, real-world graphs.

To my family

Acknowledgments

This Ph.D. experience has been one of the most rewarding, challenging, intellectually stimulating and wonderful experiences of my life. Needless to say, this would not have been possible without an army of people supporting me. I would like to thank my advisor, Prof. Seshadhri Comandur, whose time, advice and generous funding I am grateful for. I have been fortunate to have had wonderful mentors who gave me valuable help and advice at important junctures of my Ph.D. I want to thank Prof. David Gleich and Prof. Anthony Wirth for their support, career advice and kind words of encouragement. In the last one year, I have also had the opportunity to collaborate with Prof. Moses Charikar. I am grateful for his help in finding postdoc opportunities, for his advice and for giving me the chance to work with him.

I am grateful for having had the opportunity to work at Sandia National Labs with Dr. Ali Pinar. The work I did at Sandia has become a part of this thesis. My work has also been supported by NSF Awards CCF-1740850, CCF-1813165, and ARO Award W911NF1910294 and I am grateful for this support. I am grateful also to my several fantastic co-authors, including Prof. David Gleich, Prof. Austin Benson, Prof. Dana Ron, Huda Nassar, Talya Eden and Prof. Seshadhri. I have greatly enjoyed working with them and learning from them.

Teachers have played a very influential role in my life and have steadily led me on this path to a Ph.D. They have inspired me and impacted my life in profound ways. I am forever grateful to Siddhartha Sir, who was not only a brilliant mentor who went above and beyond what his job required to bring out the best in his students, but is a role model to all his students and a wonderful human being. I am thankful to Udayan Kanade, for having the courage to follow the path less traveled and for re-instilling the love of science and math in me. I am grateful to

Prof. László Babai for his infectious enthusiasm and love for teaching that made me fall in love with Discrete Math and Algorithms. I am thankful to Prof. Jason Hartline for mentoring me during my internship which gave me a glimpse of the life of a Ph.D. student and fueled my desire to do research. I am thankful to Prof. Sunil Shende, who gave me invaluable advice regarding Ph.D. admissions. I am grateful to my Algorithms teacher, Gerry, at the University of Chicago for going out of her way to convince the Dean to let me pursue Algorithms. I am also thankful to Prof. Lise Getoor for encouraging me to be at UCSC and for supporting me.

Without a sense of community, the journey of Ph.D. can seem like a very long and solitary process. I am grateful for the friendship and camaraderie of my friends at UCSC, esp. Andrew S., Hadley and Params. I am thankful to my "army" on Nashik Katta, for their encouragement and for celebrating my achievements. I would like to thank my Bay Area friends including Piyush, Atreyee, Dani, Bhago, Shardul, Sonia, Mihir, Harshad, Chirayu, David and Andrew W. for the much needed lighter moments in this journey. Thank you to my friend Akash for helping me navigate through some difficult times during my Ph.D. and for obliging me with his ever-listening ear. I am also grateful to my friends from back home including Sumit, Ameya, Aditya, Sumeet and Ketaki for always supporting me. I am thankful to Lorchen for the many insightful conversations. I am thankful to Anupama Tai and my friends from Saptak Music School for the joy of learning and exploring music with them. I am thankful to Ben and Acharya Marasaabs for their love and wisdom.

I am greatly indebted to Saurabh for his unwavering support and faith in me, and for lifting my spirits on countless occassions. You are one of the biggest reasons for my success and I would not be here today without you. Thank you

also to Arun Uncle and Meenal Aunty for their support.

Lastly, I am infinitely grateful to my ever loving family to whom this work is dedicated – Mom, Dad, Deepa, Megha, Siddharth and Aditi. I am especially thankful to Dad for instilling the love of science from an early age in me and for encouraging me to dream. I am grateful to my mother for the many sacrifices she made and for encouraging me to be an independent thinker. I have immense gratitue for my sisters Deepa and Megha, who have been my pillars of strength and my biggest supporters, and for my brother-in-law, Siddharth, who has always been there for me. Last but not the least, I am very thankful to my adorable 3-year-old niece, Aditi, for always cheering on her Sheta Maasi even though she had no clue what she was cheering for.

Chapter 1

Introduction

Graphs are ubiquitous structures. Wherever there are entities and relations between the entities, there exists a graph that represents the relations and the entities. A routing network, for example, is a graph representing connections between routers. A web graph represents which pages within an interconnected domain or across domains link to which other pages. A citation network has nodes that represent papers and edges that represent which paper cites which other paper. A PPI network or protein-protein interaction network represents which protein interacts with which other protein to carry out a certain bodily function in an organism. A graph representing a social network has nodes for people and edges representing which two people know each other on the social network. All these networks have the common property that they consist of a large number of vertices (ranging from 100s to millions) and similar number of edges. Given such big graphs, how can we make sense of them? What can we say about the structure of the graph beyond just the vertices and the edges? How can we compare, characterize, analyze and generate similar graphs?

One of the approaches to mine useful structural information about graphs which has recieved a lot of attention in recent times is to use counts of certain

patterns – (typically small) subgraphs that may occur in the given graph, for eg., cycles, triangles, cliques, etc. It has been observed that, many real world graphs show significantly higher counts of certain patterns than one would expect in a random graph [126, 179, 258]. Indeed, one of the hallmarks of many real-world graphs is the existence of a lot of triangles – set of 3 vertices, all connected to each other, and this is believed to be an artifact of the procedure generating the graph. For example, in a social network, the existence of a large number of triangles can perhaps be explained by the phenomena that a friend of a friend is likely to be a friend.

Counts of patterns can thus reveal important properties about the structure of the graph and the underlying phenomenon that the graph represents. Pattern counting is referred to with a variety of names: subgraph analysis, motif counting, graphlet analysis, etc. But the fundamental task is to count the occurrence of a small pattern graph in a large input graph. Counts of patterns have been used in anomaly detection, social network analysis, bioinformatics among others [50, 102, 126, 127, 179, 200]. (refer to tutorial [217] and references within). Similar to the vertex degree distribution, [178] and [218] defined a graphlet degree distribution and graphlet kernel, respectively as ways to compare graphs. The use of triangles, and clustering coefficients in particular, has a rich history [50, 126, 258, 259]. Triangle counts have been used in spam detection [30] and role detection [50]. [207] uses clustering coefficients (among other properties) to show that Preferential Attachment has poor clustering. [128, 214, 225] use triadic closure properties to model graphs. The past decade saw use of 4 and 5 vertex patterns [35, 249] in graph analysis. In all such applications, it is essential to have fast algorithms for pattern counting.

The number of possible patterns goes up exponentially with the size of the

pattern. Each of the patterns presents different challenges and often requires different techniques to count them. One of the patterns that is of special importance is the k-clique. A *k-clique* is a set of k vertices that are all connected to each other; thus, a triangle is a 3-clique. Cliques are extremely significant in social network analysis (Chap. 11 of [121] and Chap. 2 of [132]). They are the archetypal example of a dense subgraph, and a number of recent results use cliques to find large, dense subregions of a network [183, 208, 244, 247], in topological approaches to network analysis [224] and in community detection [166]. We focus on the specific problem of *clique counting*.

There is a large literature for counting 3-cliques (triangles) and some of these methods have been extended to counting cliques upto size 5 [11, 137, 171, 194]. However, practical algorithms for counting cliques beyond size 5 have proven to be much harder, and the reason for this is combinatorial explosion. Essentially, as k increases, the number of k-cliques blows up. An autonomous system network with ten million edges has more than a *trillion* 10-cliques. Any enumeration procedure is doomed to failure. Under complexity theoretical assumptions, clique counting is believed to be exponential in the size k [56], and we cannot hope to get a good worst-case algorithm.

Similar to global $k-$clique counts, one can define the per-vertex and per-edge k-clique counts (also sometimes called *local counts*) as the number of k-cliques that every vertex and every edge partakes in. In clustering applications, the local counts are used as vertex or edge weights, and are therefore even more useful than global counts [31, 166, 208, 244, 246, 264].

Local counting, for all k, is harder than global counting, especially given the sheer size of the output. Parallel methods would eventually need to store local counts for every subproblem, which would increase the overall memory footprint.

For local counts, sampling would require far too many random variables, each of which need to be sampled many times for convergence. (We give more explanation in §6.1.5.)

This raises the main question:

Do there exist scalable algorithms for getting all global and local clique counts, on real-world graphs with millions of edges?

We explore this question using several different approaches and tools from randomized algorithms, extremal graph theory and classic backtracking algorithms and show that it is indeed possible to count the number of k-cliques in real-world graphs of millions of vertices and edges, on a single commodity machine.

While clique counts are important, the requirement that every edge in the clique be present is excessively rigid. Data is often noisy or incomplete, and it is likely that cliques that are missing even an edge or two are significant. Hence, it is important to also look at counts of patterns that are close to being cliques. We will call these structures near-cliques but they are also known as quasi-cliques [163, 190] and defective cliques [267] and have several applications ranging from clustering to prediction. Recent work has used the fraction of near-cliques to k-cliques to define higher order variants of clustering coefficients [265]. In the bioinformatics literature, near-cliques (or defective cliques, as they are known) have been used to predict missed protein-protein interactions in noisy PPI networks [267] and have been shown to have good predictive performance. An alternative viewpoint of looking at near-cliques views them as dense subgraphs. Mining dense subgraphs is an important problem with many applications in Network Analysis. [21, 57, 107, 153, 209]

Counting cliques is already challenging, and counting near-cliques introduces more challenges. Most importantly, near-cliques do not enjoy the recursive

structural property of cliques - that a subset of a clique is also a clique. This rules out most recursive backtracking algorithms for clique counting. Empirical evidence suggests that the number of near-cliques in real world datasets is order of magnitudes higher than that of cliques, making the task of counting them equally difficult if not more.

1.1 Contributions

In this work, we present several tools for efficiently counting $k-$cliques and near-cliques in large real-world graphs. Our main contributions are described in the chapters ahead, a short summary of which we provide below. The main results of chapters 3 and 6 and 4 were published at WWW 2017, WWW 2018 and WWW 2020, respectively, and those of 5 at WSDM, 2020.

We stress that we make no distributional assumption on the graph. For each algorithm, we have made our code public for others to use.

1.1.1 TuránShadow

We present *TuránShadow* - an algorithm for estimating the number of k-cliques in a given graph, for a given k. It uses the fact that although real-world graphs are usually sparse (which makes random sampling inefficient), they consists of dense pockets. By drilling down on these dense pockets, and sampling for cliques within them, we can estimate the number of cliques faster. The algorithm uses a classic result from Extremal Combinatorics called Turán's Theorem which characterizes "dense" pockets such that we can bound the number of samples that will be required to obtain estimates with provable error guarantees. We run our algorithm on a number of real-world graphs and demonstrate a speedup

of at least 100 times for most cases over existing state-of-the-art algorithms for counting cliques upto $k = 10$. We also show an application of TuránShadow to graph visualization and demonstrate how the fast computation of k-cliques and incorporating their information in the visualization algorithms can yield more human-readable representations. This paper [134] won the Best Paper Award at WWW, 2017.

Code available at: https://bitbucket.org/sjain12/cliquecounting/

1.1.2 PEANUTS

This work uses the ideas from TuránShadow to provide an algorithm Inverse-TS to count near-cliques efficiently. Near-cliques are cliques that are missing an edge or two. Every near-clique contains a smaller clique. By using TuránShadow to find the smaller cliques and using them as clues to mine near-cliques, we give an estimate for the total number of near-cliques in the graph. We also provide a heuristic that computes the count of near-cliques in an online fashion and drastically reduces the amount of space required. On several real-world instances, Inverse-TS performed several orders of madnitude faster and more accurately over other methods. This paper will be published at WWW, 2020.

Code available at: https://bitbucket.org/sjain12/peanuts/

1.1.3 Pivoter

This chapter describes an exact clique counting algorithm that uses a classic technique called pivoting. Interestingly, it demonstrates that it is indeed possible to count all cliques without enumerating them. This technique is several orders of magnitude faster than any other known algorithm for clique counting, whether

randomized or deterministic, sequential or parallel. Pivoter makes clique counting possible for several graphs for which clique counting was infeasible before. Furthermore, it provides the first scalable and efficient algorithm for obtaining local clique counts in graphs with millions of vertices and improves the worst-case running time of clique counting from $O(2^n)$ to $O(3^{n/3}$. This paper [136] won the Best Paper Award at WSDM, 2020.

Code available at: https://bitbucket.org/sjain12/pivoter/

1.1.4 SADDLES

In this work, we demonstrate how sampling can be used to estimate the count of some graph substructure when we don't even have access to the whole graph. In particular, we design sublinear algorithm for estimating the degree distribution of a graph when given only query access to the graph. Our method required an order of magnitude lesser number of samples to achieve better accuracy that other state-of-the-art methods. This paper [92] was published at WWW, 2018.

Code available at: https://bitbucket.org/sjain12/saddles/

Chapter 2

Preliminaries and State of the art

In this chapter, we lay the groundwork and describe several definitions, concepts and procedures related to clique counting. We will use $G = G(V, E)$ to represent the graph, and k to represent the size of the clique (or near-clique, depending on the context) we want to count. We assume that G is stored as an adjacency list. We will use C_k to represent the number of k-cliques in the graphs and C'_k to represent the number of $k-$cliques estimated by the algorithm, unless specified otherwise. We will use u.a.r. as a shorthand for uniformly at random.

- **Degeneracy**

  Definition 2.0.1. *The degeneracy of a graph is the smallest value α such that any induced subgraph of the graph has a vertex with degree at most α.*

  The degeneracy of a graph is a measure of the density of the graph. Most real-world graphs have bounded degeneracy and typically, $\alpha << n$.

- **Degeneracy ordering**

  Definition 2.0.2. *The degeneracy ordering of a graph is defined as an ordering of vertices obtained by taking the lowest degree vertex in the graph*

at that point (ties may be broken by id) and removing it from the graph.

It can be shown that when the vertices of a graph are ordered by degeneracy and the graph is coverted into a DAG using this ordering i.e. edges are directed from lower to higher vertices in the ordering, the resulting outdegree of any vertex is atmost α, the degeneracy.

For many of the algorithms presented here, we will use the degeneracy ordering to convert the graph into a DAG. For $v \in V$, we will let N_v^+ represent the outneighborhood of v.

It is a well known fact in network science that the degeneracy ordering of a graph can be calculated in linear time.

Lemma 2.0.3. *[173] Given a graph $G = (V, E)$, there is a linear time algorithm that constructs an ayclic orientation of G such that all outdegrees are at most α.*

- **Brute force procedure for clique counting**

  This is a clique enumeration procedure proposed by Chiba and Nischizeki [61] that uses degeneracy ordering to split the graph into several subgraphs and recursively counts cliques in the subgraphs. Since the original algorithm was for listing every k-clique, when the subgraph was a clique it still enumerated every subset in the clique. From the point of view of counting, this is wasteful. If the subgraph is a clique of size n' then the number of r cliques in it is simply $\binom{n'}{r}$.

- **Color coding**

  This is a randomized approximation algorithm [20] that speeds up clique-counting by pruning the search space for cliques. Essentially, to count

Algorithm 1: CliquesBruteForce(G, k)

```
if k == 1:
    return |V|
if G is a clique:
    return (|V| choose k)
Let C'_k = 0
Order the vertices of G according to the degeneracy ordering and convert it into a DAG DG.
Let N_v^+ denote the outneighborhood of v in DG.
For v ∈ V:
    C'_k = C'_k + CliquesBruteForce(N_v^+, k − 1)
return C'_k
```

the number of $k-$cliques, every vertex is assigned a color chosen uniformly at random from a palette of k colors. We then count the number of multicolored $k-$cliques, where multicolored means that every vertex of the clique has a different color. The probability that all vertices of a clique will recieve a different color is $k!/k^k$. Thus, if C_k is the number of k-cliques in a graph, then in expectation $C_k * k!/k^k$ would be multicolored. By counting the number of multicolored k-cliques and comparing with the expected value, we can get an estimate for C_k.

Algorithm 2: CliquesColorCoding(G, k)

```
Assign a color to the vertices in V chosen u.a.r. from a palette of k colors
Let Col[i] denote the color of vertex i
return CliquesColorCodingRec(G, k, Col, ∅)*k!/k^k
```

The problem with this approach is that it does not scale. As k increases, the probability that a clique is multicolored goes down drastically and variance goes up. As a result, the estimates obtained are more noisy.

- **Edge sampling** Edge sampling was discussed by Tsourakakis *et al.* in the context of triangle counting [241, 242, 245], though the idea is flexible

Algorithm 3: CliquesColorCodingRec(G, k, Col, K)

```
if k == 0:
    return 0
Order the vertices of G according to the degeneracy ordering and convert it into a DAG DG.
Let N_v^+ denote the outneighborhood of v in DG.
Let C'_k = 0
Let S = {Col[w] | w ∈ K}
For v ∈ N_v^+:
    if Col[v] ∉ S:
        C'_k = C'_k + CliquesColorCodingRec(N_v^+, k − 1, Col, K ∪ {v})
return C'_k
```

and can be used for large patterns [95]. The idea here is to sample each edge independently with some probability p, and then count k-cliques in the down-sampled graph. A k-clique from the original graph survives in the subsampled graph only if all $\binom{k}{2}$ edges of the clique are retained in the subsampled graph, the probability of which is $(1-p)^{\binom{k}{2}}$. Thus, by counting the number of k-cliques in the subsampled graph and scaling up by $1/(1-p)^{\binom{k}{2}}$, we get an unbiased estimate for the number of k-cliques.

Algorithm 4: CliquesEdgeSampling(G, k, p)

```
For every edge e in E:
    Delete e from G with probability p
return CliquesBruteForce(G, k)/(1 − p)^(k choose 2)
```

Similar to color-coding, edge sampling performs poorly as k increases.

- **GRAFT**

  [201]: Rahman *et al.* give a variant of edge sampling with better performance for large pattern counts [201]. This is a sampling method in which a number of edges s are sampled uniformly at random from the graph, and the number of k-cliques that each sampled edge partakes in is calculated.

Algorithm 5: GRAFT(G, k, s)

Let $C'_k = 0$
Order the vertices of G according to the degeneracy ordering and convert it into a DAG DG.
Let N_v^+ denote the outneighborhood of v in DG.
For $i = 1$ to s:
    Sample an edge (u, v) from E u.a.r.
    $C'_k = C'_k +$CliquesBruteForce$(N_v^+ \cap N_u^+, k-2)$
return $C'_k * |E|/s$

The average number of k-cliques of the sampled edges scaled by the number of edges in the graph gives an estimate for the total number of k-cliques in the graph.

- **kClist**

  This is a parallel version of the clique enumeration algorithm of [61], described in [79]. Essentially, to count k-cliques, one only needs to count the number of $k-1$-cliques in the outneighborhood of every vertex in the graph. Thus, we can treat them as separate problems and parallelize the counting of cliques in them. The parallel algorithms given are able to list trillions of cliques in a day on commodity hardware but does not scale for very large graphs.

Chapter 3

TuránShadow

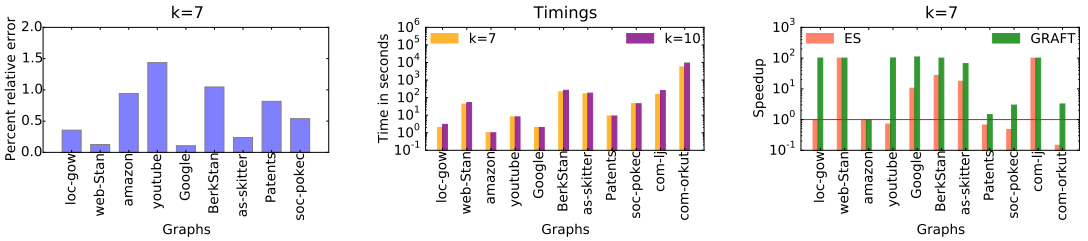
3.1 Introduction

This chapter presents a randomized algorithm called TuránShadow to estimate the counts of k-cliques that works well on real-world graphs for $k \leq 10$. Essentially, it decomposes the given graph into smaller, dense subgraphs and performs random sampling on the dense subgraphs. Crucially, it uses a classic theorem from extremal combinatorics to bound the number of samples required. TuránShadow gives orders of magnitude improvement over pre-existing methods for clique couting. The core results of this paper were presented at WWW, 2017.

3.1.1 Problem Statement

Given an undirected graph $G = (V, E)$, a k-clique is a set S of k vertices in V with all pairs in S connected by an edge. The problem is to count the number of k-cliques, for varying values of k. Our aim is to get all clique counts for $k \leq 10$.

The primary challenge is *combinatorial explosion*. An autonomous system network with ten million edges has more than a *trillion* 10-cliques. Any



(a) Percent relative error for k=7

(b) Timings for k=7 and k=10

(c) Speedup for k=7

Figure 3.1: Summary of behavior of Turán-shadow over several datasets. Fig. 3.1a shows the percent relative error in the estimates for k=7 given by Turán-shadow. We only show results for graphs for which we were able to obtain exact counts using either brute force enumeration, or from the results of [105]. The errors are always $< 2\%$ and mostly $< 1\%$. Fig. 5.1a shows the time taken by Turán-shadow for k=7 and k=10. Fig. 3.1c shows the speedup (time of algorithm/time of Turán-shadow) over other state of the art algorithms for k=7. The red line indicates a speedup of 1. We could not give a figure for speedup for k=10 because for most instances no competing algorithm terminated in min(7 hours, 100 times Turán-shadow time).

enumeration procedure is doomed to failure. Under complexity theoretical assumptions, clique counting is believed to be exponential in the size k [56], and we cannot hope to get a good worst-case algorithm. Our aim is to employ *randomized sampling* methods for clique counting, which have seen some success in counting triangles and small patterns [137, 215, 245]. We stress that we make no distributional assumption on the graph. All probabilities are over the internal randomness of the algorithm itself (which is independent of the instance).

3.1.2 Main contributions

Our main theoretical result is a randomized algorithm Turán-shadow that approximates the k-clique count, for any constant k. We implement this algorithm *on a commodity machine* and get k-clique counts (for all $k \leq 10$) on a variety of data sets, the largest of which has 100M edges. The main features of our work

follow.

Extremal combinatorics meets sampling. Our novelty is in the algorithmic use of classic extremal combinatorics results on clique densities. Seminal results of Turán [248] and Erdős [99] provide bounds on the number of cliques in a sufficiently dense graph. TURÁN-SHADOW tries to cover G by a carefully chosen collection of dense subgraphs that contains all cliques, called a *Turán-shadow*. It then uses standard techniques to design an unbiased estimator for the clique count. Crucially, the result of Erdős [99] (a quantitative version of Turán's theorem) is used to bound the variance of the estimator.

We provide a detailed theoretical analysis of TURÁN-SHADOW, proving correctness and analyzing its time complexity. The running time of our algorithm is bounded by the time to construct the Turán-shadow, which as we shall see, is quite feasible in all the experiments we run.

Extremely fast. In the worst case, we cannot expect the Turán-shadow to be small, as that would imply new theoretical bounds for clique counting. But in practice on a wide variety of real graphs, we observe it to be much smaller than the worst-case bound. Thus, TURÁN-SHADOW can be made into a *practical* algorithm, which also has provable bounds. We implement TURÁN-SHADOW and run it on a commodity machine. Fig. 5.1a shows the time required for TURÁN-SHADOW to obtain estimates for $k = 7$ and $k = 10$ in seconds. The as-skitter graph is processed in less than 3 minutes, despite there being billions of 7-cliques and trillions of 10-cliques. All graphs are processed in minutes, except for an Orkut social network with more than 100M edges (TURÁN-SHADOW handles this graph within 2.5 hours). *To the best of our knowledge, there is no existing work that gets comparable results.* An algorithm of Finocchi *et al.* also computes clique counts, but employs MapReduce on the same datasets [105]. We only require a single

machine to get a good approximation.

We tested Turán-shadow against a number of state of the art algorithmic techniques (color coding [20], edge sampling [245], GRAFT [201]). For 10-clique counting, none of these algorithms terminate for all instances even in 7 hours; Turán-shadow runs in minutes on all but one instance (where it takes less than 2.5 hours). For 7-clique counting, Turán-shadow is typically 10-100 times faster than competing algorithms. (A notable exception is com-orkut, where an edge sampling algorithm runs much faster.)

Excellent accuracy. Turán-shadow has extremely small variance, and computes accurate results (in all instances we could verify). We compute exact results for 7-clique numbers, and compare with the output of Turán-shadow. In Fig. 3.1a, we see that the accuracy is well within 2% (relative error) of the true answer for all datasets. We do detailed experiments to measure variance, and in all cases, Turán-shadow is accurate.

The efficiency and accuracy of Turán-shadow allows us to get clique counts for a variety of graphs, and track how the counts change as k increases. We seem to get two categories of graphs: those where the count increases (exponentially) with k, and those where it decreases with k, see Fig. 5.2b. This provides a new lens to view social networks, and we hope Turán-shadow can become a new tool for pattern analysis.

3.1.3 Related Work

The importance of pattern counts gained attention in bioinformatics with a seminal paper of Milo *et al.* [179], though it has been studied for many decades in the social sciences [126]. Triangle counting and its use has an incredibly rich history, and is used in applications as diverse as spam detection [30], graph

modeling [214], and role detection [50]. Counting four cliques is mostly feasible using some recent developments in sampling and exact algorithms [11, 137].

Clique counts are an important part of recent dense subgraph discovery algorithms [208, 244]. Cliques also play an important role in understanding dynamics of social capital [133], and their importance in the social sciences is well documented [121, 132]. In topological approaches to network analysis, cliques are the fundamental building blocks used to construct simplicial structures [224].

From an algorithmic perspective, clique counting has received much attention from the theoretical computer science community [20, 56, 61, 250]. Maximal clique enumeration has been an important topic [16, 97, 238] since the seminal algorithm of Bron-Kerbosch [47]. Practical algorithms for finding the maximum clique were given by Rossi *et al.* using branch and bound methods [205].

Most relevant to our work is a classic algorithm of Chiba and Nishizeki [61]. This work introduces graph orientations to reduce the search time and provides a theoretical connection to graph arboricity. We also apply this technique in Turán-shadow.

The closest result to our work is a recent MapReduce algorithm of Finocchi *et al.* for clique counting [105]. This result applied the orientation technique of [61], and creates a large set of small (directed) egonets. Clique counting overall reduces to clique counting in each of these egonets, and this can be parallelized using MapReduce. We experiment on the same graphs used in [105] (particularly, the largest ones) and get accurate results on a single, commodity machine (as opposed to using a cluster). Alternate MapReduce methods using multi-way joins have been proposed, though this is theoretical and not tested on real data [7].

A number of randomized techniques have been proposed for pattern counting, and can be used to design algorithms for clique counting. Most prominent are

color coding [20, 37, 127, 271] and edge sampling methods [201, 242, 245]. (MCMC methods [38] typically do not scale for graphs with millions of vertices [137].) We perform detailed comparisons with these methods, and conclude that they do not scale for larger clique counting.

3.2 Main Ideas

The starting point for our result is a seminal theorem of Turán [248]: if the edge density of a graph is more than $1-\frac{1}{k-1}$, then it must contain a k-clique. (The density bound is often called the Turán density for k.) Erdős proved a stronger version [99]. Suppose the graph has t vertices. Then in this case, it contains $\Omega(t^{k-2})$ k-cliques!

Consider the trivial randomized algorithm to estimate k-cliques. Simply sample a uniform random set of k vertices and check if they form a clique. Denote the number of k-cliques by C, then the success probability is $C/\binom{t}{k}$. Thus, we can estimate this probability using $\binom{t}{k}/C$ samples. By Erdős' bound, $C = \Omega(t^{k-2})$. Thus, if a graph (with t vertices) is above the Turán density, one can estimate the number of k-cliques using $O(t^2)$ samples.

Of course, the input graph G is unlikely to have such a high density, and $O(t^2)$ is a large bound. We try to cover all k-cliques in G using a collection of dense subgraphs. This collection is called a *Turán shadow*. We employ orientation techniques from Chiba-Nishizeki to recursively construct a shadow [61].

We take the degeneracy (k-core) ordering in G [213]. It is well-known that outdegrees are typically small in this ordering. To count k-cliques in G, it suffices to count $(k-1)$-cliques in every outneighborhood. (This is the main idea in the MapReduce algorithms of Finocchi et al [105].) If an outneighborhood has density higher than the Turán density for $(k-1)$, we add this set/induced subgraph to

the Turán shadow. If not, we recursively employ this scheme to find denser sets.

When the process terminates, we have a collection of sets (or induced subgraphs) such that each has density above the Turán threshold (for some appropriate k' for each set). Furthermore, the sum of cliques (k'-cliques, for the same k') is the number of k-cliques in G. Now, we can hope to use the randomized procedure to estimate the number of k'-cliques in each set of the Turán shadow. By a theorem of Chiba-Nishizeki [61], we can argue that number of vertices in any set of the Turán shadow is at most $\sqrt{2m}$ (where m is the number of edges in G). Thus, $O(m)$ samples suffices to estimate clique counts for any set in the Turán shadow.

But the Turán shadow has many sets, and it is infeasible to spend $O(m)$ samples for each set. We employ a randomized trick. We only need to approximate the sum of clique counts over the shadow, and can use random sampling for that purpose. Working through the math, we effectively set up a distribution over the sets in the Turán shadow. We pick a set from this distribution, pick some subset of random vertices, and check if they form a clique. The probability of this event can be related to the number of k-cliques in G. Furthermore, we can prove that $O(m)$ samples suffice to estimate this probability. All in all, after constructing the Turán shadow, k-clique counting can be done in $O(m)$ time.

3.2.1 Main theorem and significance

The formal version of the main theorem is Theorem 3.5.5. It requires a fair bit of terminology to state. So we state an informal version that maintains the spirit of our main result. This should provide the reader with a sense of what we can hope to prove. We will define the Turán shadow formally in later sections. But it basically refers to the construct described above.

Theorem 3.2.1. *[Informal] Consider graph $G = (V, E)$ with n vertices, m edges, and maximum core number α. Let $\boldsymbol{S}$ be the Turán k-clique shadow of G, and let $|\boldsymbol{S}|$ be the number of sets in $\boldsymbol{S}$.*

Given any $\delta > 0, \varepsilon > 0, k$, with probability at least $1 - \delta$, the procedure Turán-shadow *outputs a $(1 + \varepsilon)$-multiplicative approximation to the number of k-cliques in G. The running time is linear in $|\boldsymbol{S}|$ and $m\alpha \log(1/\delta)/\varepsilon^2$. The storage is linear in $|\boldsymbol{S}|$.*

Observe that the size of the shadow is critical to the procedure's efficiency. As long as the number of sets in the Turán shadow is small, the extra running time overhead is only linear in m. And in practice, we observe that the Turán shadow scales linearly with graph size, leading to a practically viable algorithm.

Outline: In §6.2, we formally describe Turán's theorem and set some terminology. §3.4 defines (saturated) shadows, and shows how to construct efficient sampling algorithms for clique counting from shadow. §3.5 describes the recursive construction of the Turán shadow. In §3.5.1, we describe the final procedure Turán-shadow, and prove (the formal version of) Theorem 6.3.1. Finally, in §6.5, we detail our empirical study of Turán-shadow and comparison with the state of the art.

3.3 Turán's Theorem

For any arbitrary graph $H = (V(H), E(H))$, let $C_i(H)$ denote the set of cliques in H, and $\rho_i(H) := |C_i(H)|/\binom{|V(H)|}{i}$ is the i-clique density. Note that $\rho_2(H)$ is the standard notion of edge density.

The following theorem of Turán is one of the most important results in extremal graph theory.

Theorem 3.3.1. *(Turán [248]) For any graph H, if $\rho_2(H) > 1 - \frac{1}{k-1}$, then H contains a k-clique.*

This is tight, as evidenced by the complete $(k-1)$-partite graph $T_{n,k-1}$ (also called the Turán graph). In a remarkable generalization, Erdős proved that if an n-vertex graph has even *one more edge* than $T_{n,k-1}$, it must contain many k-cliques. One can think of this theorem as a quantified version of Turán's theorem.

Theorem 3.3.2. *(Erdős [99]) For any graph H over t vertices, if $\rho_2(H) > 1 - \frac{1}{k-1}$, then H contains at least $(t/(k-1))^{k-2}$ k-cliques.*

It will be convenient to express this result in terms on k-clique densities. We introduce some notation: let $f(k) = k^{k-2}/k!$. By Stirling's approximation, $f(k)$ is well approximated by $e^k/\sqrt{2\pi k^5}$. Note that $f(k)$ is some fixed constant, for constant k. This corollary will be critical to our analysis.

Corollary 3.3.3. *For any graph H over t vertices, if $\rho_2(H) > 1 - \frac{1}{k-1}$, then $\rho_k(H) \geq 1/f(k)t^2$.*

Proof. By Theorem 3.3.2, H has at least $(\frac{t}{(k-1)})^{k-2}$ k-cliques. Thus,

$$\rho_k(H) \geq \frac{(\frac{t}{(k-1)})^{k-2}}{\binom{t}{k}} \geq t^{k-2}/t^k \times k!/(k-1)^{k-2} \geq 1/(f(k)t^2)$$

□

3.4 Clique shadows

A key concept in our algorithm is that of *clique shadows*. Consider graph $G = (V, E)$. For any set $S \subseteq V$, we let $C_\ell(S)$ denote the set of ℓ-cliques contained in S.

Definition 3.4.1. *A k-clique shadow $\boldsymbol{S}$ for graph G is a multiset of tuples $\{(S_i, \ell_i)\}$ where $S_i \subseteq V$ and $\ell_i \in \mathbb{N}$ such that: there is a bijection between $C_k(G)$ and $\bigcup_{(S,\ell)\in\boldsymbol{S}} C_\ell(S)$.*

Furthermore, a k-clique shadow $\boldsymbol{S}$ is γ-saturated if $\forall (S,\ell) \in \boldsymbol{S}$, $\rho_\ell(S) \geq \gamma$.

Intuitively, it is a collection of subgraphs, such that the sum of clique counts within them is the total clique count of G. Note that for each set S in the shadow, the associated clique size ℓ is different (for different S). Observe that $\{(V,k)\}$ is trivally a clique shadow. But it is highly unlikely to be saturated.

It is important to define the *size* of $\boldsymbol{S}$, which is really the storage required to represent it.

Definition 3.4.2. *The representation size of $\boldsymbol{S}$ is denoted* size($\boldsymbol{S}$)*, and is* $\sum_{(S,\ell)\in\boldsymbol{S}} |S|$.

Algorithm 6: sample($\boldsymbol{S}, \gamma, k, \varepsilon, \delta$)
$\boldsymbol{S}$ is γ-saturated k-clique shadow
ε, δ are error parameters

For each $(S,\ell) \in \boldsymbol{S}$, set $w(S) = \binom{|S|}{\ell}$;
Set probability distribution $\mathcal{D}$ over $\boldsymbol{S}$ where $p(S) = w(S)/\sum_{(S,\ell)\in\boldsymbol{S}} w(S)$;
For $r \in 1, 2, \ldots, t = \frac{20}{\gamma\varepsilon^2}\log(1/\delta)$;
    Independently sample (S,ℓ) from $\mathcal{D}$;
    Choose a u.a.r. ℓ-tuple A from S;
    If A forms ℓ-clique, set indicator $X_r = 1$. Else, $X_r = 0$;
Output $\frac{\sum_r X_r}{t} \sum_{(S,\ell)\in\boldsymbol{S}} \binom{|S|}{\ell}$ as estimate for $|C_k(G)|$;

When a k-clique shadow $\boldsymbol{S}$ is γ-saturated, each $(S,\ell) \in \boldsymbol{S}$ has many ℓ-cliques. Thus, one can employ random sampling within each S to estimate $|C_\ell(S)|$, and thereby estimate $C_k(G)$. We use a sampling trick to show that we do not need to estimate all $|C_\ell(S)|$; instead we only need $O(1/\gamma)$ samples in total.

Theorem 3.4.3. *Suppose $\boldsymbol{S}$ is a γ-saturated k-clique shadow for G. The procedure* sample($\boldsymbol{S}$) *outputs an estimate $\hat{C}$ such $|\hat{C} - |C_k(G)|| \leq \varepsilon |C_k(G)|$ with probability $> 1 - \delta$.*

The running time of sample($\boldsymbol{S}$) *is $O(\text{size}(\boldsymbol{S}) + \frac{1}{\gamma\varepsilon^2}\log(1/\delta))$.*

Proof. We remind the reader that $w(S) = \binom{|S|}{\ell}$. Set $\alpha = |C_k(G)|/\sum_{S\in\boldsymbol{S}} w(S)$. Observe that

$$\begin{aligned}\Pr[X_r = 1] &= \sum_{(S,\ell)\in\boldsymbol{S}} \Pr[(S,\ell) \text{ is chosen}] \\ &\times \Pr[\ell\text{-clique chosen in } S | (S,\ell) \text{ is chosen}]\end{aligned}$$

The former probability is exactly $w(S)/\sum_{S\in\boldsymbol{S}} w(S)$, and the latter is exactly $|C_\ell(S)|/\binom{|S|}{\ell} = |C_\ell(S)|/w(S)$. So,

$$\Pr[X_r = 1] = \sum_{(S,\ell)\in\boldsymbol{S}} |C_\ell(S)| / \sum_{S\in\boldsymbol{S}} w(S)$$

Since $\boldsymbol{S}$ is a k-clique shadow, $\sum_{(S,\ell)\in\boldsymbol{S}} |C_\ell(S)| = |C_k(G)|$. Thus, $\Pr[X_r = 1] = \alpha$. By the saturation property, $\rho_\ell(S) \geq \gamma$, equivalent to $|C_\ell(S)| \geq \gamma w(S)$. So $\sum_{S\in\boldsymbol{S}} |C_\ell(S)| \geq \gamma \sum_{S\in\boldsymbol{S}} w(S)$. That implies that $\alpha \geq \gamma$. By linearity of expectation, $\mathbf{E}[\sum_{r\leq t} X_r] = \sum_{r\leq t} \mathbf{E}[X_r] \geq \gamma t$.

Note that all the X_rs come from independent trials. (The graph structure plays no role, since the distribution of each X_r does not change upon conditioning on the other X_rs.) By a multiplicative Chernoff bound (Thm 1.1 of [84]),

$$\begin{aligned}&\Pr[\sum_r X_r/t \leq \alpha(1-\varepsilon)] \leq \exp(-\varepsilon^2 \mathbf{E}[\sum_r X_r]/3) \\ \leq\ &\exp(-\varepsilon^2\gamma t/3) = \exp(-5\log(1/\delta)) \leq \delta/5.\end{aligned}$$

By an analogous upper tail bound, $\Pr[\sum_r X_r/t \geq \alpha(1+\varepsilon)] \leq \delta/5$. By the union

bound, with probability at least $1 - 2\delta/5$, $\alpha(1-\varepsilon) \leq \sum_r X_r/t \leq \alpha(1+\varepsilon)$. Note that the output $\hat{C} = (\sum_r X_r/t) \sum_{S \in \boldsymbol{S}} w(S)$. We multiply the bound above on $\sum_r X_r/t$ by $\sum_{S \in \boldsymbol{S}} w(S)$, and note that $\alpha \sum_{S \in \boldsymbol{S}} w(S) = |C_k(G)|$ to complete the proof. □

We stress the significance of Theorem 3.4.3. Once we get a γ-saturated clique shadow $\boldsymbol{S}$, $|C_k(G)|$ can be approximated in time *linear* in size($\boldsymbol{S}$). The number of samples chosen only depends on γ and the approximation parameters, not on the graph size.

But how to actually generate a saturated clique shadow? Saturation appears to be extremely difficult to enforce. This is where the theorem of Erdős (Theorem 3.3.2) saves the day. It merely suffices to make the edge density of each set in the clique shadow high enough. The k-clique density *automatically* becomes large enough.

Theorem 3.4.4. *Consider a k-clique shadow $\boldsymbol{S}$ such that $\forall (S, \ell) \in \boldsymbol{S}$, $\rho_2(S) > 1 - \frac{1}{\ell - 1}$. Let $\gamma = 1/\max_{(S,\ell) \in \boldsymbol{S}} f(\ell)|S|^2$. Then, $\boldsymbol{S}$ is γ-saturated.*

Proof. By Corollary 3.3.3, for every $(S, \ell) \in \boldsymbol{S}$, $\rho_\ell(S) \geq 1/(f(\ell)|S|^2)$. We simply set γ to be the minimum such density over all $(S, \ell) \in \boldsymbol{S}$. □

3.5 Constructing saturated clique shadows

We use a refinement process to construct saturated clique shadows. We start with the trivial shadow $\boldsymbol{S} = \{(V, k)\}$ and iteratively "refine" it until the saturation property is satisfied. By Theorem 3.4.4, we just have to ensure edge densities in each set are sufficiently large.

In this section, we need notation for induced subgraphs. For any set $S \subset V$, $G|_S$ is the subgraph of G induced by S. Given an unsaturated k-clique shadow $\boldsymbol{S}$,

we find some $(S, \ell) \in \boldsymbol{S}$ such that $\rho_2(S) \leq 1 - \frac{1}{\ell - 1}$. By iterating over the vertices, we replace (S, ℓ) by various neighborhoods in $G|_S$ to get a new shadow. We would like the edge densities of these neighborhoods to increase, in the hope of crossing the threshold given in Theorem 3.4.4.

The key insight is to use the *degeneracy ordering* to construct specific neighborhoods of high density that also yield a valid shadow. This is basically the classic graph theoretic technique of computing core decompositions, which is widely used in large-graph analysis [108, 213]. As mentioned earlier, this idea is used for fast clique counting as well [61, 105].

The *degeneracy DAG* of G, denoted $D(G)$ is obtained by orienting edges in degeneracy order. In other words, every edge $(u, v) \in G$ is directed from lower to higher in the degeneracy ordering.

The degeneracy ordering is the deletion time of the standard linear time procedure that computes the degeneracy [173] (Lemma 5.3.1) It is convenient for us to think of the degeneracy in terms of graph orientations. As defined earlier, any permutation on V can be used to make a DAG out of G. We use this idea for generating saturated clique shadows. Essentially, while G may be sparse, *out-neighborhoods* in G are typically dense. (This has been observed in numerous results on dense subgraph discovery [22, 208, 240].)

We now define the procedure Shadow-Finder(G, k), which works by a simple, iterative refinement procedure. Think of $\boldsymbol{T}$ as the current working set, and $\boldsymbol{S}$ as the final output. We take a set (S, ℓ) in $\boldsymbol{T}$, and construct all outneighborhoods in the degeneracy DAG. Any such set whose density is above the Turán threshold goes to $\boldsymbol{S}$ (the output), otherwise, it goes to $\boldsymbol{T}$ (back to the working set).

It is useful to define the *recursion tree* $\mathcal{T}$ of this process as follows. Every pair (S, ℓ) that is ever part of $\boldsymbol{T}$ is a node in $\mathcal{T}$. The children of (S, ℓ) are precisely the

Algorithm 7: Shadow-Finder(G, k)

1 Initialize $\boldsymbol{T} = \{(V, k)\}$ and $\boldsymbol{S} = \emptyset$;
2 While $\exists (S, \ell) \in \boldsymbol{T}$ such that $\rho_2(S) \leq 1 - \frac{1}{\ell - 1}$;
3 Construct the degeneracy DAG $D(G|_S)$;
4 Let N_s^+ denote the outneighborhood (within $D(G|_S)$) of $s \in S$;
5 Delete (S, ℓ) from $\boldsymbol{T}$;
6 For each $s \in S$;
7 If $\ell \leq 2$ or $\rho_2(N_s^+) > 1 - \frac{1}{\ell - 2}$;
8 Add $(N_s^+, \ell - 1)$ to $\boldsymbol{S}$;
9 Else, add $(N_s^+, \ell - 1)$ to $\boldsymbol{T}$;
10 Output $\boldsymbol{S}$;

pairs $(N_s^+, \ell - 1)$ added in Step 8. (At the point, (S, ℓ) is deleted from $\boldsymbol{T}$, and all the $(N_s^+, \ell - 1)$ are added.) Observe that the root of $\boldsymbol{T}$ is (V, k), and the leaves are precisely the final output $\boldsymbol{S}$.

Theorem 3.5.1. *The output $\boldsymbol{S}$ of Shadow-Finder(G, k) is a γ-saturated k-clique shadow, where $\gamma = 1/\max_{(S,\ell)\in\boldsymbol{S}}(f(\ell)|S|^2)$.*

Proof. We first prove by induction the following loop invariant for Shadow-Finder: $\boldsymbol{T} \cup \boldsymbol{S}$ is always a k-clique shadow. For the base case, note that at the beginning, $\boldsymbol{T} = \{(V, k)\}$ and $\boldsymbol{S} = \emptyset$. For the induction step, assume that $\boldsymbol{T} \cup \boldsymbol{S}$ is a k-clique shadow at the beginning of some iteration. The element (S, ℓ) is deleted from $\boldsymbol{T}$. Each $(N_s^+, \ell - 1)$ is added to $\boldsymbol{S}$ or to $\boldsymbol{T}$.

Thus, it suffices to prove that there is a bijection between $C_\ell(S)$ and $\bigcup_{s\in S} C_{\ell-1}(N_s^+)$. (By the induction hypothesis, we can then construct a bijection between $C_k(G)$ and the appropriate cliques in $\boldsymbol{T} \cup \boldsymbol{S}$.) Consider an ℓ-clique K in S. Set s to be the minimum vertex according to the degeneracy ordering in $D(G|_S)$. Observe that the remaining vertices form an $(\ell - 1)$-clique in N_s^+, which we map the K to. This is a bijection, because every clique K can be mapped to a (unique) $(\ell - 1)$-clique, and furthermore, every $(\ell - 1)$-clique in $\bigcup_{s\in S} C_{\ell-1}(N_s^+)$ is in the image of this mapping.

Thus, when Shadow-Finder terminates, $\boldsymbol{T} \cup \boldsymbol{S}$ is a k-clique shadow. Since $\boldsymbol{T}$ must be empty, $\boldsymbol{S}$ is a k-clique shadow. Furthermore, a pair (S, ℓ) is in $\boldsymbol{S}$ iff $\rho_2(S) > 1 - \frac{1}{\ell - 1}$. By Theorem 3.4.4, $\boldsymbol{S}$ is $1/\max_{(S,\ell)\in\boldsymbol{S}}(f(\ell)|S|^2)$-saturated. □

We have a simple, but important claim that bounds the size of any set in the shadow by the degeneracy.

Claim 3.5.2. *Consider non-root $(S, \ell) \in \mathcal{T}$. Then $|S| \leq \alpha(G)$.*

Proof. Suppose the parent of (S, ℓ) is $(P, \ell + 1)$. Observe that S is the outneighborhood of some node p in the DAG $D(G|_P)$. Thus, $|S| \leq \alpha(G|_P)$. The degeneracy can never be larger in a subgraph. (This is apparent by an alternate definition of degeneracy, the maximum smallest degree of an induced subgraph [173].) Hence, $\alpha(G|_P) \leq \alpha(G)$. □

Theorem 3.5.3. *The running time of Shadow-Finder(G, k) is $O(\alpha(G)\text{size}(\boldsymbol{S}) + m + n)$. The total storage is $O(\text{size}(\boldsymbol{S}) + m + n)$.*

Proof. Every time we add $(N_S^+, \ell - 1)$ (Step 8) to $\boldsymbol{T}$, we explicitly construct the graph $G|_{N_s^+}$. Thus, we can guarantee that for every (S, ℓ) present in $\boldsymbol{T}$, we can make queries in the graph $G|_S$. This construction takes $O(|S|^2)$ time, to query every pair in S. (This is *not* required when $S = V$, since $G|_V = G$.) Furthermore, this construction is done for every $(S, \ell) \in \mathcal{T}$, except for the root node in $\mathcal{T}$. Once we have $G|_S$, the degeneracy order can be computed in time linear in the number of edges in $G|_S$ [173].

Thus, the running time can be bounded by $O(\sum_{(S,\ell)\in\mathcal{T}:S\neq V} |S|^2 + m + n)$. By Claim 3.5.2, we can bound $\sum_{(S,\ell)\in\mathcal{T}:S\neq V} |S|^2 = O(\alpha(G) \sum_{(S,\ell)\in\mathcal{T}} |S|)$. We split the sum over leaves and non-leaves. The sum over leaves is precisely a sum over the sets in $\boldsymbol{S}$, so that yields $O(\alpha(G)\text{size}(\boldsymbol{S}))$. It suffices to prove that $\sum_{(S,\ell)\in\mathcal{T}:S \text{ non-leaf}} |S| = O(\text{size}(\boldsymbol{S}))$, which we show next.

Observe that a non-leaf node (S, ℓ) in $\mathcal{T}$ has exactly $|S|$ children, one for each vertex $s \in S$. Thus,

$$\begin{aligned} \sum_{(S,\ell)\in\mathcal{T}:(S,\ell)\text{non-leaf}} |S| &= \sum_{(S,\ell)\in\mathcal{T}} \#\text{ children of } (S,\ell) \\ &= \#\text{ edges in } \mathcal{T} \end{aligned}$$

All internal nodes in $\mathcal{T}$ have at least 2 children, so the number of edges in $\mathcal{T}$ is at most twice the number of leaves in $\mathcal{T}$. But this is exactly the number of sets in the output $\mathcal{S}$, which is at most size($\boldsymbol{S}$).

The total storage is $O(\sum_{(S,\ell)\in\mathcal{T}} |S| + m + n)$, which is $O(\text{size}(S) + m + n)$ by the above arguments. □

We now formally define the Turán shadow to be output of this procedure.

Definition 3.5.4. *The k-clique Turán shadow of G is the output of* Shadow-Finder(G, k).

3.5.1 Putting it all together

Algorithm 8: TURÁN-SHADOW$(G, k, \varepsilon, \delta)$

1 Compute $\boldsymbol{S} = \text{Shadow-Finder}(G, k)$;
2 Set $\gamma = 1/\max_{(S,\ell)\in\boldsymbol{S}}(f(\ell)|S|^2)$;
3 Output $\hat{C}_k = \text{sample}(G, k, \gamma, \varepsilon, \delta)$;

Theorem 3.5.5. *Consider graph $G = (V, E)$ with m edges, n vertices, and degeneracy $\alpha(G)$. Assume $m \leq n^2/4$. Let $\boldsymbol{S}$ be the Turán k-clique shadow of G.*

With probability at least $1 - \delta$ (this probability is over the randomness of TURÁN-SHADOW*; there is no stochastic assumption on G), $|\hat{C}_k - |C_k(G)|| \leq \varepsilon|C_k(G)|$.*

The running time of Turán-shadow *is* $O(\alpha(G)\text{size}(\boldsymbol{S}) + f(k)m\log(1/\delta)/\varepsilon^2 + n)$ *and the total storage is* $O(\text{size}(\boldsymbol{S}) + m + n)$.

Proof. By Theorem 3.5.1, $\boldsymbol{S}$ is γ-saturated, for $\gamma = 1/\max_{(S,\ell)\in\boldsymbol{S}} f(\ell)|S|^2$. Since $m \leq n^2/4$, the procedure Shadow-Finder(G,k) cannot just output $\{(V,k)\}$. All leaves in the recursion tree must have depth at least 2, and by Claim 3.5.2, for all $(S,\ell) \in \boldsymbol{S}$, $|S| \leq \alpha(G)$. A classic bound on the degeneracy asserts that $\alpha(G) \leq \sqrt{2m}$ (Lemma 1 of [61]). Since $f(\ell)$ is increasing in ℓ, $\max_{(S,\ell)\in\boldsymbol{S}} f(\ell)|S|^2 \leq 2f(k)m$. Thus, $\gamma = \Omega(1/(f(k)m))$.

By Theorem 3.4.3, the running time of sample is $O(\text{size}(\boldsymbol{S}) + \log(1/\delta)/(\gamma\varepsilon^2))$, which is $O(\text{size}(\boldsymbol{S}) + f(k)m\log(1/\delta)/\varepsilon^2)$. Theorem 3.4.3 also asserts the accuracy of the output. Adding the bounds of Theorem 3.5.3, we prove the running time and storage bounds. □

3.5.2 The shadow size

The practicality of Turán-shadow hinges on $\text{size}(\boldsymbol{S})$ being small. It is not hard to prove a worst-case bound, using the degeneracy.

Claim 3.5.6. $\text{size}(\boldsymbol{S}) = O(n\alpha(G)^{k-2})$.

Proof. By arguments in the proof of Theorem 3.5.3, we can show that $\text{size}(\boldsymbol{S})$ is at most the number of edges in $\mathcal{T}$. In $\mathcal{T}$, the degree of the root is n, and by Claim 3.5.2, the degree of all other nodes is at most $\alpha(G)$. The depth of the tree is at most $k-1$, since the value of ℓ decreases every step down the tree. That proves that $n\alpha^{k-2}$ bound. □

This bound is not that interesting, and the Chiba-Nishizeki algorithm for exact clique enumeration matches this bound [61]. Indeed, we can design instances

where Claim 3.5.6 is tight (a set of n/α Erdős-Rényi graphs $G_{\alpha,1/3}$). In any case, beating an exponential dependence on k for any algorithm is unlikely [56].

The key empirical insight of this paper is that Turán clique shadows are small for real-world graphs. We explain in more detail in the next section; Fig. 3.3 shows that the shadow sizes are typically less than m, and never more than $10m$.

3.6 Experimental results

Preliminaries: We implemented our algorithms in `C++` and ran our experiments on a commodity machine equipped with a 3.00GHz Intel Core i7 processor with 8 cores and 256KB L2 cache (per core), 20MB L3 cache, and 128GB memory. All code for the experiments is available at: https://bitbucket.org/sjain12/cliquecounting/

We performed our experiments on a collection of graphs from SNAP [159], the largest with more than 100M edges. The collection includes social networks, web networks, and infrastructure networks. Each graph is made simple by ignoring direction. Basic properties of these graphs are presented in Tab. 3.1.

In the implementation of Turán-shadow, there is just one parameter to choose: the number of samples chosen in Step 2 in `sample`. Theoretically, it is set to $(20/\gamma\varepsilon^2)\log(1/\delta)$; in practice, we just set it to 50K for all our runs. Note that γ is not a free parameter and is automatically set in Step 1 of Turán-shadow.

We focus on counting k-cliques for k ranging from 5 to 10. We ignore $k = 3, 4$, since there is much existing (scalable) work for this setting [11, 137, 215]. For the sake of presentation, we showcase results for $k = 7, 10$. We focus on $k = 10$ since no existing algorithm produces results for 10-cliques in reasonable time. We also show specifics for $k = 7$, to contrast with $k = 10$.

Convergence of Turán-shadow: We picked two smaller graphs `amazon0601`

Table 3.1: Graph properties

					k=5			k=7			k=10		
graph	**vertices**	**edges**	**degen**	**max degree**	**estimate**	**% error**	**time**	**estimate**	**% error**	**time**	**estimate**	**% error**	**time**
loc-gowalla	1.97E+05	9.50E+05	51	14730	1.46E+07	0.20	2	4.78E+07	0.36	2	1.08E+08	1.63	3
web-Stanford	2.82E+05	1.99E+06	71	38625	6.21E+08	0.00	20	3.47E+10	0.13	43	6.63E+12	-	52
amazon0601	4.03E+05	4.89E+06	10	2752	3.64E+06	0.93	1	9.98E+05	0.95	1	9.77E+03	0.01	1
com-youtube	1.13E+06	2.99E+06	51	28754	7.29E+06	1.08	7	7.85E+06	1.38	8	1.83E+06	0.20	8
web-Google	8.76E+05	4.32E+06	44	6332	1.05E+08	0.10	2	6.06E+08	0.09	2	1.29E+10	0.82	2
web-BerkStan	6.85E+05	6.65E+06	201	84230	2.19E+10	0.00	101	9.30E+12	1.05	214	5.79E+16	-	262
as-skitter	1.70E+06	1.11E+07	111	35455	1.17E+09	0.01	153	7.30E+10	0.23	164	2.28E+13	-	180
cit-Patents	3.77E+06	1.65E+07	64	793	3.05E+06	0.34	10	1.89E+06	0.83	9	2.55E+03	4.46	9
soc-pokec	1.63E+06	2.23E+07	47	14854	5.29E+07	0.13	42	8.43E+07	0.48	45	1.98E+08	0.01	45
com-lj	4.00E+06	3.47E+07	360	14815	2.46E+11	-	106	5.14E+14	-	153	1.47E+19	-	252
com-orkut	3.07E+06	1.17E+08	253	33313	1.57E+10	0.00	3119	3.61E+11	1.97	5587	3.14E+13	-	9298

Table 3.2: Table shows the sizes, degeneracy, maximum degree of the graphs, the counts of 5, 7 and 10 cliques obtained using Turán-shadow, the percent relative error in the estimates, and time in seconds required to get the estimates. Some of the exact counts were obtained from [105] (where available). This is the first such algorithm that obtains these counts with $< 2\%$ error without using any specialized hardware.

and `web-Google` for which the exact k-clique count is known (for all $k \in [5, 10]$). We choose both $k = 7, 10$. For each graph, for sample size in [10K,50K,100K,500K,1M], we perform 100 runs of the algorithm. We plot the spread of the output of Turán-shadow, over all these runs. The results are shown in Fig. 4.4. The red line denotes the true answer, and there is a point for the output of every single run. Even for 10-clique counting, the spread of 100 runs is absolutely minimal. For 50K samples, the range of values is within 2% of the true answer. This was consistent with all our runs.

Accuracy of Turán-shadow: For many graphs (and values of k), it was not feasible to get an exact algorithm to run in reasonable time. The run time of exact procedures can vary wildly, so we have exact numbers for some larger graphs but could not generate numbers for smaller graphs. We collected as many exact results as possible to validate Turán-shadow. For the sake of presentation, we only show a snapshot of these results here.

For $k = 7$, we collected exact results for a collection of graphs, and for each graph, compared the output of a single run of Turán-shadow (with 50K

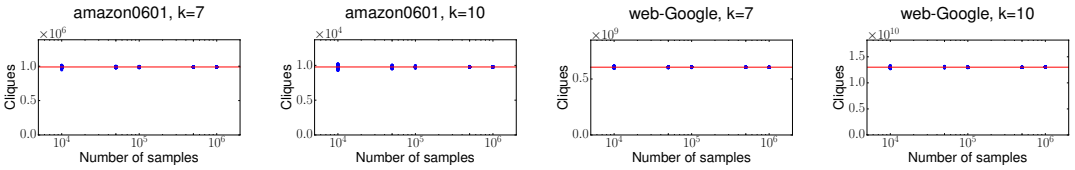


Figure 3.2: Figure shows convergence over 100 runs of TURÁN-SHADOW using 10K, 50K, 100K, 500K and 1M samples each. TURÁN-SHADOW has an extremely low spread and consistently gives very accurate results.

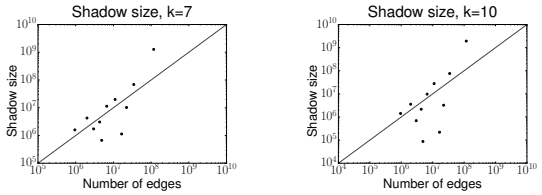


Figure 3.3: Figures show the sizes of the Turán shadows generated for k=7 and k=10 in all the graphs. The runtime of the algorithm is proportional to the size of the shadow and crucially, the sizes scale only linearly with the number of edges.

samples) with the true answer. We compute *relative error*: |true - estimate|/true. These results are presented in Fig. 3.1a. Note that the errors are within 2% in all cases, again consistent with all our runs.

In Tab. 3.1, we present the output of our algorithm for a single run on all instances and $k = 5, 7, 10$. For every graph where we know the true value, we present the relative error. Barring one example (`cit-Patents` for $k = 10$), all errors are less than 2%. Even in the worst case, the error is at most 5%.

Running time: All runtimes are presented in Tab. 3.1. (We show the time for a single run, since there was little variance for different runs on the same graph.) In all instances except `com-orkut`, the runtime was a few minutes, even for graphs with tens of millions of edges. We stress that these are all on a single machine. For `com-orkut`, the runtime is at most 2.5 hours. Previously, such graphs were processed with MapReduce on clusters [105].

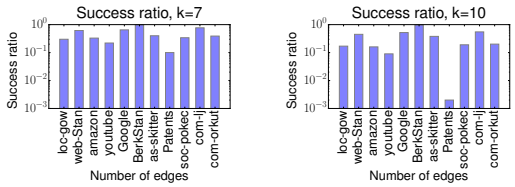
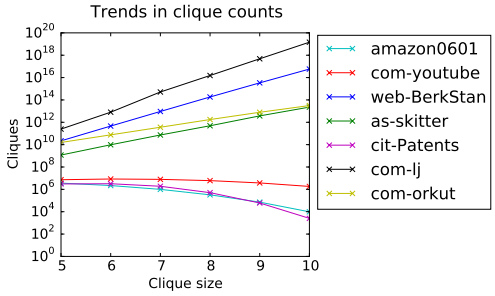


Figure 3.4: Figures show the success ratio (probability of finding a clique) obtained in the sampling experiments in all the graphs.



3.6.1 Comparison with other algorithms

We compare TuránShadow with color coding, edge sampling, GRAFT and the brute force algorithm. All of these are described in 2. Note that the parallel clique counting method of [79] called kClist was developed after this work was published. We provide a comparison with kClist in 5

We focus on $k = 7, 10$ for clarity. In all cases, we simply terminate the algorithm if it takes more than the minimum of 7 hours and 100 times the time required by Turán-shadow. We present the speedup of Turán-shadow with respect to all these algorithms in Fig. 3.1c for k=7. For k=10, for most instances, no competing algorithm terminated.

- $k = 7$ (Fig. 3.1c): Turán-shadow outperformed Color Coding and GRAFT across all instances. Color Coding never gave good accuracy, so we ignore it in our speedup plots. We do note that Edge Sampling gives extremely good performance in some instances, but can be very slow in others. For

amazon0601, com-youtube, cit-Patents, and soc-pokec, Edge Sampling is faster than Turán-shadow. But Turán-shadow handles all these graphs with a minute. The only exception is com-orkut, where GRAFT is much faster than Turán-shadow. We note that all other algorithms can perform extremely poorly on fairly small graphs: Edge Sampling is 10-100 times slower on a number of graphs, which have only millions of edges. On the other hand, Turán-shadow always runs in minutes for these graphs.

- $k = 10$: *No competing algorithm* is able to handle 10 cliques for all datasets, even in 7 hours (giving a speedup of anywhere between 3x to 100x). They all generally fail for at least half of the instances. Turán-shadow gets an answer for com-orkut within 2.5 hours, and handles all other graphs in minutes.

3.6.2 Details about Turán-shadow

Shadow size: In Fig. 3.3, we plot the size of the k-clique Turán shadow with respect to the number of edges in each instance. This is done for $k = 7, 10$. (The line $y = x$ is drawn as well.) As seen from Theorem 3.5.5, the size of the shadow controls the storage and runtime of Turán-shadow. We see how in almost all instances, the shadow size is around the number of edges. This empirically explains the efficiency of Turán-shadow. The worst case is com-orkut, where the shadow size is at most ten times the number of edges.

Success probability: The final estimate of Turán-shadow is generated through sample. We asserted (theoretically) that $O(m)$ samples suffice, and in practice, we use 50K samples. In Fig. 3.4, we plot (for $k = 7, 10$) the empirical probability of finding a clique in Step 6 of sample. The higher this is, the fewer samples we require and the more confidence in the statistical validity of our estimate. Almost all (empirical) probabilities are more than 0.1, and 50K

samples are more than enough for convergence.

Trends in clique numbers: Fig. 5.2b plots the number of k-cliques (as computed by Turán-shadow) versus k. (We do not consider all graphs for the sake of clarity.) Interestingly, there are some graphs where the number of cliques grows exponentially. This is probably because of a large clique/dense-subgraph, and it would be interesting to verify this. For another class of graphs, the clique counts are consistently decreasing. This seems to classify graphs into one of two types. We feel further analysis of these trends would be interesting, and Turán-shadow can be a useful tool for network analysis.

3.7 Demonstration of clique sampling

An important observation about TuránShadow is that it not only estimates the count of k-cliques but provides an efficient method to sample k-cliques uniformly at random. This is useful in applications that want to incorporate the information about higher-order structures like cliques in their modeling/analysis tasks. In joint work with Huda Nassar (Stanford University), Prof. David Gleich (Purdue University), Prof. Austin Benson (Cornell University) and Caitlin Kennedy (Salesforce Inc.) we demonstrate one such application of cliques in better graph visualization.

The goal in graph visualization is to give a 2D representation of a graph that meaningfully shows connections and communities in a graph. Having a meaningful layout is often useful to help interpret the results from network sciences tasks such as community detection and link prediction. There are several existing graph visualization techniques in the literature that are based on spectral methods, graph embeddings, or optimizing graph distances. Despite the large number of methods, it is still often challenging or extremely time consuming to produce

meaningful layouts of graphs with hundreds of thousands of vertices. Existing methods often either fail to produce a visualization in a meaningful time window, or produce a layout colorfully called a "hairball", which looks like a filled ellipse with small hairs emerging that does not illustrate any internal structure in the graph.

We show that adding higher order information based on cliques to a classic eigenvector based graph visualization techniques enables it to produce meaningful plots of large graphs. As noted earlier, one of the hallmarks of real world graphs is that they are not random and the edges are concentrated in dense pockets. For example, in social networks, groups with similar location or interests or belonging to some organization tend to be densely connected. This suggests that we could leverage the information of these dense subgraphs to group vertices in such a way that densely connected vertices are close-by in the representation.

We employ TuránShadow to sample a number of $k-$cliques (for a number of k) u.a.r. and add the higher order information based on the sampled cliques to a classic eigenvector based graph visualization technique. Essentially, we look at the number of k-cliques that any given edge (u, v) participates in and use that as its weight. We observed that using higher order information improved the graph layout over other existing methods like Node2Vec [117], LGL [6] and DRL [172] and arguably, produces more meaningful plots. As an example, for the Rice31 graph from the Facebook100 dataset [239](a set of 100 graphs representing the Facebook social network from 100 U.S. educational institutions) the representations obtained looked as shown in Fig. 3.6. The dataset contains labeled information about each node (student) in the graph, and the label we use as a prior on community structure is the dorm a student belonged to. In this scenario, our expectation is that if two students lived in the same dorm, it is likely

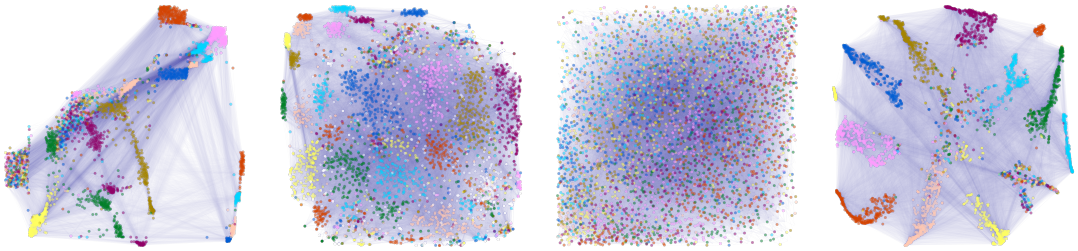


Figure 3.6: Figures show the graph visualizations obtained for Rice31 graph. LAPTS uses the information about cliques obtained using TuránShadow

that they know each other, and so we expect them to appear next to each other in the graph layout. As we can see, incorporating information about k-cliques indeed helps provide better separation among the clusters.

This work was published at The Web Conference, 2020.

3.8 Future work

Some natural extensions to Turán-shadow come to mind. Firstly, we have directly implemented the theoretical algorithm using Turán densities. It is likely that this is overkill, and we can design an even faster heuristic by modifying the density threshold for a shadow. Also, our algorithm currently creates a new shadow for different values of k. We believe that these can all be "folded together" to get one algorithm that gives all k-clique counts for $k \leq 10$.

An intriguing question is whether the approaches in this paper can be used for finding dense subgraphs (not just cliques).

In follow up work to TuránShadow, in 5, we show that it is possible to count cliques without enumerating them and we give an even faster algorithm which uses a different set of techniques. Nonetheless, the techniques used in TuránShadow lend themselves to algorithms for other problems. We demonstrate one such

application in 4 where we use TuránShadow to count the number of near-cliques. We also give an online version of TuránShadow that does not require us to store the entire Turán shadow before sampling but rather, computes the estimate of clique counts as the shadow is being generated, thus reducing the memory footprint of the algorithm.

Chapter 4

Counting near-cliques

4.1 Introduction

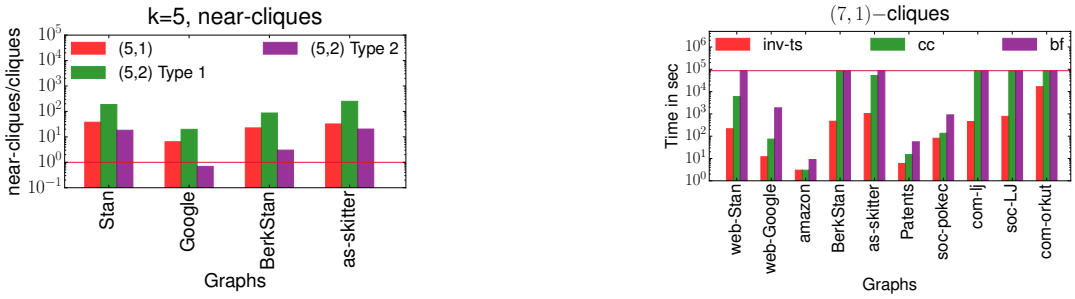
The requirement that every edge in a clique be present is perhaps excessively rigid. Data is often noisy or incomplete, and it is likely that cliques that are missing even an edge or two are significant. Hence, it is important to also look at counts of patterns that are extremely close to being cliques. We will call these structures near-cliques but they are also known as quasi-cliques [163, 190] and defective cliques [267] and have several applications ranging from clustering to prediction. Recent work on has used the fraction of near-cliques to k-cliques to define higher order variants of clustering coefficients [265].

In the bioinformatics literature, near-cliques (or defective cliques, as they are known) have been used to predict missed protein-protein interactions in noisy PPI networks [267] and have been shown to have good predictive performance. An alternative viewpoint of looking at near-cliques views them as dense subgraphs. Mining dense subgraphs is an important problem with many applications in Network Analysis. [21, 57, 107, 153, 209]

Counting cliques is already challenging, and counting near-cliques introduces

more challenges. Most importantly, near-cliques do not enjoy the recursive structural property of cliques - that a subset of a clique is also a clique. This rules out most recursive backtracking algorithms for clique counting. Moreover, empirical evidence suggests that the number of near-cliques in real world datasets is order of magnitudes higher than that of cliques, making the task of counting them equally difficult if not more. Fig. 4.1a shows the ratio of 3 different types of near-cliques to the number of k-cliques for $k = 5$ for 4 real world graphs. The number of near-cliques is often ten times higher than the number of k-cliques.

There are several different ways of defining near-cliques. [240] define α-quasi-cliques as cliques that are missing a α fraction of the edges. Other formulations define them in terms of graph properties like degree of every vertex in the near-clique or diameter of the near-clique. A set S of size n is called a $k-$plex if every member of the set is connected to $n - k$ others. A k-club is a subset S of nodes such that in the subgraph induced by S, the diameter is k or less. All these formulations have the common property that they represent a clique that is missing a few edges. We formulate near-cliques in a slightly different way, as cliques that are missing 1 or 2 edges. The advantage of defining them this way is that they allow us to leverage the machinery of clique counting. Every such near-clique has a smaller clique contained in it. By sampling the smaller cliques and using them as hints to find near-cliques, we give an estimate for the total number of near-cliques. In §4.6.1 we show an interesting application of such near-cliques where we run our algorithm on a citation network to discover papers that perhaps should have cited other papers but did not.



(a) Ratios **(b)** Timings

Figure 4.1: Fig. 4.1a shows the ratio of number of different types of near-cliques to k-cliques for $k = 5$ in four real world graphs. The red line indicates ratio $= 1$. In most cases the number of near-cliques is at least of the same order of magnitude as number of k-cliques, if not more. Fig. 4.1b shows the time required by Inverse-TS (inv-ts), color-coding (cc) and brute force (bf) to estimate the number of $(7, 1)$-cliques in 10 real world graphs. The $y-$axis shows time in seconds on a log scale. The red line indicates 86400 seconds (24 hours). All experiments that ran for more than 24 hours were terminated. Inverse-TS terminated in minutes in all cases except com-orkut, giving a speedup of anywhere between 3x-100x.

4.1.1 Problem description

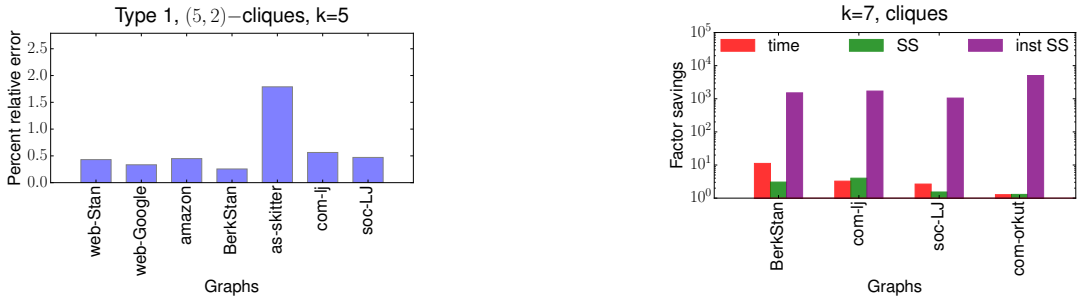
A k-clique is a set of k vertices such that there is an edge between all pairs of vertices belonging to the set. We define $(k, 1)$-clique and $(k, 2)$-clique below. For the rest of this paper, whenever we say near-cliques, we will imply the following 3 kinds of near-cliques (unless mentioned otherwise)

Definition 4.1.1. *A $(k, 1)$-clique is a k-clique with exactly 1 edge missing.*

For $(k, 2)$-cliques, there are 2 configurations possible - one in which the missing edges share a vertex, and one in which they don't.

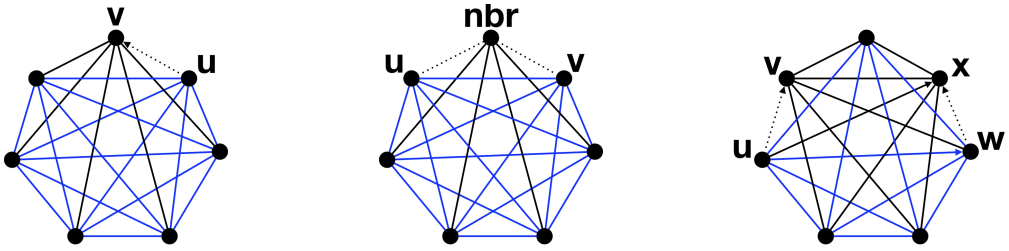
Definition 4.1.2. *A Type 1 $(k, 2)$-clique is a k-clique with exactly 2 edges missing such that the missing edges share a vertex.*

Definition 4.1.3. *A Type 2 $(k, 2)$-clique is a k-clique with exactly 2 edges missing such that the missing edges do not share a vertex.*



(a) Error **(b)** TS vs Inverse-TS

Figure 4.2: Fig. 4.2a shows the percentage error in the estimates for Type 1 $(k, 2)$-cliques for $k = 5$ obtained using Inverse-TS. As we can see, the error is $< 2\%$ and in most cases $< 1\%$. Fig. 4.2b shows the savings in time and space when using Inverse-TS (500000 samples) vs when using TuránShadow (50000 samples) to estimate the number of 7-cliques in 4 of the largest real world graphs we experimented with. The green bars show the factor savings in the percentage of the Turán Shadow that was explored (factor of 2-10). The purple bar shows the factor saving in the maximum amount of space required for the Turán Shadow at any instant.



(a) $(7, 1)$ **(b)** Type 1 $(7, 2)$ **(c)** Type 2 $(7, 2)$

Figure 4.3: Near-7-cliques. Dotted lines indicate the missing edges. Blue lines mark the contained clique.

The different types of near-cliques are shown in Fig. 4.3. We want to estimate the number of $(k, 1)$-cliques and $(k, 2)$-cliques in G. Note that all our near-cliques are induced and obtaining counts of non-induced near-cliques is simply a matter of taking a linear combination of the number of k-cliques and near-cliques. For the sake of brevity, we skip a detailed discussion.

We stress that we make no distributional assumption on the graph. All probabilities are over the internal randomness of the algorithm itself (which is independent of the instance).

4.1.2 Our contributions

We provide a randomized algorithm based on TuránShadow called PEANUTS which estimates the counts of $(k, 1)$-cliques and $(k, 2)$-cliques. In addition, we also provide a heuristic algorithm called Inverse-TS based on PEANUTS which takes roughly the same time as PEANUTS (and in some cases, upto 10x less time) but drastically reduces the space required. Our implementation of Inverse-TS on a commodity machine showed significant savings in terms of time in obtaining counts of near-cliques over other methods like color-coding and brute force counting and showed consistently low error over 100s of runs of the algorithm.

- **Leveraging cliques for near-cliques:** Data being noisy, cliques are brittle and as a result, number of near-cliques is often very large. However, it is not at all clear how one can count their number without looking at every set of $k-$vertices, which is computationally very expensive. PEANUTS uses the fact that near-cliques themselves contain cliques, and leverages TuránShadow to count near-cliques. There exist algorithms for generic pattern counting which can be used for counting near-cliques but there is no known algorithm dedicated to finding near-cliques that exlploits the clique-like structure of near-cliques to give a faster

estimate.

- **Extremely fast:** PEANUTS is based on the observation that every near-clique contains a smaller clique. Thus, we can use cliques as clues for finding near-cliques. We leverage a fast clique-counting algorithm (TuránShadow) to achieve fast and accurate near-clique counting. Fig. 4.1b shows the time taken by Inverse-TS, color-coding (cc) and brute force (bf) to count the number of $(7, 1)$-cliques for a variety of graphs. Inverse-TS is able to estimate their number to within 2% error in a graph (com-lj) with 4 million vertices and 34 million edges in 452 seconds which is at least 100 times faster than cc and bf. As we will show later, similar performance is found in the estimation of other near-cliques and on other graphs.

- **Extremely accurate:** Similar to TuránShadow, Inverse-TS uses the seminal result from extremal combinatorics, called Turán's theorem which allows for efficiently sampling cliques, which translates to fast and accurate estimation of the number of near-cliques. Fig. 4.2a shows the error in the estimate obtained for number of Type 2 $(5, 2)-$cliques (a specific configuration of $(5, 2)-$cliques) in a variety of graphs using Inverse-TS. As we can see, all the errors were within 2%. Moreover, unlike color-coding, Inverse-TS allows us to control the number of samples we take, and even using 500K samples, Inverse-TS was more accurate and took less time than color-coding (6.5).

For many of the graphs we experimented with, the brute force algorithm had not terminated within 1 day and thus was unable to give us ground truth values, but in the cases where the algorithm did terminate, we saw that Inverse-TS gave $< 5\%$ error and mostly $< 2\%$. For the cases where the brute force algorithm did not terminate, we looked at the output of 100 runs of our algorithm. In all cases, the algorithm showed very good convergence properties (more details in §6.5).

- **Excellent space efficiency:** TuránShadow requires that the entire shadow be generated and stored, which for a graph with 100s of millions of edges can potentially require large amount of memory. Our practical implementation of Inverse-TS addresses this by removing the separation in the Shadow construction and sampling phases and instead, performs sampling while the shadow is being constructed in an online fashion. This eliminates the need for storing the entire Shadow and consequently gives savings of orders of magnitude in space required. The purple bars in Fig. 4.2b show the factor savings in the maximum shadow size required to be stored at any point (instantaneous shadow size or inst SS) for Inverse-TS vs the space required by TuránShadow. There is atleast 100x savings in space using Inverse-TS.

- **Comparison with other algorithms:** We do a thorough analysis of Inverse-TS by deploying it on a number of real-world graphs of varying sizes. In most cases we observed that Inverse-TS was considerably fast while showing consistently low error over 100s of runs of the algorithm. We also do a thorough comparison of Inverse-TS with other generic pattern-counting algorithms like color-coding. Fig. 4.1b shows the time required for counting $(7, 1)$-cliques by the different methods. Across all of our experiments we observe that Inverse-TS was at least 10 times faster on most graphs as compared to other algorithms.

- **All code and data available:** All the datasets we used are publicly available at [159] and our code is available at https://bitbucket.org/sjain12/peanuts/.

4.1.3 Related Work

Pattern counting, also known as graphlet counting or motif counting has been an important tool for graph analysis. It has been used in bioinformatics [179, 199,

260], social sciences [126], spam detection [30], graph modeling [214], etc. Triangle counting, and more recently, clique counting have gained a lot of attention [79, 105, 135] due to their special role in characterizing real-world graphs. Clique counts have been employed in applications such as discovery of dense subgraphs [208, 244], in topological approaches to network analysis [224], graph clustering [265] among others. More generally, motif counts have been used in clustering [247, 265], evaluation of graph models [214, 222], classification of graphs [249] etc.

On the theoretical side, several motif-counting algorithms exist [61, 77, 265]. On the more practical side, only recently, efficient methods for counting graphlets upto size 5 [124, 137, 195, 255] have been proposed. For patterns of larger sizes, two widely used techniques are the MCMC [120, 253] and color-coding (CC) of [20]. However, as shown in [45], MCMC based methods have poorer accuracy for the same running time than CC and for patterns of sizes greater than 5, CC is also generally quite inefficient, as we will show in our results. Motif counting has been studied in streaming [42, 143] and distributed settings [96] and in temporal networks [187].

All these methods are geared towards counting arbitrary patterns with upto 6 nodes but none of these methods scale beyond 6 nodes. Moreover, these are generic pattern counting methods that do not utilize the clique-like nature of near-cliques to give more efficient methods. Ours is the first work to do so.

Dense subgraph algorithms: The notion of dense subgraphs as near-cliques was introduced by Tsourakakis et. al. in [240]. There are several different formulations of dense subgraphs, many of which are NP-Hard (indeed, even the problem of finding the densest subgraph on k vertices, known as the densest-k-subgraph is NP-Hard [208]). The algorithms of Andersen and Chellapilla [22], Rossi et al. [205], and Tsourakakis et al. [240, 244] provide

practical algorithms for some of the formulations. However, most of them focus on finding or approximating the densest subgraph rather than giving global stats.

4.2 Main ideas

The starting point of our result is the TuránShadow algorithm from 3 for estimating the number of k-cliques in a graph. TuránShadow is based on a seminal theorem of Turán and Erdös that says that: if the edge density of an $n-$vertex graph is greater than $1-1/(k-1)$ (the Turan density), then the graph is guaranteed to have many ($O(n^{k-2})$) k-cliques. This implies that if we randomly sample a k-vertex set from the graph, the probability of it being a k-clique would be high. TuránShadow exploits this fact by splitting G into (possibly overlapping) Turan-dense subgraphs such that there is a one-to-one correspondence between the cliques of a specific size in each subgraph, and the number of k-cliques in G. The set of all such subgraphs of G is called the Turán Shadow of G. Essentially, TuránShadow reduces the search space for k-cliques in G from 1 large sparse graph to several dense subgraphs.

More importantly though, for any h, TuránShadow provides an efficient way of sampling a u.a.r. h-clique from G. Let C_h be the set of all h-cliques in G and let $f : C_h \to \mathbb{R}^+$ be a bounded function over all h-cliques, then we can obtain an unbiased estimate for $F = \sum_{K \in C_h} f(K)$ by obtaining the average of f over a set of uniformly sampled h-cliques and scaling by the total number of h-cliques. In other words, we can use this clique sampler to obtain an unbiased estimate of the sum (and mean value) of any bounded function over h-cliques. We exploit this fact to obtain an estimate of the number of near-cliques.

To estimate the number of $(k,1)$-cliques, we make the following observation: Every $(k,1)$-clique has exactly two $k-1$-cliques embedded in it. Let $C_{k_{-1}}$ be

the set of $(k,1)$-cliques in G and C_{k-1} be the set of $k-1$-cliques in G, and $\forall K \in C_{k-1}$, let $f(K)$ = number of $(k,1)$-cliques that clique K is contained in, then $\sum_{K \in C_{k-1}} f(K) = 2|C_{k-1}|$. However, since every $(k,1)$-clique is counted twice, the variance of the estimator can be pretty large. We observe that if the missing edge in a $(k,1)$-clique is $(u,v), u < v$, exactly one of the $k-1$-cliques contains u and the other contains v. In order to reduce the variance, we define $f(K)$ = number of $(k,1)$-cliques that clique K is contained in, such that $u \in K$ i.e. we break ties based on the direction of the missing edge. With this formulation, $\sum_{K \in C_{k-1}} f(K) = |C_{k-1}|$.

For $(k,2)$-cliques, there are 2 possible configurations, as shown in Fig. 4.3. Type 1 consists of exactly one $k-1$-clique embedded in it. Hence, we set $f(K)$ = number of Type 1 $(k,2)$-cliques that a given $k-1$-clique K is contained in. Type 2 $(k,2)$-cliques are a bit more complicated. A Type 2 $(k,2)$-clique has exactly four $k-2$-cliques embedded in it. If the edges (u,v) and (w,x) are missing, then there is an induced cycle involving u, v, w and x and every edge of this cycle gives a different $k-2$-clique of the four $k-2$-cliques embedded in the $(k,2)$-clique. Let $min(u,v,w,x) = u$ and let $min(w,x) = w$. Then, for $k-2$-clique K, we set $f(K)$ = number of $(k,2)$-cliques such that $u, w \in K$.

As long as f is bounded and is a "well behaved function" i.e. has low variance, we can efficiently estimate F using TuránShadow as a black box. Improving the running time of the black box only improves the running time of the overall algorithm. We observe that in TuránShadow, most of the time is spent in constructing the Shadow, but only a small fraction of it is used to gather samples. Thus, if we can first sample and determine which areas of the Shadow the samples lie in, we can save time by developing only those parts of the Shadow instead of developing the whole Shadow. Additionally, when the number of samples are

fixed (as is the case in the practical implementation of our algorithm), we can interleave the development of the parts of the Shadow with sampling for h-cliques from those parts, thus obtaining our estimate of F in an online fashion. This leads to considerable savings in space and time.

Outline: In §6.2 we set some basic notation. In §4.4 we show our basic framework PEANUTS and an optimized version of it called Inverse-TS. Depending on which type of pattern we want to count, we propose and analyze different counters in §4.5. Finally, in §6.5 we provide a detailed experimental study of Inverse-TS and its comparison with the state-of-the-art.

4.3 Preliminaries

We set some notation. The input graph G has n vertices and m edges. We will assume that $m \geq n$. Let α be the degeneracy of the graph. Recall that the degeneracy is the maximum outdegree of any vertex when the edges of the graph are oriented according to the degeneracy ordering of the vertices in G. Let $N_v(G)$ represent the neighborhood of v and let $N_v^+(G)$ represent the outneighborhood of v when the vertices are ordered by degeneracy.

We use “u.a.r." as a shorthand for “uniform at random".

We will be using the following (rescaled) Chernoff bound.

Theorem 4.3.1. *[Theorem 1 in [84]] Let $X_1, X_2, \ldots, X_k$ be a sequence of iid random variables with expectation μ. Furthermore, let $X_i \in [0, B]$. Then, for $\varepsilon < 1$, $\Pr[|\sum_{i=1}^{k} X_i - \mu k| \geq \varepsilon \mu k] \leq 2\exp(-\varepsilon^2 \mu k/3B)$.*

4.4 Main algorithm

At the core of TuránShadow lies an object called the shadow. We define an analogous structure called Prefixed-Shadow.

Definition 4.4.1. *Let $C_k(G)$ be the set of all $k-$cliques in G. A k-clique Prefixed-Shadow $\boldsymbol{S}$ for graph G is a set of triples $\{(P_i, S_i, \ell_i)\}$ where $P_i \subseteq V$, $S_i \subseteq V$ and $\ell_i \in \mathbb{N}$ such that $\forall (P_i, S_i, \ell_i) \in \boldsymbol{S}, \forall c \in C_{\ell_i}(S_i), P_i \cup c$ is a unique k-clique in G and there is a bijection between $C_k(G)$ and $\bigcup_{(P_i,S_i,\ell_i)\in \boldsymbol{S}} \bigcup_{c \in C_{\ell_i}(S_i)} P_i \cup c$.*

Moreover, if the multiset $\{(S_i, \ell_i)\}$ is such that $\forall (S_i, \ell_i), \rho_2(S_i) > 1 - 1/(\ell_i - 1)$ where $\rho_2(S_i)$ represents the edge density of S_i, then $\boldsymbol{S}$ is a k-clique Prefixed-Turán-Shadow of G.

It is easy to see that $\{(v, N_v^+, h-1)\}$ is an h-clique Prefixed-Shadow of G.

We will briefly recap how TuránShadow constructs the shadow. It orders the vertices of G by degeneracy and converts it into a DAG. As shown in [105], to count k-cliques in G it suffices to count the number of $k-1$-cliques in the outneighborhood of every vertex. Hence, for every vertex $v \in V$, TuránShadow counts the number of k-cliques with v as the lowest order vertex by looking at the number of $k-1$-cliques in the outneighborhood of v, and it applies this procedure recursively. When the outneighborhood becomes dense enough, instead of continuing to expand the partial clique, it adds the outneighborhood to the shadow and continues until there are no more outneighborhoods left to be added to the shadow.

Algorithm PrefixedTuránShadowFinder carries out exactly the same steps as *Shadow-Finder* in 3, except that at each stage it also maintains the partial clique P.

Algorithm 9: PrefixedTuránShadowFinder(G, k)

Initialize $\boldsymbol{T} = \{(\emptyset, V, k)\}$ and $\boldsymbol{S} = \emptyset$
While $\exists (P, S, \ell) \in \boldsymbol{T}$ such that $\rho_2(S) \leq 1 - \frac{1}{\ell - 1}$
    Construct the degeneracy DAG $D(G|_S)$
    Let N_s^+ denote the outneighborhood (within $D(G|_S)$) of $s \in S$
    Delete (P, S, ℓ) from $\boldsymbol{T}$
    For each $s \in S$
        If $\ell \leq 2$ or $\rho_2(N_s^+) > 1 - \frac{1}{\ell - 2}$
            Add $(P \cup \{s\}, N_s^+, \ell - 1)$ to $\boldsymbol{S}$
        Else, add $(P \cup \{s\}, N_s^+, \ell - 1)$ to $\boldsymbol{T}$
Output $\boldsymbol{S}$

Claim 4.4.2. *Given a graph G and integer k, PrefixedTuránShadowFinder returns a k-clique Prefixed-Turán-Shadow of G. Its running time is $O(|G|^{k+1})$.*

Proof. When the function returns, T is empty, and any element $(P, S, \ell) \in \boldsymbol{S}$ was added to $\boldsymbol{S}$ only when $\rho_2(S) > 1 - 1/(\ell - 1)$. Thus, if $\boldsymbol{S}$ is a Prefixed-Shadow, it is also a Prefixed-Turán-Shadow.

By Theorem 3.5.1, multiset $\{(S, \ell)\}$ is a shadow and hence, there is a bijection between $C_k(G)$ and $\bigcup_{(P,S,\ell) \in \boldsymbol{S}} C_\ell(S)$. Thus, it suffices to prove that $\forall (P, S, \ell) \in \boldsymbol{T} \cup \boldsymbol{S}, \forall c \in C_\ell(S), P \cup c$ is a unique k-clique in G. We will prove this using induction. At the start of the first iteration, P is empty, $S = V$ and $\ell = k$, $\boldsymbol{S} = \{(P, S, \ell)\}$ and $\boldsymbol{T}$ is empty. Thus, for the base case, the hypothesis is trivially true.

Suppose the hypothesis is true at the start of some iteration and lets say element $E = (P', S', \ell')$ is deleted from $\boldsymbol{T}$ at the start of this iteration. Each $E_s = (P' \cup \{s\}, N_s^+, \ell' - 1)$ for $s \in S'$ is added to $\boldsymbol{S}$ or to $\boldsymbol{T}$. Let $\mathcal{K}(E) = \{P' \cup c | c \in C_{\ell'}(S')\}$ denote the set of k-cliques obtained from E. It suffices to prove that: (i) for any $k-$clique $K \in \mathcal{K}(E), K \in \bigcup_s \mathcal{K}(E_s)$, (ii) $|\mathcal{K}| = \sum_s |\mathcal{K}(E_s)|$.

Consider a k-clique $K = P' \cup c, c \in C_{\ell'}(S')$. Let s be the lowest order vertex in c according to the degeneracy ordering in $G|_{S'}$. Then, $c \setminus \{s\}$ is an $\ell - 1$-clique

in N_s^+. Thus, $K \in \mathcal{K}(E_s)$. Additionally, for $c \in C_{\ell'}(S')$ the smallest vertex in c defines a partition over $C_{\ell'}(S')$. Hence, $|C_{\ell'}(S')| = \sum_{s \in S'} |C_{\ell'-1}(N_s^+)|$ i.e. $|\mathcal{K}(E)| = \sum_{s \in S'} |\mathcal{K}(E_s)|$. Hence, proved.

The out-degree of every vertex is at most $|V|$ and the depth of the recursive calls is atmost $k-1$. When processing an element (P, S, ℓ) it constructs the graph $G|_S$ which takes time atmost $|V|^2$ since it queries every pair of vertices in S and $|S| < |V|$. Thus, the time required is $O(|V|^{k+1})$. □

Algorithm 10: Sample-clique($\mathbf{S}$)
Inputs: $\mathbf{S}$: $k-$clique Prefixed-Turán-Shadow of some graph G
Output: B: $k-$vertex set

1 Let $w(\boldsymbol{S}) = \sum_{(P',S',\ell') \in \mathbf{S}} \binom{|S'|}{\ell'}$
2 Set probability distribution D over $\mathbf{S}$ such that $(P, S, \ell) \in \mathbf{S}$ is sampled with probability $\binom{|S|}{\ell}/w(\boldsymbol{S})$
3 Sample a (P, S, ℓ) from D
4 Choose a u.a.r. $\ell-$tuple c from S
5 Let $B = P \cup \{c\}$
6 return B

Claim 4.4.3. *The probability of any k-clique K in G being returned by a call to Sample-clique is $\frac{1}{w(\boldsymbol{S})}$.*

Proof. Let $E = (P, S, \ell) \in \boldsymbol{S}$ where $\boldsymbol{S}$ is the k-clique Prefixed-Shadow of some graph G. Note that $w(\boldsymbol{S}) = \sum_{(P',S',\ell') \in \mathbf{S}} \binom{|S'|}{\ell'}$. Let c be an $\ell-$clique in S and let $K = P \cup c$ then K must be a unique $k-$clique in G.

$Pr(\text{K is sampled}) = Pr(\text{E is sampled from D}) * Pr(\text{c is sampled from S}) = \frac{\binom{|S|}{\ell}}{w(\boldsymbol{S})} * \frac{1}{\binom{|S|}{\ell}} = \frac{1}{w(\boldsymbol{S})}$. Thus, every $k-$clique in G has the same probability of being returned by Sample-clique. □

We will first describe PEANUTS. Essentially, it constructs the

Prefixed-Turán-Shadow of G, samples h-cliques, obtains f for the sampled h-clique and estimates the value of F.

Algorithm 11: PEANUTS$(G, h, s, Func)$
Inputs: G: input graph, h: clique size // $= k$ for cliques, $k-1$ for $(k,1)$-clique and Type 1 $(k,2)$-clique, $k-2$ for Type 2 $(k,2)$-clique
s: budget for samples, $Func$: Function that returns $f(K)$ for h-clique K.
Output: $\hat{F}$: estimated F

$\mathbf{S} =$ **PrefixedTuránShadowFinder**$(\mathbf{G}, \mathbf{h})$
Let $w(\boldsymbol{S}) = \sum_{(P,S,\ell)\in\mathbf{S}} \binom{|S|}{\ell}$
For $i = 1, 2, ..., s$:
    $K = Sample - clique(\mathbf{S})$
    If K is a clique, set $X_i = Func(G, K)$
    else set $X_i = 0$
    $W = W + X_i$
let $\hat{F} = \frac{W}{s} w(\boldsymbol{S})$
return $\hat{F}$

Theorem 4.4.4. *Let f be a function over h-cliques, bounded above by B such that given an h-clique, it takes $O(T_f)$ time to obtain the value of f. Let $\hat{F}$ be the output of PEANUTS, then $\boldsymbol{E}[\hat{F}] = F$. Moreover, given any $\varepsilon > 0, \delta > 0$ and number of samples $s = 3w(\boldsymbol{S})B\ln(2/\delta)/\varepsilon^2 F$, then with probability at least $1-\delta$ (this probability is over the randomness of PEANUTS; there is no stochastic assumption on G), $|\hat{F} - F| \leq \varepsilon F$.*

Let $\boldsymbol{S}$ denote the h-clique Turán shadow of G and $\text{size}(\boldsymbol{S}) = \sum_{(S,\ell)\in\boldsymbol{S}} |S|$. *The running time of PEANUTS is $O(\alpha\text{size}(\boldsymbol{S}) + sT_f + m + n)$ and the total storage is $O(\text{size}(\boldsymbol{S}) + m + n)$.*

Proof. The X_i are all iid random variables and by the arguments in Claim 4.4.3, every h-clique in G has the same probability of being returned by Sample-clique. $\mathbf{E}[X_i] = \sum_{K\in C_k(G)} \frac{f(K)}{w(\boldsymbol{S})} = \frac{F}{w(\boldsymbol{S})}$. Suppose $X_i \in [0, B]$. By Theorem 6.2.1, $Pr[|\sum_{i=1}^{s} X_i - s\mathbf{E}[X_i]| \geq \varepsilon s\mathbf{E}[X_i] \leq \delta$ when $s = 3w(\boldsymbol{S})B\ln(2/\delta)/\varepsilon^2 F$.

The running time and storage required are a direct consequence of the running time and storage required for TuránShadow. The only difference is the addition of sT_f in the running time which is the time required to obtain f for s samples.

□

4.4.1 Inverse-TS

We observed that with TuránShadow, bulk of the time is spent in building the tree, and only a small fraction is needed for sampling. To give a few examples, for the web-Stanford graph, construction of the shadow took 155 seconds for approximating number of 7 cliques, while taking $50K$ samples required 0.2 seconds. Similar results were observed for all other graphs we experimented with. Thus, naturally, to optimize the performance of TuránShadow it would be beneficial to minimize the fraction of the shadow that is required to be built. Consider one extreme of minimizing building the shadow - we will call it level 1 sampling. Let N_v^+ be the outneighborhood of v in DG, $\Phi_v = \binom{|N_v^+|}{h-1}$ and $\Phi = \sum_v \Phi_v$. $\{(v, N_v^+, h-1)\}$ is an h-clique Prefixed-Shadow of G. If we sample a v with probability proportional to Φ_v, and sample $h-1$-tuple of vertices from N_v^+ u.a.r., the probability of sampling a particular $h-1$-clique in N_v^+ would be $\Phi_v/\Phi * 1/\Phi_v = 1/\Phi$. If there are C_h h-cliques in G then the probability that a sampled set of $h-$vertices is a clique is C_h/Φ (we call this the success ratio). Hence, number of samples required to find a h-clique would be $O(\Phi/C_h)$. But Φ is typically very large compared to C_h and hence the number of samples required would be very large. In other words, most of the $h-$vertex sets picked will not be cliques.

TuránShadow remedies this by first finding the Turán shadow and then sampling within the subgraphs of the shadow which are dense and hence require

lesser samples to find a k-clique. Thus, TuránShadow saves on the number of samples required at the cost of building the shadow.

The advantage of level 1 sampling is that we do not need to spend time finding the Turán Shadow. We mimic the process of sampling an h-clique from this Prefixed-Shadow, but boost the success ratio by using the latter approach. In particular, we sample a v proportional to $\Phi_v = \binom{|N_v^+|}{h-1}$, and obtain the $h-1$-clique Prefixed-Turán-Shadow $\boldsymbol{S}$ of N_v^+. Suppose the shadow size $\phi_v = \sum_{(P,S,\ell)\in\boldsymbol{S}} \binom{S}{\ell}$ then probability of sampling a $h-1$-clique $= C_{h-1}(G|_{N_v^+})/\phi_v$. Thus, the success ratio goes from $C_{h-1}(G|_{N_v^+})/\Phi_v$ to $C_{h-1}(G|_{N_v^+})/\phi_v$. Since ϕ_v is typically much smaller than Φ_v, the success ratio is much improved. However, to account for the fact that we are now sampling u.a.r. in a search space of size ϕ_v and not Φ_v, we give a smaller weight (ϕ_v/Φ_v) to every clique obtained from N_v^+.

Algorithm 12: Inverse-TS$(G, h, s, Func)$

Order G by degeneracy and convert it to a DAG DG.
Let M be a map, $W = 0$
Set probability distribution D over V where $p(v) = \sum_v \Phi_v/\Phi$.
For $i = 1, 2, ..., s$:
    Independently sample a vertex v from D.
    If $M[v]$ exists, set $\boldsymbol{S} = M[v]$
    else
        $\mathbf{S} = PrefixedTuránShadowFinder(G_{|N_v^+}, h-1)$
        $M[v] = \mathbf{S}$
    Let $\phi_v = \sum_{(P,S,\ell)\in\mathbf{S}} \binom{|S|}{\ell}$
    Let $K = \{v\} \cup Sample - clique(\mathbf{S})$
    If K is a clique, set $X_i = \frac{\phi_v}{\Phi_v} * Func(G, K)$
    else set $X_i = 0$
    $W = W + X_i$
let $\hat{F} = \frac{W}{s}\Phi$
return $\hat{F}$

For an element $E = (P, S, \ell) \in \boldsymbol{S}$ where $\boldsymbol{S}$ is the $k-$clique

Prefixed-Turán-Shadow of a graph G, let $\mathcal{K}(E) = \{P \cup c, c \in C_\ell(S)\}$ denote the set of k-cliques obtained from E.

Lemma 4.4.5. *Let $\hat{F}$ be the value returned by Inverse-TS. Then $\boldsymbol{E}[\hat{F}] = F$.*

Proof. Consider an h-clique $K \in C_h(G)$ and let v be the lowest order vertex according to degenerecy ordering of vertices in G. Let $E = (v, N_v^+, h-1)$ then, $K \in \mathcal{K}(E)$.

Let $\boldsymbol{S}_v$ be the $h-1$-clique Prefixed-Turán-Shadow of $G|_{N_v^+}$ and let $E_v = (P, S, \ell)$ be the element in $\boldsymbol{S}_v$ such that $K = v \cup P \cup c, c \in C_\ell(S)$.

$Pr(K \text{ is sampled in Step 11}) = Pr(E \text{ is sampled}) * Pr(E_v \text{ is sampled}) * Pr(c \text{ is sampled}) = \frac{\Phi_v}{\Phi} * \frac{\binom{|S|}{\ell}}{\phi_v} * \frac{1}{\binom{|S|}{\ell}} = \frac{\Phi_v}{\Phi\phi_v}$ Thus, $\mathbf{E}[X_i] = \sum_{v \in V} \sum_{K \in \mathcal{K}(E_v)} \frac{\Phi_v}{\Phi\phi_v} \frac{\phi_v}{\Phi_v} f(K) = \sum_{K \in C_k(G)} \frac{f(K)}{\Phi} = \frac{F}{\Phi}$

Moreover, $W = \sum_{i=1}^{s} X_i$. Therefore, $\mathbf{E}[W] = \mathbf{E}[\sum_{i=1}^{s} X_i] = \sum_{i=1}^{s} \mathbf{E}[X_i] = s\frac{F}{\Phi}$.

Hence, $\mathbf{E}[\hat{F}] = \mathbf{E}[\frac{W}{s}\Phi] = F$. □

Theorem 4.4.6. *Let f be a function over h-cliques, bounded above by B such that given an h-clique, it takes $O(T_f)$ time to obtain the value of f. Given any $\varepsilon > 0, \delta > 0$ and number of samples $s = 3\Phi B \ln(2/\delta)/\varepsilon^2 F$, Inverse-TS outputs an estimate $\hat{F}$ such that with probability at least $1-\delta$, $|\hat{F} - F| \leq \varepsilon F$.*

Let $\boldsymbol{S}$ denote the k-clique Turán shadow of G and $\text{size}(\boldsymbol{S}) = \sum_{(S,\ell) \in \boldsymbol{S}} |S|$. The running time of Inverse-TS is $O(min(s\alpha^h, \alpha\text{size}(\boldsymbol{S})) + sT_f + m + n)$ and the total storage is $O(\text{size}(\boldsymbol{S}) + m + n)$.

Proof. The X_i are all iid random variables and by the arguments in Lemma 4.4.5, their expectation $\mu = F/\Phi$. Suppose $X_i \in [0, B]$. By Theorem 6.2.1, $Pr[|\sum_{i=1}^{s} X_i - \mu s| \geq \varepsilon s\mu] \leq \delta$ when $s = 3\Phi B \ln(2/\delta)/\varepsilon^2 F$.

The degeneracy of G can be computed in time linear in the size of the graph [173] (Lemma 5.3.1). For any v, the map $M[v]$ in Inverse-TS stores the

$h-1$-clique Prefixed-Turán-Shadow of N_v^+. For any v that gets sampled in Step 5, Inverse-TS checks if the Prefixed-Turán-Shadow of N_v^+ has been constructed and if so, it uses the already-constructed shadow. If not, it constructs it in Step 8 and stores it in M. Thus, in the worst case, it calculates the Prefixed-Turán-Shadow of N_v^+ for every v i.e. it calculates the Prefixed-Turán-Shadow of G which requires time $O(\alpha\text{size}(\boldsymbol{S}))$ according to Theorem 3.5.3. On the other hand, given any v, the size of N_v^+ is atmost α so constructing the $h-1$-clique Prefixed-Turán-Shadow takes time at most $O(\alpha^h)$ (Claim 4.4.2) and it samples s such vertices from D so time required is $O(s\alpha^h)$.

There are s h-vertex sets sampled in Step 11 and checking if the sampled vertices form a clique takes time h^2, while calculating f given that the sampled set is a clique, takes time T_f.

Thus, the total time required by Inverse-TS is $O(min(\alpha\text{size}(\boldsymbol{S}), s\alpha^k) + sT_f + m + n)$.

□

Depending on which structure we are counting, we can find appropriate values for B and T_f. Notice that in the worst case, depending on the structure of the graph, Inverse-TS may end up building the entire shadow in which case it will not provide any savings over PEANUTS. However, practically, we observe that we get significant savings in the amount of shadow built using Inverse-TS in most cases. Unless specified otherwise, all results in this paper are obtained using Inverse-TS.

Algorithm 13: Func-$(k,1)$-Clique(G,K)

$f' = 0$
Let u and v be two distinct vertices from K
Let $nbrs = N_u \cup N_v$
For $nbr \in nbrs$:
    If nbr is connected to all vertices in K except 1 vertex, say w and $nbr > w$, then $f' = f' + 1$
return f'

4.5 Counting cliques and near-k-cliques

4.5.1 Counting $(k,1)$-cliques

Definition 4.5.1. *Let $(u,v), u < v$, be the missing edge in a $(k,1)$-clique J. The lower-order $k-1$-clique in J is the $k-1$-clique $J \setminus \{u\}$, and $J \setminus \{v\}$ is the higher-order $k-1$-clique in J.*

Claim 4.5.2. *Let $f(K)$ for $k-1$-clique K denote the number of $(k,1)$-cliques that K is the lower-order $k-1$-clique in. Then $F = \sum_{K \in C_{k-1}(G)} f(K) =$ total number of $(k,1)$-cliques in G.*

Proof. Every $(k,1)$-clique has exactly 1 lower-order $k-1$-clique. If $f(K)$ denotes the number of $(k,1)$-cliques that K is a part of and is the lower-order clique in, then $\sum_{K \in C_{k-1}(G)} f(K) = F =$ total number of $(k,1)$-cliques in G. □

Claim 4.5.3. *For input $k-1$-clique K, Func-$(k,1)$-Clique returns $f(K)$.*

Proof. For any $nbr \in V$, if $K \cup \{nbr\}$ is a $(k,1)$-clique, then either $nbr \in N_u$ or $nbr \in N_v$ or both. For a given K, Func-$(k,1)$-Clique finds the set of nbr $(nbrs)$ that are connected to every vertex in K except one. Thus, every $\{nbr\} \cup K$ for $nbr \in nbrs$ is a $(k,1)$-clique and it is counted in f' iff K is a lower-order $k-1$-clique. Thus, the value returned, $f' = f(K)$. □

Theorem 4.5.4. *Let d_{max} be the maximum degree of any vertex in G. Then $B = min(2d_{max}, n)$ and $T_f = O(d_{max})$ for Func-$(k,1)$-Clique.*

Proof. By Claim 4.5.2, F =total number of $(k,1)$-cliques in G. For any $(k,1)$-clique $J = K \cup \{nbr\}$ that K is the lower-order $k-1$-clique in, either $nbr \in N_u$ or $nbr \in N_v$ or both. Thus the number of $(k,1)$-cliques in which it is the lower-order $k-1$-clique is atmost $2d_{max}$. On the other hand, there can be atmost n nbr, thus $B = min(2d_{max}, n)$. Finding $nbrs$ takes time $O(d_{max})$ and checking if $nbr \in nbrs$ forms a $(k,1)$-clique with K takes time $O(1)$. Hence, $T_f = O(d_{max})$

□

4.5.2 Counting Type 1, $(k,2)$-cliques

Algorithm 14: Func-$(k,2)$-Clique-Type1(G, K)

```
1 f' = 0
2 For u ∈ K:
3     For v ∈ K, v > u:
4         Let nbrs be the set of vertices connected to all vertices in K except u
    and v
5         f' = f' + |nbrs|
6 return f'
```

Claim 4.5.5. *Let $f(K)$ for $k-1$-clique K denote the number of Type 1 $(k,2)$-cliques that K is contained in. Then $F = \sum_{K' \in C_{k-1}(G)} f(K')$ = the total number of Type 1 $(k,2)$-cliques in G.*

Proof. Every Type 1 $(k,2)$-clique contains exactly 1 $k-1$-clique (Fig. 4.3). Thus, $\sum_{K' \in C_{k-1}(G)} f(K') = F$ = the total number of Type 1 $(k,2)$-cliques in G. □

Claim 4.5.6. *For input $k-1$-clique K, Func-$(k,2)$-Clique-Type1 returns $f(K)$.*

Proof. Given K, for every distinct pair of vertices u and $v \in K, v > u$, Func-$(k,2)$-Clique-Type1 finds the set of vertices $nbrs$ such that $\forall nbr \in nbrs$, nbr is connected to all vertices in K except u and v. Thus, $K \cup \{nbr\}$ is a k-clique with exactly 2 edges missing - (u, nbr) and (v, nbr) with the missing edges having a vertex in common (nbr) i.e. it is a Type 1 $(k,2)$-clique. Thus, Func-$(k,2)$-Clique-Type1 returns the number of Type 1 $(k,2)$-cliques that K is contained in i.e. it returns $f(K)$. □

Theorem 4.5.7. *$B = min(3d_{max}, n)$, $T_f = O(d_{max})$ for Func-$(k,2)$-Clique-Type1.*

Proof. For any 3 vertices $u, v, w \in K$ and for any $(k,2)$-clique $J = K \cup \{nbr\}$ that K is contained in, atleast one of $(u, nbr), (v, nbr), (w, nbr) \in E(G)$. Thus, any K can be a part of atmost $min(3d_{max}, n)$ Type 1 $(k,2)$-cliques. For every pair (u, v) in K, Func-$(k,2)$-Clique-Type1 calculates the number of vertices connected to all in K but u and v which takes time $O(d_{max})$. Thus, $T_f = O(d_{max})$. □

4.5.3 Counting Type 2 $(k,2)$-cliques

Definition 4.5.8. *Given a Type 2 $(k,2)$-clique J, $v, x \in J$, the set $K = J \setminus \{v, x\}$ is the lowest order $k-2$-clique of J if it fulfills all the following conditions:*

1. *$(u, v) \notin E(G)$, $(w, x) \notin E(G)$ (note that this implies that K is a $k-2$-clique).*

2. *$degen(u) < degen(v)$*

3. *$degen(u) < degen(w) < degen(x)$.*

Note that u, v, w and x are all distinct and J consists of exactly 4, $k-2$-cliques: $J \setminus \{v, x\}$, $J \setminus \{v, w\}$, $J \setminus \{u, x\}$ and $J \setminus \{u, w\}$ (Fig. 4.3), and the lowest order

Algorithm 15: Func-$(k,2)$-Clique-Type2(G,K)

1 $f'=0$
2 Let $degen(u)$ denote the position of u in the degeneracy order of G.
3 For $u \in K$:
4 For $w \in K, degen(w) > degen(u)$:
5 Let $nbrsu = N_u^+$ be the set of out-nbrs of u such that they are connected to all vertices in K except w and $\forall nbru \in nbrsu, degen(w) < degen(nbru)$.
6 Let $nbrsw$ be the set of neighbors of w in G such that they are connected to all vertices in K except u
7 For $x \in nbrsu$:
8 For $v \in nbrsw$:
9 If $(nbru, nbrw) \in E(G) : f' = f' + 1$
10 return f'

$k-2$-clique of J is the one which has the vertex (u) with minimum position in the degeneracy ordering of G and the minimum neighbor of u.

Claim 4.5.9. *Let $f(K)$ for $k-2$-clique K denote the number of Type 2 $(k,2)$-cliques that K is the lowest-order $k-2$-clique in. Then $F = \sum_{K' \in C_{k-2}(G)} f(K')$ = total number of Type 2 $(k,2)$-cliques in G.*

Proof. Every Type 2 $(k,2)$-clique has exactly one lowest order $k-2$-clique in it. If $f(K)$ denotes the number of Type 2 $(k,2)$-cliques that K is the lowest-order $k-2$-clique in, then $\sum_{K' \in C_{k-2}(G)} f(K')$ =total number of Type 2 $(k,2)$-cliques in G. □

Claim 4.5.10. *For input $k-2$-clique K, Func-$(k,2)$-Clique-Type2 returns $f(K)$.*

Proof. Given a $k-2$-clique K, Step 3 and Step 4 loop over all possible candidates for u and w, maintaining the condition that $degen(u) < degen(w)$. In Step 5, Func-$(k,2)$-Clique-Type2 picks the outneighbors of u that are potential candidates for x $(nbrsu)$ such that $(w,x) \notin E(G)$ and $degen(w) < degen(x)$. In Step 6, it picks potential candidates for v $(nbrsw)$ i.e. neighbors of w that are connected to

all vertices in K except u. Finally, in Step 9, it checks if v and x are connected. Thus, f' in Step 9 is incremented iff all the conditions of a lowest order $k-2$-clique of a Type 2 $(k,2)$-clique are fulfilled. Thus, the returned value $f' = f(K)$. □

Theorem 4.5.11. *$B = min(n^2, k^2\alpha d_{max}/2)$, $T_f = O(\alpha + d_{max})$ for Func-$(k,2)$-Clique-Type2.*

Proof. Given K, there can be atmost $k^2/2$ candidates for (u,w). There can be at most α candidates for x (since it has to be an outneighbor of u) and atmost d_{max} candidates for v (neighbors of w). On the other hand, there can be atmost n candidates for x and v each. Thus, $B = min(n^2, k^2\alpha d_{max}/2)$

Given a set of $k-2$ vertices, it takes $O(k^2)$ time to check if it forms a clique. There are $O(k^2)$ candidates for (u,w) each. There are atmost α candidates for x and d_{max} candidates for v whose connections to each of the $k-2$ vertices need to be checked. This takes time $O(\alpha + d_{max})$. Altogether, $T_f = O(\alpha + d_{max})$. □

4.6 Experimental Results

Preliminaries: All code for our experiments is available here: https://bitbucket.org/sjain12/peanuts/. We implemented our algorithms in `C++` and ran our experiments on a commodity machine equipped with a 1.4GHz AMD Opteron(TM) processor 6272 with 8 cores and 2048KB L2 cache (per core), 6144KB L3 cache, and 128GB memory. We performed our experiments on a collection of graphs from SNAP [159], including social networks, web networks, and infrastructure networks. The largest graph has more than 100M edges. Basic properties like degeneracy, maximum degree etc. of these graphs are presented in Table 6.1. We consider the graph to be simple and undirected.

Our practical implementation differs slightly from Inverse-TS in two ways: we

					k=5			k=7			k=10			
graph	**vertices**	**edges**	**degen**	d_{max}	**estimate**	**% error**	**time**	**estimate**	**% error**	**time**	**estimate**	**% error**	**time**	**type**
web-Stanford	2.82E+05	1.99E+06	71	38625	2.36E+10	0.85	142	8.99E+11	-	216	2.16E+14	-	129	$(k,1)$
					1.15E+11	0.46	8283	7.33E+11	-	3802	1.12E+14	-	1087	$(k,2)$ Type 1
					1.12E+10	1.19	5396	2.51E+11	-	538	1.04E+14	-	293	$(k,2)$ Type 2
					6.21E+8			3.47E+10			5.82E+12			k
web-Google	8.76E+05	4.32E+06	44	6332	6.76E+08	0.44	13	2.19E+09	0.45	12	2.41E+10	0.41	10	$(k,1)$
					2.08E+09	0.48	276	4.45E+09	0.01	172	2.05E+10	-	42	$(k,2)$ Type 1
					7.18E+07	1.10	21	2.93E+08	0.01	18	7.70E+09	0.01	13	$(k,2)$ Type 2
					1.05E+08			6.06E+08			1.29E+10			k
amazon0601	4.03E+05	4.89E+06	10	2752	1.17E+07	0.00	4	2.88E+06	0.01	3	3.76E+04	0.02	1.5	$(k,1)$
					5.38E+07	0.01	10	7.84E+06	0.01	7	8.70E+04	0.01	3	$(k,2)$ Type 1
					3.16E+06	0.01	4	1.30E+06	0.01	5	2.96E+04	0.00	3	$(k,2)$ Type 2
					3.64E+06			9.98E+05			9.77E+03			k
web-BerkStan	6.85E+05	6.65E+06	201	84230	4.89E+11	0.93	397	2.89E+13	-	470	1.85E+16	-	704	$(k,1)$
					1.89E+12	0.32	20534	7.39E+13	-	6080	1.43E+16	-	5383	$(k,2)$ Type 1
					6.61E+10	0.09	12400	7.32E+11	-	605	1.65E+14	-	646	$(k,2)$ Type 2
					2.19E+10			9.30E+12			5.79E+16			k
as-skitter	1.70E+06	1.11E+07	111	35455	3.94E+10	4.52	1180	5.44E+11	-	1034	7.91E+13	-	800	$(k,1)$
					2.97E+11	1.63	31724	2.48E+12	-	16220	2.27E+13	-	10461	$(k,2)$ Type 1
					2.34E+10	1.37	4132	3.97E+11	-	2598	8.55E+13	-	1038	$(k,2)$ Type 2
					1.17E+09			7.30E+10			1.43E+13			k
cit-Patents	3.77E+06	1.65E+07	64	793	4.12E+07	0.01	10	7.20E+07	0.01	6	9.06E+05*	42.22	4	$(k,1)$
					1.11E+08	1.83	17	1.31E+08	2.29	8	1.43E+06*	49.11	5	$(k,2)$ Type 1
					1.31E+08	0.01	6	6.76E+08	3.36	9	2.54E+07*	31.35	5	$(k,2)$ Type 2
					3.05E+06			1.89E+06			2.55E+03			k
soc-pokec	1.63E+06	2.23E+07	47	14854	4.22E+08*	8.48	218	5.41E+07*	9.96	81	7.67E+08	4.24	55	$(k,1)$
					2.40E+09*	6.19	218	1.59E+09*	4.6	136	1.67E+09	0.02	68	$(k,2)$ Type 1
					3.34E+08	0.00	38	6.78E+08*	7.61	95	1.28E+09	0.01	64	$(k,2)$ Type 2
					5.29E+07			8.43E+07			1.98E+08			k
com-lj	4.00E+06	3.47E+07	360	14815	2.85E+11	0.11	200	4.28E+14	-	452	1.18E+19	-	558	$(k,1)$
					4.63E+11	0.34	756	5.11E+14	-	613	1.22E+19	-	680	$(k,2)$ Type 1
					5.39E+10	0.53	269	1.24E+14	-	581	4.23E+18	-	568	$(k,2)$ Type 2
					2.47E+11			4.51E+14			1.47E+19			k
soc-LJ	4.84E+06	8.57E+07	372	20333	6.32E+11	0.03	677	1.01E+15	-	779	4.14E+19	-	960	$(k,1)$
					1.03E+12	0.17	1504	1.27E+15	-	1107	4.57E+19	-	1320	$(k,2)$ Type 1
					1.34E+11	0.41	506	2.77E+14	-	1007	1.17E+19	-	1111	$(k,2)$ Type 2
								4.49E+14						k
com-orkut	3.07E+06	1.17E+08	253	33313	1.56E+11	-	9507	2.26E+12	-	16546	4.66E+13	-	26370	$(k,1)$
					1.46E+12	-	21213	7.82E+12	-	24148	1.04E+14	-	29881	$(k,2)$ Type 1
					2.37E+11	-	3879	3.51E+12	-	11617	1.60E+14	-	22676	$(k,2)$ Type 2
					1.57E+10			3.61E+11			3.03E+13			k

Table 4.1: Table shows the sizes, degeneracy, maximum degree of the graphs, the counts of 5, 7 and 10 cliques and near-cliques obtained using Inverse-TS, the percent relative error in the estimates (for those graphs for which we were able to get exact numbers within 24 hours), and time in seconds required to get the estimates. The rows whose types are k in the rightmost column show the number of k-cliques.For most instances, the algorithm terminated in minutes. Values marked with * have significant errors which are addressed in Tab. 4.2

fix the number of samples to 500K. Moreover, since the number of samples are fixed, we can sample from D in Inverse-TS all at once and maintain counts of the number of cliques to be sampled from each outneighborhood. We can then explore the outneighborhoods in an online fashion, sampling as we build the shadow. Once the samples from a vertex's outneighborhood have been ontained, we no longer need the shadow of the outneighborhood and the shadow can be discarded. Thus, we don't need to store the entire shadow but only the shadow of the current vertex's outneighborhood.

We focus on counting near-k-cliques for k ranging from 5 to 10.

Accuracy and convergence of Inverse-TS: We picked some graphs for

which the exact near-clique counts are known (for all $k \in [5, 10]$). For each graph and near-clique type, for sample size in [10K,50K,100K,500K,1M], we performed 100 runs of the algorithm. We show here results for `amazon0601` for $k = 7$, though similar results were observed for other graphs and k. We plot the spread of the output of Inverse-TS, over all these runs. The results are shown in Fig. 4.4. The red line denotes the true answer, and there is a point for the output of every single run. As we can see, the output of Inverse-TS fast converges to the true value as we increase the number of samples. For 500K samples, the range of values is within 5% of the true answer which is much less compared to the spread of cc. Similar results were observed for other graphs for which the exact counts were available, except soc-pokec. The error was mostly $< 5\%$ and often $< 1\%$ as can be seen from Tab. 6.1.

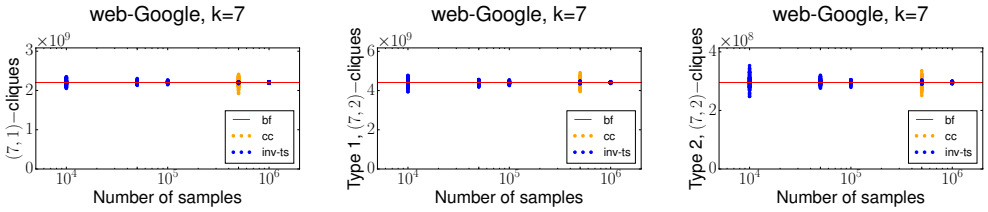
In cases like soc-pokec the error can be high. This happens when most of the samples end up empty, either because the sampled vertices did not form a clique, or the samples belonged to out-neighborhoods that did not have a clique of the required size or the sampled clique does not participate in any near-cliques. This can be detected by observing how many of the samples taken in Step 11 were cliques with non-zero f. If this number is $<<$ 5000, the estimates are likely to have substantial error. This can be remedied by either taking more samples or using PEANUTS. Tab. 4.2 shows the revised estimates obtained using PEANUTS using 500K samples, for values in Tab. 6.1 that have substantial error (marked with an asterisk).

For the graphs for which we could not get exact numbers (since the bf algorithm did not terminate in 1 day), we were unable to obtain error percentages. However, even for such graphs we saw good convergence over 100 runs of the algorithm.

Running time: The runtimes for near-cliques of size 7 are presented in

graph	k	revised estimate	revised % error	type
cit-Patents	10	648944	1.9	$(k,1)$
		2.84E+06	1.32	$(k,2)$ Type 1
		3.69+07	0.3	$(k,2)$ Type 2
soc-pokec	5	3.91E+08	0.4	$(k,1)$
		2.27E+09	0.1	$(k,2)$ Type 1
soc-pokec	7	4.92E+08	0.0	$(k,1)$
		1.53E+09	0.2	$(k,2)$ Type 1
		6.27E+08	0.6	$(k,2)$ Type 2

Table 4.2: Table revised estimates and revised error for the counts of near-cliques obtained using PEANUTS with 500K samples for the erroneous estimates in Tab. 6.1 (marked with *).



(a) $(k,1)$-clique **(b)** Type 1, $(k,2)$-clique **(c)** Type 2, $(k,2)$-clique

Figure 4.4: Fig. 4.4a, Fig. 4.4b, Fig. 4.4c show convergence over 100 runs of Inverse-TS using number of samples in [10K, 50K, 100K, 500K,1M] for all near-clique types. The red line indicates the true value.

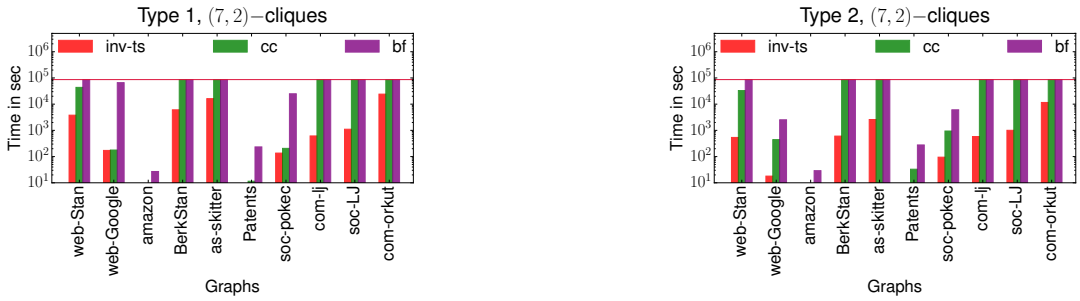


Figure 4.5: Figure shows the time required by Inverse-TS (inv-ts), color-coding (cc) and brute force (bf) to estimate the number of Type 1 and Type 2 $(k, 2)$-cliques resp. in 10 real world graphs for $k = 7$. The red line indicates 86400 seconds (24 hours).

Tab. 6.1. We show the time for a single run in each case. In all cases except com-orkut, the algorithm terminated in minutes (for com-orkut, it took less than a day) where cc and bf did not terminate in an entire day (and in some cases, even after 5 days).

Comparison with other algorithms: Our exact brute-force procedure is a well-tuned algorithm that uses the degeneracy ordering and exhaustively searches outneighborhoods for cliques (based on the approach by Chiba-Nishizeki [61]). Once a clique is found, we count all the near-cliques the clique is a part of and sum this quantity over all cliques.

On average, color-coding took time anywhere between 2x to 100x time taken by Inverse-TS, while giving poorer accuracy. Brute force took even more time. Inverse-TS has reduced the time required to obtain these estimates from days to minutes.

4.6.1 Near-cliques in practice

One of the important applications of near-cliques is in finding missing edges that likely should have been present in the graph in the first place. We deployed our algorithm on a citation network [235]. Using Inverse-TS we were able to obtain several sets of papers in which, ever pair of paper either cited or was cited by the other paper (depending on the chronological order of the papers), except 1 or 2 pairs. For example, a $(7, 1)$-clique we obtained comprised of the papers with the following titles:

1. A ray tracing solution for diffuse interreflection
2. Distributed ray tracing
3. A global illumination solution for general reflectance distributions
4. Adaptive radiosity textures for bidirectional ray tracing
5. The rendering equation
6. A two-pass solution to the rendering equation: A synthesis of ray tracing and radiosity methods
7. A framework for realistic image synthesis

in which, only (1) and (3) were not connected. Thus, by mining near-cliques one can discover missing links and offer suggestions for which items should be related. In applications where the data is known to be noisy, it would be interesting to see how the properties of the graph change upon adding these (possibly) missing links and obtaining a more complete picture.

Listing near-cliques: In some applications of near-cliques, a u.a.r. sample of near-cliques may be required. Suppose we want to provide a u.a.r. sample of

Type 1 $(k, 2)$-cliques for a given k. PEANUTS allows us to sample cliques u.a.r. Once a clique K is sampled, suppose we return a u.a.r. Type 1 $(k, 2)$-clique that K participates in. Let J be a Type 1 $(k, 2)$-clique that K participates in, then the probability of J being returned is inversely proportional to $f(K)$. In other words, this approach does not give us a u.a.r. sample of Type 1 $(k, 2)$-cliques. However, if we list all the Type 1 $(k, 2)$-cliques that K participates in, and repeat this process for several different K, even though the samples in the list may be correlated, every Type 1 $(k, 2)$-cliques in G has equal probability of being put in the list. In applications where some amount of correlation in samples is tolerable, such a list can be useful.

4.7 Missteps and practical insights

There were several insights we gained during the implementation of this project which are practically useful.

Our first approach to tackling the problem of estimating the number of $(k, 1)$-cliques was as follows: for every missing edge, add that edge to the graph and count the number of k-cliques containing that missing edge. In other words, given a vertex v, we look at all vertices u such that $degen(u) > degen(v)$, $(u, v) \notin E(G)$ and u and v have atleast $k - 2$ common neighbors. Every $k - 2$-clique in this common neighborhood corresponds to a unique $(k, 1)$-clique in G. Essentially, there is branch spawned for every wedge that is not a part of a triangle when u and v have atleast $k - 2$ common neighbors. However, practical experiments showed that the resultant Turán shadow was much larger compared to the shadow for Turán $k - 1$-clique. In other words, rather than keeping the vertices in dense regions together, the dense regions were getting split, resulting in many copies of the same vertices.

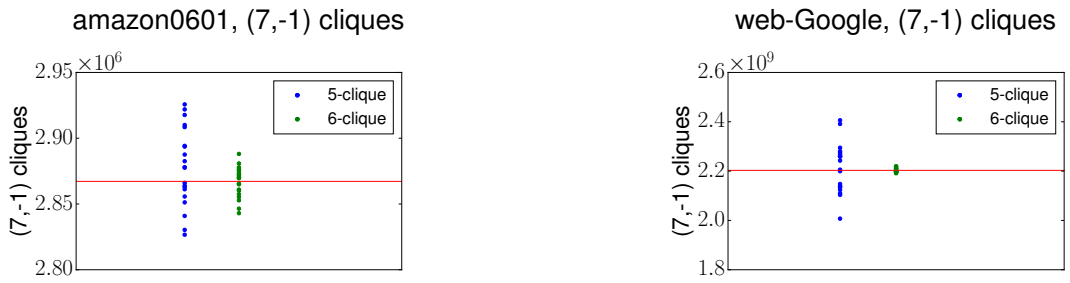


Figure 4.6: Figure shows the estimates obtained from 20 runs each of 2 counters of near-cliques for $k = 7$, one where we sample a 5 clique and the one where we sample a 6-clique, as in the case of Func-$(k, 1)$-Clique. Red line shows the actual count of $(7, -1)-$cliques.

Another important insight we gained was from a different approach for counting $(k, 1)$-clique: every $(k, 1)$-clique J has a unique $k-2$-clique $K = J \setminus \{u, v\}$ such that $(u, v) \notin E(G)$. Let us call such a $k-2$-clique the anchor of J then every $(k, 1)$-clique has a unique anchor. Given a $k-2$-clique K, let $f(K) =$ number of $(k, 1)$-cliques K is the anchor for, then total number of $(k, 1)$-cliques $= F$. However, it turned out that the variance in this case in f was much larger. Fig. 4.6 shows the spread from 20 runs of PIVOTER for $k = 7$ using Func-$(k, 1)$-Clique (6-clique) and the 5-clique approach. One way to explain this behavior is that in the 5-clique approach, F is spread over $C_5(G)$ i.e. the set of 5-cliques, while in 6-clique approach (as used in Func-$(k, 1)$-Clique), F is spread over $C_6(G)$. If we look at the trend in clique counts of graphs in Fig. 5 of [135], barring a couple of small graphs, in most graphs, for $5 \leq k \leq 10$, number of k-cliques shows an increasing trend, which means that F is spread over more objects (cliques) in the 6-clique than in the 5-clique case which could be an explanation for the higher variance in 5-clique case. Fig. 4.7 shows the distribution of f and as we can see, in the case of 5-clique, the range of f (orange line) is indeed very large, compared to the range of f in the 6-clique case.

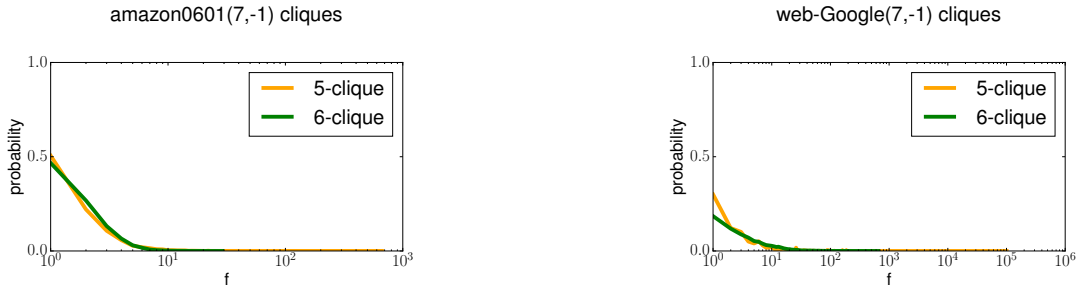


Figure 4.7: Figure shows the distribuion of f obtained using 2 different counters of near-cliques for $k = 7$, one where we sample a 5 clique and one where we sample a 6-clique. x-axis is on log scale. As expected, the variance and max value of f is much larger when sampling 5-cliques than when sampling 6-cliques.

4.8 Future Work

We leverage the fast clique counting algorithm TuránShadow to count near-cliques that are essentially k-cliques missing 1 or 2 edges, for k upto 10. The proposed algorithm gives significant savings in space and time compared to state of the art.

One could generalize the definition of near-cliques to larger values of r and define a $(k, r)-$clique as a $k-$ clique that is missing exactly r edges. It would be interesting to see how far r can be increased such that near-clique counting would still be feasible using this clique-centered approach.

Chapter 5

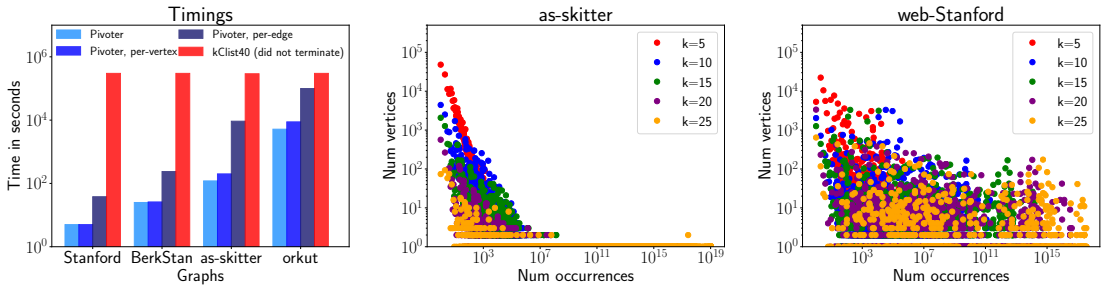
Pivoter

5.1 Introduction

We revisit the problem of clique counting. Despite much effort on this problem, it has been challenging to get scalable algorithms for clique counting. For large graphs, some recent practical algorithms have succeeded in counting up to (around) 10-cliques [79, 105, 134]. They either use randomized approximation or parallelism to speed up their counting. But they do not scale for larger k and it is difficult to obtain more refined clique counts (such as counts for every vertex or every edge).

5.1.1 Problem Statement

We are given an undirected, simple graph $G(V, E)$. For $k \geq 3$, a k-clique is a set of k vertices that induce a complete subgraph (it contains all edges among the k vertices). We will denote the number of k-cliques as C_k. For a vertex $v \in V$, we use $c_k(v)$ to denote the number of k-cliques that v participates in. Analogously, we define $c_k(e)$ for edge $e \in E$.



(a) Timings **(b)** Frequency distribution **(c)** Frequency distribution

Figure 5.1: Fig. 5.1a shows the comparison of time taken (in seconds) by PIVOTER for 4 of our largest graphs to count *all* $k-$cliques with the time taken by kClist40 (the parallel version of the state of the art algorithm kClist that uses 40 threads) to count the number of $k-$cliques, where k is the maximum clique size in each graph. For Stanford, BerkStan, as-skitter, orkut, the maximum clique sizes were 61, 201, 67 and 51 resp. PIVOTER terminated for most graphs in minutes, (except for orkut, for which it took about 2 hours) whereas kClist40 had not terminated even after 3 days, giving a speedup of 100x to 10000x. Fig. 5.1a also shows the time taken by PIVOTER to obtain the per-vertex and per-edge $k-$clique counts. They were within a factor of the time taken to obtain global $k-$clique counts. Fig. 5.1b and Fig. 5.1c shows the frequency distribution of k-cliques i.e. for every number r on the x-axis, the y-axis shows the number of vertices that participate in r k-cliques, for $k \in [5, 10, 15, 20, 25]$ for as-skitter and web-Stanford graphs.

We focus on the following problems, in increasing order of difficulty. We stress that k is *not* part of the input, and we want results for all values of k.

- Global clique counts: Output, $\forall k \geq 3$, C_k.
- Per-vertex clique counts: Output, $\forall k$, $\forall v \in V$, the value $c_k(v)$.
- Per-edge clique counts: Output, $\forall k$, $\forall e \in E$, the value $c_k(e)$.

The per-vertex and per-edge counts are sometimes called *local counts*. In clustering applications, the local counts are used as vertex or edge weights, and are therefore even more useful than global counts [31, 166, 208, 244, 246, 264].

Challenges: Even the simplest problem of getting global clique counts subsumes a number of recent results on clique counting [79, 105, 134]. The main challenge is combinatorial explosion: for example, the `web-Stanford` web

graph with 2M edges has *3000 trillion* 15-cliques. These numbers are even more astronomical for larger graphs. Any method that tries to enumerate is doomed to failure.

Amazingly, recent work by Danisch-Balalau-Sozio uses parallel algorithms to count beyond trillions of cliques. But even their algorithm fails to get all global clique counts for a number of datasets. Randomized methods have been used with some success, but even they cannot estimate all clique counts [105, 134].

Local counting, for all k, is even harder, especially given the sheer size of the output. Parallel methods would eventually need to store local counts for every subproblem, which would increase the overall memory footprint. For local counts, sampling would require far too many random variables, each of which need to be sampled many times for convergence. (We give more explanation in §6.1.5.)

This raises the main question:

Is there a scalable, exact algorithm for getting all global and local cliques counts, on real-world graphs with millions of edges?

To the best of our knowedge, there is no previous algorithm that can solve these problems on even moderate-sized graphs with a few million edges.

5.1.2 Main contributions

Our main contribution is a new practical algorithm Pivoter for the global and local clique counting problems.

Exact counting without enumeration: Current methods for exact clique counting perform an *enumeration*, in that the algorithm explicitly "visits" every clique. Thus, this method cannot scale to counting larger cliques, since the number of cliques is simply too large. Our main insight is that the method of *pivoting*, used to reduce recursion trees for maximal clique enumeration [48, 98], can be

applied to counting cliques of all sizes.

Succinct Clique Trees through Pivoting: We prove that pivoting can be used to construct a special data structure called the *Succinct Clique Tree* (SCT). The SCT stores a unique representation of all cliques, but is much smaller than the total number of cliques. It can also be built quite efficiently. Additionally, given the tree, one can easily "read off" the number of k-cliques and various local counts in the graph. Remarkably, we can get all counts without storing the entire tree and the storage required at any point is linear in the number of edges. The SCT can also be used to obtain local counts for bigger structures. For eg. how many 7-cliques is a given 3-clique a part of. Applications that require an enumeration of such structures require a lot of memory to store all the structures. SCT provides a compact way of representing them such that they can be efficiently read from the tree.

Excellent practical performance: We implement PIVOTER on a commodity machine. For global clique counting, PIVOTER is able to process graphs of up to tens of millions of edges in *minutes*. Previous results either work only for small values of k (typically up to 10) or take much longer. Consider Fig. 5.1a, where the time of PIVOTER is compared with that of kClist (the state of the art parallel algorithm for clique counting) [79]. In the instances shown kClist did not terminate even after running for 3 days. By contrast, for the largest `com-orkut` social network with more than 100M edges, PIVOTER gets all values of C_k within two hours. (Typically, in this time, kClist gets $k-$clique counts only up to $k \leq 13$.)

Feasible computation of local counts: PIVOTER is quite efficient for per-vertex counts, and runs in at most twice the time for global counts. The times for local clique counting are given in Fig. 5.1a. Even for the extremely

challenging problem of per-edge counts, in most instances PIVOTER gets these numbers in a few hours. (For the `com-orkut` social network though, it takes a few days.)

This allows us to get data shown in Fig. 5.1b and Fig. 5.1c, that plots the frequency distribution of k-cliques. (In other words, for every number r, we plot the number of vertices that participate in r k-cliques.) As mentioned earlier, this information is used for dense subgraph discovery [208, 244]. To the best of our knowledge, this is the first algorithm that is able to get such information for real-world graphs.

5.1.3 Related Work

Subgraph counting has an immensely rich history in network science, ranging from applications across social network analysis, bioinformatics, recommendation systems, graph clustering (we refer the reader to the tutorial [217] and references within). We only describe work directly relevant to clique counting.

The simplest case of clique counting is *triangle counting*, which has received much attention from the data mining and algorithms communities. Recent work has shown the relevance of counts of large subgraphs (4, 5 vertex patterns) [35, 206, 224, 249, 263]. Local clique counts have played a significant role in a flurry of work on faster and better algorithms for dense subgraph discovery and community detection [31, 208, 244, 246]. The latter results define the "motif conductance", where cuts are measured by the number of subgraphs (not just edges) cut. This has been related to higher order clustering coefficients [263, 264]. These quantities are computed using local clique counts, underscoring the importance of these numbers.

The problem of counting cliques (and variants such as counting maximal

cliques) has received much attention both from the applied and theoretical computer science communities [20, 56, 61, 250]. Classic techniques like color-coding [37, 271] and path sampling [137, 216, 254] have been employed for counting cliques up to size 5.

For larger cliques, Finocchi-Finocchi-Fusco gave a MapReduce algorithm that uses orientation and sampling techniques [105]. 3 uses methods from extremal combinatorics to give a fast sampling algorithm [134], that is arguably the fastest approximate clique counter to date. In a remarkable result, Danisch-Balalau-Sozio gave a parallel implementation (kClist) of a classic algorithm of Chiba-Nishizeki, which is able to enumerate upto trillions of cliques [79]. For exact counting, we consider kClist as the state of the art. Despite the collection of clever techniques, none of these methods really scale beyond counting (say) 10-cliques for large graphs.

Why local counting is hard: Note that either parallelism or sampling is used to tame the combinatorial explosion. Even though (at least for small k), one can enumerate all cliques in parallel, local counting requires updating a potentially global data structure, the list of all $c_k(v)$ or $c_k(e)$ values. To get the benefits of parallelism, one would either have to duplicate a large data structure or combine results for various threads to get all local counts. While this may be feasible, it adds an extra memory overhead.

Sampling methods typically require some overhead for convergence. For local counts, there are simply too many samples required to get accurate values for (say) all $c_k(v)$ values. For these reasons, we strongly believe that new ideas were required to get efficient local counting.

Maximal clique enumeration: Extremely relevant to our approach is a line of work of maximal clique enumeration. A *maximal clique* is one that is

not contained in a larger clique. Unlike the combinatorial explosion of k-cliques, maximal cliques tend to be much fewer. The first algorithm for this problem is the classic Bron-Kerbosch backtracking procedure from the 70s [16, 48]. They also introduced an idea called *pivoting*, that prunes the recursion tree for efficiency. Tomita-Tanaka-Takahashi gave the first theoretical analysis of pivoting rules, and showed asymptotic improvements [238]. Eppstein-Löeffler- Strash combined these ideas with orientation methods to give a practical and provably fast algorithm for maximal clique enumeration [97, 98]. An important empirical observation of this line of work is that the underlying recursion tree created with pivoting is typically small for real-world graphs. This is the starting point for our work.

5.2 Main Ideas

Inspired by the success of maximal clique enumeration through pivoting, we design the Succinct Clique Tree (SCT) of a graph for clique counting.

To explain the SCT, it is useful to begin with the simple backtracking algorithm for listing all cliques. For any vertex v, let $N(v)$ denote the neighborhood of v. Any clique containing v is formed by adding v to a clique contained in $N(v)$. Thus, we can find all cliques by this simple recursive procedure: for all v, recursively enumerate all cliques in $N(v)$. For each such clique, add v to get a new clique. It is convenient to think of the recursion tree of this algorithm. Every node of the tree (corresponding to a recursive call) corresponds to a subset $S \subseteq V$, and the subtree of calls enumerates all cliques contained in S. A call to S makes a recursive call corresponding to every $s \in S$, which is over the set $N(s) \cap S$ (the neighbors of v in S). We can label every edge of the tree (call them *links* to distinguish from edges of G) with a vertex, whose neighborhood leads to the next recursive call. It is not hard to see that the link labels, along any path from a

root (that might not end at a leaf), give a clique. Moreover, every clique has such a representation.

Indeed, every permutation of clique forms such a path. A simple and classic method to eliminate multiple productions of a clique is *acyclic orientations*. Simply orient the graph as a DAG, and only make recursive calls on out-neighborhoods. Typically, an orientation is chosen by degeneracy/core decomposition or degree orderings, so that out-neighborhood sizes are minimized. This is a central technique in all recent applied algorithms on clique counting [79, 105, 134]. Yet it is not feasible to construct the recursion tree to completion, and it is typically truncated at some depth (≤ 10) for large graphs.

Is it possible to somehow "compress" the tree, and get a unique (easily accessible) representation of all cliques?

The power of pivoting: We discover a suprising answer, in pivoting. This was discovered by Bron-Kerbosch in the context of *maximal* cliques [48]. We describe, at an intuitive level, how it can be applied for global and local clique counting. For the recursive call at S, first pick a *pivot* vertex $p \in S$. Observe that the cliques in S can be partitioned into three classes as follows. For clique C contained in S: (i) $p \in C$, (ii) $C \subset N(p)$, (iii) C contains a non-neighbor of p. There is 1-1 correspondence between cliques of type (i) and (ii), so we could hope to only enumerate type (ii) cliques.

Thus, from a recursive call for S, we make recursive calls to find cliques in $N(p) \cap S$, and $N(u) \cap S$ for every *non-neighbor* u of p in S. We avoid making recursive calls corresponding to vertices in $N(p)$. This gives the main savings over the simple backtracking procedure. The natural choice of p is the highest degree vertex in the graph induced on S. The recursion tree obtained is essentially the SCT. We stress that this is quite different from the Bron-Kerbosch recursion tree.

The BK algorithm also maintains a set of excluded vertices since it only cares for maximal cliques. This excluded set is used to prune away branches that cannot be maximal; moreover, the pivots in BK are potentially chosen from outside S to increase pruning. The SCT is constructed in this specific manner to ensure unique clique representations, which the BK tree does not provide.

The SCT is significantly smaller than recursion trees that use degeneracy orientations (which one cannot feasibly construct). In practice, it can be constructed efficiently for graphs with tens of millions of edges. As before the nodes of the SCT are labeled with subsets (corresponding to the recursive calls), and links are labeled with vertices (corresponding to the vertex whose neighborhood is being processed). Abusing notation, in the following discussion, we refer to a path by the set of link labels in the path.

How can we count all cliques using the SCT? Every root to leaf path in the tree corresponds to a clique, but not all cliques correspond to paths. This is distinct from the standard recursion tree discussed earlier, where every clique corresponds to a path from the root. Indeed, this is why the standard recursion trees (even with degeneracy orientations) are large.

We prove the following remarkable "unique encoding" property. Within any root to leaf path T, there is a subset of links P corresponding to the pivot calls. Every clique C in the graph can be *uniquely* expressed as $(T \setminus P) \cup Q$ for some $Q \subseteq P$ (for a specific path T). The uniqueness is critical for global and local counting, since we can simply write down formulas to extract all counts. Thus, the SCT gives a unique encoding for every clique in the graph.

Intuitively, the source of compression can be seen in two different ways. The simplest way is to see that pivoting prunes the tree, because recursive calls are only made for a subset of vertices. But also, not every clique is represented by

(the link labels of) a path from the root. Thus, there are far fewer paths in the SCT. The final algorithm is quite simple and the main work was coming up with the above insight. Despite this simplicity, it outperforms even parallel methods for exact clique counting by orders of magnitude.

Our main theorem follows. Basically, clique counts can be obtained in time proportional to the size of the SCT. All the technical terms will be formally defined in §5.3.1.

Theorem 5.2.1. *Let G be an input graph with n vertices, m edges, and degeneracy α. Let $SCT(G)$ be the Succinct Clique Tree of graph G.*

The procedure PIVOTER(G) *correctly outputs all global and local counts. For global and per-vertex counts, the running time is $O(\alpha^2|\,SCT(G)| + m + n)$. For per-edge counts, the running time is $O(\alpha^3|\,SCT(G)| + m + n)$. The storage cost is $O(m + n)$.*

Empirically, we observe that the SCT is quite small. In the worst-case, $|\,SCT(G)| = O(n3^{\alpha/3})$, which follows from arguments by Eppstein-Löeffler-Strash [98] and Tomita-Tanaka-Takahashi [238] (an exponential dependence is necessary because of the NP-hardness of maximum clique). We give a detailed description in §5.5

5.3 Main Algorithm

5.3.1 Preliminaries

We start with the mathematical formalism required to describe the main algorithm and associated proofs. The input is a simple, undirected graph $G = (V, E)$, where $|V| = n$ and $|E| = m$. It is convenient to assume that G

is connected. We use vertices to denote the elements of V (the term *nodes* will be used for a different construct). We use the following notation for neighborhoods.

- $N(v)$: This is the neighborhood of v.
- $N(S, v)$: For any subset of vertices S, we use $N(S, v)$ to denote $N(v) \cap S$. Alternately, this is the neighborhood of v in S.

We will use *degeneracy orderings* (or core decompositions) to reduce the recursion tree. This is a standard technique for clique counting [61, 79, 105, 134]. This ordering is obtained by iteratively removing the minimum degree vertex, and can be computed in linear time [173]. Typically, one uses this ordering to convert G into a DAG. The largest *out-degree* is the graph degeneracy, denoted α. We state this fact as a lemma, which is considered a classic fact in graph theory and network science.

Lemma 5.3.1. *[173] Given a graph $G = (V, E)$, there is a linear time algorithm that constructs an ayclic orientation of G such that all outdegrees are at most α.*

The most important construct we design is the *Succinct Clique Tree* (SCT) $\boldsymbol{T}$. The SCT stores special node and link attributes that are key to getting global and local clique counts, for all values of k. The construction and properties of the SCT are given in the next section. Here, we list out technical notation associated with the SCT $\boldsymbol{T}$.

Formally, $\boldsymbol{T}$ is a tree where nodes are labeled with subsets of V, with the following properties.

- The root is labeled V.
- Parent labels are strict supersets of child labels.
- Leaves are labeled with the empty set $\emptyset$.

An important aspect of $\boldsymbol{T}$ are *link labels*. A link label is a pair with a vertex of V and a "call type". The label is of the form $(v, \mathfrak{p})$ or $(v, \mathfrak{h})$, where $\mathfrak{p}$ is shorthand

for “pivot" and $\mathfrak{h}$ for “hold". For a link label $(v, \cdot)$ of the link (S, S') (where $S \supset S'$ is the parent), v will be an element of S.

Consider a root to leaf path T of $\boldsymbol{T}$. We have the following associated set of vertices. It is convenient to think of T as a set of tree links.

- $H(T)$: This is the set of vertices associated with “hold" call types, among the links of T. Formally, $H(T)$ is $\{v|(v, \mathfrak{h})$ is label of link in $T\}$.
- $P(T)$: This is the set of vertices with “pivot" calls. Formally $P(T)$ is $\{v|(v, \mathfrak{p})$ is label of link in $T\}$.

We now describe our algorithm. We stress that the presentation here is different from the implementation. The following presentation is easier for mathematical formalization and proving correctness. The implementation is a recursive version of the same algorithm, which is more space efficient. This is explained in the proof of Theorem 6.3.1.

5.4 Building the SCT

We give the algorithm to construct the SCT. We keep track of various attributes to appropriately label the edges. The algorithm will construct the SCT $\boldsymbol{T}$ in a breadth-first manner. Every time a node is processed, the algorithm creates its children and labels all the new nodes and links created.

As mentioned earlier, the child of the node labeled S has one child corresponding to the pivot vertex p, and children for all non-neighbors of p. Importantly, we label each “call" with $\mathfrak{p}$ or $\mathfrak{h}$. This is central to getting unique representations of all the cliques.

Now for our main theorem about SCT.

Theorem 5.4.1. *Every clique C (in G) can be* uniquely *represented as $H(T) \cup Q$,*

Algorithm 16: SCTBuilder(G)
Output: SCT of G

Find degeneracy orientation of G, and let $N^+(v)$ denote the outneighborhood of a vertex v.
Initialize tree $\boldsymbol{T}$ with root labeled V.
For every $v \in V$, create a child of root with node label $N^+(v)$. Set the edge label to $(v, \mathfrak{h})$.
Insert all these child nodes into a queue $\boldsymbol{Q}$.
While $\boldsymbol{Q}$ is non-empty:
    Dequeue to get node γ. Let node label be S.
    If $S = \emptyset$, continue.
    Find $p \in S$ with largest $N(S, p)$ value.
    Create child node of γ with vertex label $N(S, p)$. Add this node to $\boldsymbol{T}$ and set the link label (of the new link) to $(p, \mathfrak{p})$. Also, add this node to $\boldsymbol{Q}$.
    Let $S \setminus (p \cup N(p)) = \{v_1, v_2, \ldots, v_\ell\}$ (listed in arbitrary order).
    For each $i \leq \ell$: create child node of γ labeled $N(S, v_i) \setminus \{v_1, v_2, \ldots, v_{i-1}\}$. Add this node to $\boldsymbol{T}$ and set link label to $(v_i, \mathfrak{h})$. Also add this node to $\boldsymbol{Q}$.
Return $\boldsymbol{T}$.

where $Q \subseteq P(T)$ and T is a root to leaf path in $\boldsymbol{T}$. (Meaning, for any other root to leaf path $T' \neq T$, $\forall Q \subseteq P(T')$, $C \neq H(T') \cup Q$.)

We emphasize the significance of this theorem. Every root to leaf path T represents a clique, given by the vertex set $H(T) \cup P(T)$. Every clique C is a subset of potentially many such sets; and there is no obvious bound on this number. So one can think of C "occurring" multiple times in the tree $\boldsymbol{T}$. But Theorem 5.4.1 asserts that if we take the labels into account ($H(T)$ vs $P(T)$), then there is a *unique* representation or "single occurrence" of C.

Proof. (of Theorem 5.4.1) Consider a node γ of $\boldsymbol{T}$ labeled S. We prove, by induction on $|S|$, that every clique $C \subseteq S$ can be expressed as $H(T) \cup Q$, where T is a path from γ to a leaf, and $Q \subseteq P(T)$. The theorem follows by setting γ to the root.

The base case is vacuously tree, since for empty S, all relevant sets are empty.

Now for the induction. We will have three cases. Let p be the pivot chosen in Step 8. (If S is the root, then there is no pivot. We will directly go to Case (iii) below.)

Case (i): $p \in C$. By construction, there is a link labeled $(p, \mathfrak{p})$ to a child of γ. Denote the child β. The child β has label $N(S, p)$. Observe that $C \setminus p$ is a clique in $N(S, p)$ (since by assumption, C is a clique in S.) By induction, there is a unique representation $C \setminus p = H(T) \cup Q$, for path T from the child node to a leaf and $Q \subseteq P(T)$. Moreover there cannot be a representation of C by a path rooted at β, since $N(S, p) \not\ni p$. Consider the path T' that contains T and starts from γ. Note that $H(T') = H(T)$ and $P(T') = P(T) \cup p$. We can express $C = H(T') \cup (Q \cup p)$, noting that $Q \cup p \subseteq P(T')$. This proves the existence of a representation. Moreover, there is only one representation using a path through β.

We need to argue that no other path can represent C. The pivoting is critical for this step. Consider any path rooted at γ, but not passing through β. It must pass through some other child, with corresponding links labeled $(v_i, \mathfrak{h})$, where v_i is a *non-neighbor* of p. Since $C \ni p$, a non-neighbor v_i cannot be in C. Moreover, for any path $\hat{T}$ passing through these other children, $\hat{T}$ must contain some non-neighbor. Thus, $\hat{T}$ cannot represent C.

Case (ii): $C \subseteq N(S, p)$. The argument is essentially identical to the one above. Note that $C \setminus p = C$, and by induction $C \setminus p$ has a unique representation using a path through β. For uniqueness, observe that C does not contain a non-neighbor of p. The previous argument goes through as is.

Case (iii): C contains a non-neighbor of p. Recall that $S \setminus (N(p) \cup p)$ (the set of non-neighbors in S) is denoted $\{v_1, v_2, \ldots, v_\ell\}$. Let i be the smallest index i such that $v_i \in C$. For any $1 \leq j \leq \ell$, let $N_j := N(S, v_j) \setminus \{v_1, v_2, \ldots, v_{j-1}\}$.

Observe that for all j, there is a child labeled N_j. Moreover, all the link labels have $\mathfrak{h}$, so for path T passing through N_j, $H(T) \ni v_j$. Thus, if T can represent C, it cannot pass through N_j for $j < i$. Moreover, if $j > i$, then $N_j \not\ni v_i$ and no path passing through this node can represent C.

Hence, if there is a path that can represent C, it must pass through N_i. Note that $C \setminus v_i$ is a clique contained in N_i. By induction, there is a unique path T rooted at N_i such that $C \setminus v_i = H(T) \cup Q$, for $Q \subseteq P(T)$. Let T' be the path that extends T to γ. Note that $H(T') = H(T) \cup v_i$, so $C = H(T') \cup Q$. The uniqueness of T implies the uniquesness of T'. □

5.5 Getting global and local counts

The tree $\boldsymbol{T}$ is succinct and yet one can extract fine-grained information from it about all cliques.

The storage complexity of the algorithm, as given, is potentially $O(\alpha^2 |\, SCT(G)|)$, since this is required to store the tree. In the proof of Theorem 6.3.1, we explain how to reduce the storage.

Proof. (of Theorem 6.3.1) **Correctness:** By Theorem 5.4.1, a root to leaf path T of $\boldsymbol{T}$ represents exactly $2^{P(T)}$ different cliques, with $\binom{P(T)}{i}$ of size $|H(T)| + i$. Moreover, over all T, this accounts for all cliques in the graph. This proves the correctness of global counts.

Pick a vertex $v \in H(T)$. For every subset of $P(T)$, we get a different clique containing v (that is uniquely represented by Theorem 5.4.1). This proves the correctness of Step 5. For a vertex $v \in P(T)$, we look at all subsets containing v. Equivalently, we get a different represented clique containing v for every subset of $P(T) \setminus v$. This proves the correctness of Step 6.

Algorithm 17: PIVOTER(G)
Output: Clique counts of G

1 Let $\boldsymbol{T} = SCTBuilder(G)$.
2 Initialize all clique counts to zero.
3 For every root to leaf path T in $\boldsymbol{T}$:
4 For every $0 \leq i \leq |P(T)|$, increment $C_{|H(T)|+i}$ by $\binom{P(T)}{i}$.
5 For every $v \in H(T)$ and every $0 \leq i \leq |P(T)|$, increment $c_{|H(T)|+i}(v)$ by $\binom{P(T)}{i}$.
6 For every $v \in P(T)$ and every $0 \leq i \leq |P(T)| - 1$, increment $c_{|H(T)|+i+1}(v)$ by $\binom{P(T)-1}{i}$.
7 For every edge $e(u, v), u \in H(T), v \in H(T), u \neq v$ and every $0 \leq i \leq |P(T)|$, increment $c_{|H(T)|+i}(e)$ by $\binom{P(T)}{i}$.
8 For every edge $e(u, v), u \in P(T), v \in H(T)$ and every $0 \leq i \leq |P(T)| - 1$, increment $c_{|H(T)|+i+1}(e)$ by $\binom{P(T)-1}{i}$.
9 For every edge $e(u, v), u \in P(T), v \in P(T), u \neq v$ and every $0 \leq i \leq |P(T)| - 2$, increment $c_{|H(T)|+i+2}(e)$ by $\binom{P(T)-2}{i}$.
10 Output the sets of values $\{C_k\}$, $\{c_k(v)\}$ and $\{c_k(e)\}$.

Pick an edge $e = (u, v), u \in H(T), v \in H(T)$. For every subset of $P(T)$, we get a different clique containing e (that is uniquely represented by Theorem 5.4.1). This proves the correctness of Step 7. For an edge $e = (u, v), u \in P(T), v \in H(T)$, we look at all subsets of $P(T)$ containing u. Equivalently, we get a different represented clique containing e for every subset of $P(T) \setminus u$. This proves the correctness of Step 8. For an edge $e = (u, v), u \in P(T), v \in P(T)$, we look at all subsets of $P(T)$ containing both u and v. Equivalently, we get a different represented clique containing e for every subset of $P(T) \setminus v \setminus u$. This proves the correctness of Step 9.

Running time (in terms of $|SCT(G)|$): Consider the procedure $SCTBuilder(G)$. Note that the size of $\boldsymbol{T}$ is at least n, so we can replace any running time dependence on n by $|\boldsymbol{T}|$. The degeneracy orientation can be found in $O(m+n)$ [173]. For the actual building of the tree, the main cost is in determining

the pivot and constructing the children of a node. Suppose a non-root node labeled S is processed. The above mentioned steps can be done by constructing the subgraph induced on S. This can be done in $O(|S|^2)$ time. Since this is not a root node, $|S| \leq \alpha$ (this is the main utility of the degeneracy ordering). Thus, the running time of S$CTBuilder(G) = O(\alpha^2|\boldsymbol{T}|) = O(\alpha^2|\, SCT(G)|)$.

Now we look at Pivoter. Note that the subsequent counting steps do *not* need the node labels in $\boldsymbol{T}$; for all path T, one only needs $P(T)$ and $H(T)$. The paths can be looped over by a DFS from the root. For each path, there are precisely $|P(T)|+1$ updates to global clique counts, and at most $|H(T)\cup P(T)|\times(|P(T)|+1)$ updates to per-vertex clique counts. The length of T is at most α, and thus both these quantities are $O(\alpha^2)$. Thus, the total running time is $O(\alpha^2|\, SCT(G)|)$ for global and per-vertex clique counting.

Similarly, for each path, at most $|H(T) \cup P(T)|^2 \times (|P(T)| + 1)$ updates are made to per-edge clique counts. This quantity is $O(\alpha^3)$. Thus, the total running time is $O(\alpha^3|\, SCT(G)|)$.

Running time (in terms of n and α): One crucial difference between the algorithm of Bron-Kerbosch and SCTBuilder is that in Bron-Kerbosch, the pivot vertex can be chosen not only from S but also from a set of already processed vertices. Hence, the tree obtained in Bron-Kerbosch can potentially be smaller than that of Pivoter. Despite this difference, the recurrence and bound on the worst case running time of S$CTBuilder$ is the same as Bron-Kerbosch.

Theorem 5.5.1. *Worst case running time of* S$CTBuilder$ *is* $O(n3^{\alpha/3})$.

Proof. Let $T(s)$ be the worst case running time required by SCTBuilder to process S where $s = |S|$.

Let $R = S \setminus N(p)$. Let $T_r(s)$ be the worst case running time of processing S when $|R| = r$. Note that when S is being processed it creates a total of r child

nodes.

Thus, $T(s) = \max_r\{T_r(s)\}$.

Note that all steps other than Step 9 and Step 11 take time $O(s^2)$. Say, they take time $p_1 s^2$, where $p_1 > 0$ is a constant.

Thus, we have that:

$$T_r(s) \leq \sum_{v \in R} T(|N(S, v)|) + p_1 s^2. \tag{5.1}$$

Moreover,

$$|N(S, v)| \leq s - r \leq s - 1, \forall v \in R. \tag{5.2}$$

This is because p has the largest neighborhood in S and p's neighborhood is of size atmost $s - r$, and since $|S| \geq 1, s - r \leq s - 1$.

Thus, Lemma 2 and Theorem 3 from [238] hold, which implies that $T(s) = O(3^{s/3})$. Since there are n vertices and their outdegree is atmost α, the worst case running time of SCTBuilder (which is also an upper bound for $|sct(G)|$) is $nT(\alpha) = O(n3^{\alpha/3})$ and hence, worst case running times of Pivoter for obtaining global, per-vertex and per-edge clique counts are $O(n\alpha 3^{\alpha/3})$, $O(n\alpha^2 3^{\alpha/3})$ and $O(n\alpha^3 3^{\alpha/3})$, respectively.

□

Storage cost: Currently, Pivoter is represented through two parts: the construction of $SCT(G)$ and then processing it to get clique counts. Conceptually, this is cleaner to think about and it makes the proof transparent. On the other hand, it requires storing $SCT(G)$, which is potentially larger than the input graph. A more space efficient implementation is obtained by combining these steps.

We do not give full pseudocode, since it is somewhat of a distraction. (The

details can be found in the code.) Essentially, instead of constructing $SCT(G)$ completely in breadth-first manner, we construct it depth-first through recursion. This will loop over all the paths of $\boldsymbol{T}$, but only store a single path at any stage. The updates to the clique counts are done as soon as any root to leaf path is constructed. The total storage of a path is the storage for all the labels on a path. As mentioned earlier in the proof of Theorem 6.3.1, all non-root nodes are labeled with sets of size at most α. The length of the path is at most α, so the total storage is $O(\alpha^2)$. A classic bound on the degeneracy is $\alpha \leq \sqrt{2m}$ (Lemma 1 of [61]), so the storage, including the input, is $O(m+n)$.

□

Parallel version of Pivoter: While this is not central to our results, we can easily implement a parallel version of Pivoter for *global* clique counts. We stress that our aim was not to delve into complicated parallel algorithms, and merely to see if there was a way to parallelize the counting involving minimal code changes. The idea is simple, and is an easier variant of the parallelism in kCList [79]. Observe that the children of the root of $SCT(G)$ correspond to finding cliques in the sets $N^+(v)$, for all v. Clique counting in each of these sets can be treated as an independent problem, and can be handled by an independent thread/subprocess. Each subprocess maintains its own array of global clique counts. The final result aggregates all the clique counts. The change ends up being a few lines of code to the original implementation.

Note that this becomes tricky for local counts. Each subprocess cannot afford (storage-wise) to store an entire copy of the local count data structure. The aggregation step would be more challenging. Nonetheless, it should be feasible for each subprocess to create local counts for $N^+(v)$, and appropriately aggregate all counts. We leave this for future work.

Counting k-cliques for a specific k: PIVOTER can be modified to obtain clique counts upto a certain user specified k (instead of counting for all k). Whenever the number of links marked $\mathfrak{h}$ becomes greater than k in any branch of the computation, we simply truncate the branch (as further calls in the branch will only yield cliques of larger sizes).

5.6 Experimental results

Preliminaries: All code for PIVOTER is available here: https://bitbucket.org/sjain12/pivoter/. We implemented our algorithms in C and ran our experiments on a commodity machine equipped with a 1.4GHz AMD Opteron(TM) processor 6272 with 8 cores and 2048KB L2 cache (per core), 6144KB L3 cache, and 128GB memory. We performed our experiments on a collection of social networks, web networks, and infrastructure networks from SNAP [49]. The graphs are simple and undirected (for graphs that are directed, we ignore the direction). A number of these graphs have more than 10 million edges, and the largest has more than 100 million edges. Basic properties of these graphs are presented in Tab. 5.1.

The data sets are split into two parts, in Tab. 5.1. The upper part are instances feasibly solved with past work (notably kClist40 [79]), while the lower part has instances that cannot be solved with previous algorithm (even after days). We give more details in §5.6.1.

Competing algorithms: We compare with (what we consider) are the state of the art clique counting algorithms: Turán-Shadow (TS) [134] and kClist40 [79].

kClist40: This algorithm by Danisch-Balalau-Sozio [79] uses degeneracy orientations and parallelization to enumerate all cliques. The kClist40 algorithm, to the best of our knowledge, is the only existing algorithm that can feasibly

compute all global counts for some graphs. Hence, our main focus is runtime comparisons with kClist40.

We note that the implementation of kClist40 visits every clique, but only updates the (appropriate) C_k. While it could technically compute local counts, that would require more expensive data structure updates. Furthermore, there would be overhead in combining the counts for independent threads, and it is not immediately obvious how to distribute the underlying data structure storing local counts. As a result, we are unaware of any algorithm that computes local counts (at the scale of dataset in Tab. 5.1).

We perform a simple optimization of kClist40, to make counting faster. Currently, when kClist40 encounters a clique, it enumerates every smaller clique contained inside it. For the purpose of counting though, one can trivially count all subcliques of a clique using formulas. We perform this optimization (to have a fair comparison with kClist40), and note significant improvements in running time.

In all our runs, for consistency, we run kClist with 40 threads. Note that we compare the *sequential* Pivoter with the *parallel* kClist40.

TS: This is the approximate clique counting algorithm describe in 3 [134]. It mines dense subgraphs (shadows) and samples cliques within the dense subgraphs to give an estimate. For fast randomized estimates, it is arguably the fastest algorithm. It runs significantly faster than a sequential implementation of kClist, but is typically comparable with a parallel implementation of kClist. It requires the entire shadow to be available for sampling which can require considerable space.

Graph	Vertices	Edges	Degen	Max clique	Pivoter (C_k)	Pivoter ($c_k(v)$)	Pivoter ($c_k(e)$)	Pivoter (C_k parallel)
Feasible by previous algorithms								
dblp-v5	1.56E+06	2.08E+06	15	10	7	7	8	19
dblp-v7	3.67E+06	4.18E+06	19	12	15	16	19	34
amazon0601	4.03E+05	2.44E+06	10	11	4	5	6	4
web-Google	8.76E+05	4.32E+06	44	44	8	9	15	9
youtube	1.13E+06	2.99E+06	51	17	7	8	11	9
cit-Patents	3.77E+06	1.65E+07	64	11	40	41	53	46
soc-pokec	1.63E+06	2.23E+07	47	29	68	75	93	44
Not feasible for previous algorithms								
Stanford	2.82E+05	1.99E+06	71	61	5	5	38	3
BerkStan	6.85E+05	6.65E+06	201	201	25	26	237	9
as-skitter	1.70E+06	1.11E+07	111	67	120	200	9245	75
com-orkut	3.07E+06	1.17E+08	253	51	5174	8802	99389	3441
com-lj	4.00E+06	3.47E+07	360	-	-	-	-	108000*

Table 5.1: Table shows the sizes, degeneracy, maximum clique size, and the time taken (in seconds) by Pivoter to obtain global $k-$clique counts, per-vertex and per edge $k-$cliques counts for all k. *For the com-lj graph, we were not able to get all $k-$clique counts in 1 day so we tested for the maximum k we could count in about a day. Pivoter was able to count the number of 9-cliques in 30 hours whereas kClist40 had not terminated even after 6 days.

5.6.1 Running time and comparison with other algorithms

Running time for global counting: We show the running time results in Tab. 5.1. For most of the graphs, Pivoter was able to count all k-cliques in seconds or minutes. For the largest `com-orkut` graph, Pivoter ran in 1.5 hours. This is a huge improvement on the state of the art. For the “infeasible" instances in Tab. 5.1, we do not get results even in two days using previous algorithms. (This is consistent with results in Table 2 of [79], where some of the graphs are also listed as “very large graphs" for which clique counting is hard.)

A notable hard instance is `com-lj` where Pivoter is unable to get all clique counts in a day. Again, previous work also notes this challenge, and only gives counts of 7-cliques. We can get some partial results for `com-lj`, as explained later.

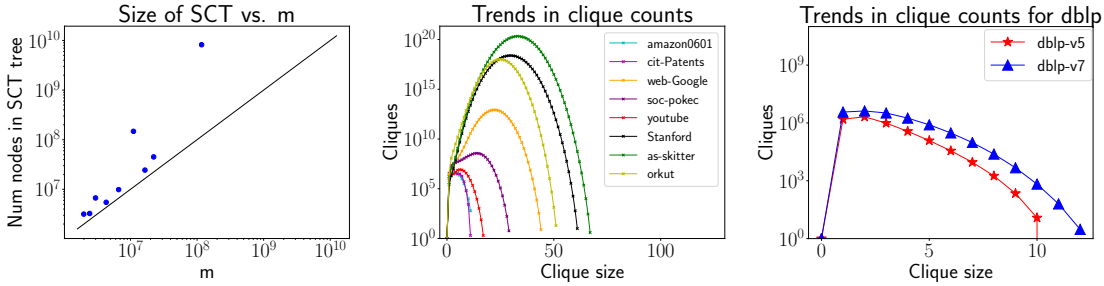
Feasible local counting: Notably, Pivoter can get per-vertex counts in less than twice the time of global clique counting. Thus, we get results for more

Graph	k=13,TS	k=13, kClist40	all k, Pivoter
Stanford	230	12600	5
BerkStan	1198	$>$ 172800	25
as-skitter	798	12480	120
com-orkut	$>$ 28800	$>$ 172800	5174

Table 5.2: Time taken in seconds by the state-of-the-art randomized (TS, short for TuránShadow) and parallel (kClist40) algorithms. Note that Pivoter obtains **all** $k-$clique counts for these graphs in a fraction of the time taken by other methods to count just 13-cliques.

k	k-cliques	kClist40	Pivoter
7	4.49E+15	2.2 hours	1.2 hours
8	1.69E+16	42.5 hours	6.4 hours
9	5.87E+17	$>$ 6 days	30 hours
10	1.89E+19	$>$ 6 days	5.9 days

Table 5.3: Table shows the time taken to count k-cliques for com-lj graph. For k=9, Pivoter terminated in about 30 hours where kClist40 had not terminated in 6 days.



(a) Number of nodes in SCT vs m

(b) Trends in different graphs

(c) Trends in dblp over time.

Figure 5.2: Fig. 5.2a shows the number of nodes in the SCT vs the number of edges (m) for different graphs. The running time of Pivoter is directly proportional to the SCT size which seems to be roughly linear in the number of edges. Fig. 5.2b shows the trends in clique counts for a number of graphs. For some of the graphs, the complete distribution of their clique counts has been obtained for the first time. Fig. 5.2c shows the trends in the clique counts of 2 different versions over time of the dblp graph.

graphs in a few minutes, and can process the `com-orkut` graph within 3 hours. We consider this a significant achievement, given the combinatorial explosion of

clique counting.

PIVOTER is also able to get per-edge clique counts, though it can take an order of magnitude more time than global clique counting. Note that for obtaining the per-vertex and per-edge $k-$clique counts, the result data structure can become extremely large. Indeed, most of the time is spent in updating the data structure, rather than in constructing the SCT. Nonetheless, for all but the `as-skitter` and `com-orkut` graph, it runs in minutes.

Comparison with state of the art: We only focus on the "infeasible" instances of Tab. 5.1. For all the other instances, both PIVOTER and kClist40 get results within two minutes. For space considerations, we do not report all the running times for such instances. It is worth noting that the sequential PIVOTER is comparable to the parallel kClist40 (when they both terminate).

In Tab. 5.2, we report times on TS and kClist40 on the hard datasets. We are unable to get all values of C_k using either of these two method. We run these algorithms for up to 100 times the running time of PIVOTER or two days, whichever is shorter. We try to count the largest feasible clique count.

Let us focus on kClist40, where we cannot go beyond counting 13-cliques (we note that this is consistent with results reported in [79]). Notably, in the `BerkStan` graph, kClist40 needs more than 2 days to count 13-cliques, while PIVOTER gets all clique counts in a minute. As mentioned earlier, clique counting on the large `com-orkut` graph is done in a few hours by PIVOTER, while even counting 13-cliques takes kClist40 more than two days.

TS also does not scale well for larger cliques and PIVOTER is faster than TS. For example, for the `Stanford` graph, TS required 230 seconds to estimate the number of 13-cliques whereas PIVOTER obtained all $k-$clique counts in 5 seconds. Similar trends are observed with other graphs.

Parallel global clique counting: As mentioned in §5.5, we do a simple parallelization of the global clique counting of Pivoter using 30 threads. It gives moderate benefits for most instances, and about a factor two speedup for large instances. For the challenging `com-lj` instances, the effect is much more dramatic. We are able to count 7-cliques in an hour using the parallel Pivoter, while the sequential version takes more than a day.

Performance on com-lj. This is a particularly challenging graph. The sequential version of Pivoter for counting all k-cliques did not terminate within a day, so we used the parallel version of our algorithm to show a comparison for global counts upto $k = 10$. We can truncate the SCT to get cliques of some fixed size. Tab. 5.3 shows the results. Even for this graph, the parallel version of Pivoter is faster than kClist40 for $k = 7$ and beyond. kClist40 did not terminate after six days, for $k = 9$ and beyond. We note the astronomical number of 10-cliques ($> 10^{19}$), which makes enumeration infeasible, but Pivoter was able to get the exact count.

Size of $SCT(G)$: In Fig. 5.2a, we plot the number of nodes of $SCT(G)$ as a function of the number of edges in G. We observe that for most graphs, the size is quite close to m, explaining why Pivoter is efficient.

5.6.2 Demonstrations of Pivoter

Global and local cliques have numerous applications. It is outside the scope of this work for detailed demonstrations, but we show a few examples in this section.

As mentioned earlier, local clique counts are an important aspect of graph processing. In Fig. 5.1b and Fig. 5.1c, we plot the per-vertex clique distributions, also called the *graphlet* degree distribution in bioinformatics [199] for the as-skitter and web-Stanford graphs. We choose values of $k = 5, 10, 15, 20, 25$. Then, we plot

the function $f_k(b)$ that is the number of vertices that participate in b k-cliques. We notice interesting trends. While the as-skitter graph has a nicely decaying f_k function, there is much more noise in web-Stanford. It would be interesting to design models that can capture such behavior in the local clique counts.

In Fig. 5.2b, we plot the C_k values for a number of graphs. We notice, for example, that the `soc-pokec` network has a "flatter" distribution of C_k for some of the initial values, while the `com-orkut` graph looks much closer to a binomial distribution. The latter suggests that the bulk of cliques are coming from the maximum clique in the `com-orkut` graph, but not so in the `soc-pokec` graph.

In Fig. 5.2c, we plot the k-clique counts (vs k) for two different versions across time for the DBLP citation network [236]. Interestingly, despite the later version only having less than twice as many edges, the clique distribution (plotted in semilog) has a much bigger difference. It appears that the graph is becoming significantly dense in certain part. This sort of analysis may help in understanding dynamic graphs.

5.7 Future work

The success of [79] in using parallelization for clique counting suggests combining their ideas with our pivoting techniques. We may be able to come up with an efficient parallel building of the SCT that is much faster than our current implementation. Indeed, the results on the `com-lj` graph suggest that even Pivoter has its limits for real data.

An orthogonal approach would be to exploit the sampling techniques in the Turán-Shadow algorithm [134]. For many subgraph counting problems, randomization has been the key to truly practical algorithms. We believe that Pivoter could be made faster with these ideas.

Moreover, it also gives per-edge and per-vertex $k-$clique counts. This is the first time that $k-$clique counts are known for many of the graphs we experimented with and this will open doors for further use of cliques in generation and analysis of graphs.

Chapter 6

Estimating the degree distribution

6.1 Introduction

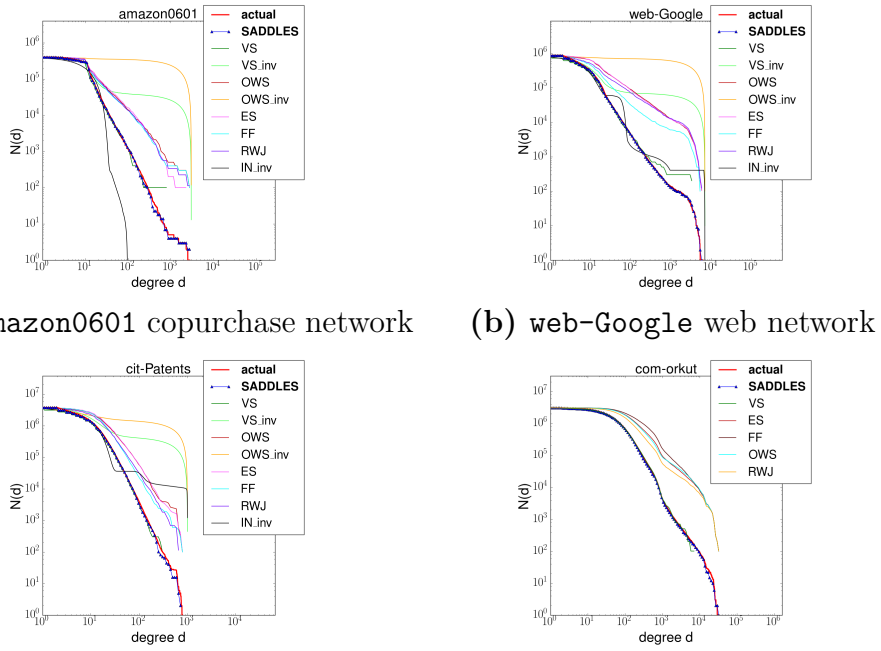
Continuing the theme of sampling, in this chapter, we explore how sampling can be used to estimate some property of the graph when we have only query access to the graph. Contrary to the setup of clique counting problems seen in earlier chapters where we have access to the whole graph, in this setup, we have access to only a part of the graph and the goal is to be able to relate the properties of the part to the properties of the whole. In this work, the goal is to estimate the degree distribution of the graph.

In domains as diverse as social sciences, biology, physics, cybersecurity, graphs are used to represent entities and the relationships between them. This has led to the explosive growth of network science as a discipline over the past decade. One of the hallmarks of network science is the occurrence of specific graph properties that are common to varying domains, such as heavy tailed degree

distributions, large clustering coefficients, and small-world behavior. Arguably, the most significant among these properties is the degree distribution, whose study led to the foundation of network science [29, 46, 101].

Given an undirected graph G, the degree distribution (or technically, histogram) is the sequence of numbers $n(1), n(2), \ldots$, where $n(d)$ is the number of vertices of degree d. In almost all real-world scenarios, the average degree is small, but the variance (and higher moments) is large. Even for relatively large d, $n(d)$ is still non-zero, and $n(d)$ typically has a smooth non-increasing behavior. In Fig. 6.1, we see the typical degree distribution behavior. The average degree in a Google web network is less than 10, but the maximum degree is more than 5000. There are also numerous vertices with all intermediate degrees. This is referred to as a “heavy tailed" distribution. The degree distribution, especially the tail, is of significant relevance to modeling networks, determining their resilience, spread of information, and for algorithmics [17, 52, 67, 85, 182, 184, 185, 192, 214].

With full access to G, the degree distribution can be computed in linear time, by simply determining the degree of each vertex. Yet in many scenarios, we only have *partial* access to the graph, provided through some graph samples. A naive extrapolation of the degree distribution can result in biased results. The seminal research paper of Faloutsos et al. claimed a power law in the degree distribution on the Internet [101]. This degree distribution was deduced by measuring a power law distribution in the graph sample generated by a collection of traceroute queries on a set of routers. Unfortunately, it was mathematically and empirically proven that traceroute responses can have a power law *even if the true network does not* [5, 64, 156, 193]. In general, a direct extrapolation of the degree distribution from a graph subsample is not valid for the underlying graph. This leads to the primary question behind our work.



(a) `amazon0601` copurchase network **(b)** `web-Google` web network

(c) `cit-Patents` citation network **(d)** `com-orkut` social network

Figure 6.1: The output of SADDLES on a collection of networks: `amazon0601` (403K vertices, 4.9M edges), `web-Google` (870K vertices, 4.3M edges), `cit-Patents` (3.8M vertices, 16M edges), `com-orkut` social network (3M vertices, 117M edges). SADDLES samples 1% of the vertices and gives accurate results for the entire (cumulative) degree distribution. For comparison, we show the output of a number of sampling algorithms from past work, each run with the same number of samples. (Because of the size of `com-Orkut`, methods involving optimization [270] fail to produce an estimate in reasonable time.)

How can we provably and practically estimate the degree distribution without seeing the entire graph?

There is a rich literature in statistics, data mining, and physics on estimating graph properties (especially the degree distribution) using a small subsample [8, 10, 13, 88, 158, 160, 167, 202, 229, 270]. Nonetheless, there is no provable algorithm for the entire degree distribution, with a formal analysis on when it is sublinear in the number of vertices. Furthermore, most empirical studies typically sample 10-30% of the vertices for reasonable estimates.

6.1.1 Problem description

We focus on the *complementary cumulative degree histogram* (often called the cumulative degree distribution) or *ccdh* of G. This is the sequence $\{N(d)\}$, where $N(d) = \sum_{r \geq d} n(r)$ is the number of vertices of degree at least d. The ccdh is typically used for fitting distributions, since it averages out noise and is monotonic [65]. Our aim is to get an accurate bicriteria approximation to the ccdh of G, at all values of d.

Definition 6.1.1. *The sequence* $\{\widetilde{N}(d)\}$ *is an* $(\varepsilon, \varepsilon)$*-estimate of the ccdh if* $\forall d$, $(1-\varepsilon)N((1+\varepsilon)d) \leq \widetilde{N}(d) \leq (1+\varepsilon)N((1-\varepsilon)d)$.

Computing an $(\varepsilon, \varepsilon)$-estimate is significantly harder than approximating the ccdh using standard distribution measures. Statistical measures, such as the KS-distance, χ^2, ℓ_p-norms, etc. tend to ignore the tail, since (in terms of probability mass) it is a negligible portion of the distribution. An $(\varepsilon, \varepsilon)$-estimate is accurate for all d.

The query model: A formal approach requires specifying a *query model* for accessing G. We look to the subfields of property testing and sublinear algorithms within theoretical computer science for such models [111, 112]. Consider the following three kinds of *queries*.

- Vertex queries: acquire a uniform random vertex $v \in V$.
- Neighbor queries: given $v \in V$, acquire a uniform random neighbor u of V.
- Degree queries: given $v \in V$, acquire the degree d_v.

An algorithm is only allowed to make these queries to process the input. It has to make some number of queries, and finally produce an output. We discuss two query models, and give results for both.

The Standard Model (SM) All queries allowed: This is the standard model in numerous sublinear algorithms results [90, 91, 111–113]. Furthermore,

most papers on graph sampling implicitly use this model for generating subsamples. Indeed, any method involving crawling from a random set of vertices and collecting degrees is in the SM. This model is the primary setting for our work, and allows for comparison with rich body of graph sampling algorithms. It is worth noting that in the SM, one can determine the entire degree distribution in $O(n \log n)$ queries (the extra $\log n$ factor comes from the coupon collector bound of finding all the vertices through uniform sampling). Thus, it makes sense to express the number of queries made by an algorithm as a fraction of n. Alternately, the number of queries is basically the number of vertices encountered by the algorithm. Thus, a sublinear algorithm makes $o(n)$ queries.

The Hidden Degrees Model (HDM) Vertex and neighbor queries allowed, not degree queries: This is a substantially weaker model. In numerous cybersecurity and network monitoring settings, an algorithm cannot query for degrees, and has to infer them indirectly. Observe that this model is significantly harder than the SM. It takes $O((m+n)\log n)$ to determine all the degrees, since one has to at least visit all the edges to find degrees exactly. In this model, we express the number of queries as a fraction of m.

Regarding uniform random vertex queries: This is a fairly powerful query, that may not be realizable in all situations. Indeed, Chierichetti et al. explicitly study this problem in social networks and design (non-trivial) algorithms for sampling uniform random vertices [62]. In a previous work, Dasgupta, Kumar, and Sarlos study algorithms for estimating average degree when only random walks are possible [81]. Despite this power, we believe that SM is a good testbed for understanding *when* a small sample of a graph provably gives properties of the whole. Furthermore, in the context of graph sampling, access to uniform random vertices is commonly (implicitly) assumed [10, 88, 158, 160, 196, 202, 270]. The

vast majority of experiments conducted often use uniform random vertices.

As a future direction, we believe it is important to investigate sampling models without random vertex queries.

6.1.2 Our contributions

Our main theoretical result is a new sampling algorithm, the Sublinear Approximations for Degree Distributions Leveraging Edge Samples, or SADDLES. This algorithm provably provides $(\varepsilon, \varepsilon)$-approximations for the ccdh. We show how to design SADDLES under both the SM and the HDM. We apply SADDLES on a variety of real datasets and demonstrate its ability to accurately approximate the ccdh with a tiny sample of the graph.

• **Sampling algorithm for estimating ccdh:** Our algorithm combines a number of techniques in random sampling to get $(\varepsilon, \varepsilon)$-estimates for the ccdh. A crucial component is an application of an edge simulation technique, first devised by Eden et al. in the context of triangle counting [90, 91]. This (theoretical) technique shows how to get a collection of weakly correlated uniform random edges from independent uniform vertices. SADDLES employs a weighting scheme on top of this method to estimate the ccdh.

• **Heavy tails leads to sublinear algorithms:** The challenge in analyzing SADDLES is in finding parameters of the ccdh that allow for sublinear query complexity. To that end, we discuss two parameters that measure "heaviness" of the distribution tail: the classic h-index and a newly defined z-index. We prove that the query complexity of SADDLES is sublinear (for both models) whenever these indices are large.

• **Excellent empirical behavior:** We deploy an implementation of SADDLES on a collection of large real-world graphs. In all instances, we achieve

extremely accurate estimates for the entire ccdh by sampling at most 1% of the vertices of the graph. Refer to Fig. 6.1. Observe how SADDLES tracks various jumps in the ccdh, for all graphs in Fig. 6.1.

• **Comparison with existing sampling methods:** A number of graph sampling methods have been proposed in practice, such as vertex sampling (VS), snowball sampling (OWS), forest-fire sampling (FF), induced graph sampling (IN), random walk (RWJ), edge sampling (ES) [10, 88, 158, 160, 196, 202, 270]. A recent work of Zhang et al. explicitly addresses biases in these sampling methods, and fixes them using optimization techniques [270]. We run head-to-head comparisons with all these sampling methods, and demonstrate the SADDLES gives significantly better practical performance. Fig. 6.1 shows the output of all these sampling methods with a total sample size of 1% of the vertices. Observe how across the board, the methods make erroneous estimates for most of the degree distribution. The errors are also very large, for all the methods. This is consistent with previous work, where methods sample more than 10% of the number of vertices.

6.1.3 Theoretical results in detail

Our main theoretical result is a new sampling algorithm, the Sublinear Approximations for Degree Distributions Leveraging Edge Samples, or SADDLES.

We first demonstrate our results for power law degree distributions [29, 46, 101]. Statistical fitting procedures suggest they occur to some extent in the real-world, albeit with much noise [65]. The classic power law degree distribution sets $n(d) \propto 1/d^{\gamma}$, where γ is typically in $[2, 3]$. We build on this to define a power law lower bound.

Definition 6.1.2. *Fix $\gamma > 2$. A degree distribution is bounded below by a power law with exponent γ, if the ccdh satisfies the following property. There exists a constant $\tau > 0$ such that for all d, $N(d) \geq \lfloor \tau n / d^{\gamma-1} \rfloor$.*

The following is a corollary of our main result. For convenience, we will suppress query complexity dependencies on ε and $\log n$ factors, using $\widetilde{O}(\cdot)$.

Theorem 6.1.3. *Suppose the degree distribution of G is bounded below by a power law with exponent γ. Let the average degree be denoted by $\overline{d}$. For any $\varepsilon > 0$, the* SADDLES *algorithm outputs (with high probability) an $(\varepsilon, \varepsilon)$-approximation to the ccdh and makes the following number of queries.*

- *SM:* $\widetilde{O}(n^{1-\frac{1}{\gamma}} + n^{1-\frac{1}{\gamma-1}}\overline{d})$
- *HDM:* $\widetilde{O}(n^{1-\frac{1}{2(\gamma-1)}}\overline{d})$

In most real-world instances, the average degree $\overline{d}$ is typically constant. Thus, the complexities above are strongly sublinear. For example, when $\gamma = 2$, we get $\widetilde{O}(n^{1/2})$ for both models. When $\gamma = 3$, we get $\widetilde{O}(n^{2/3})$ and $\widetilde{O}(n^{3/4})$.

Our main result is more nuanced, and holds for all degree distributions. If the ccdh has a heavy tail, we expect $N(d)$ to be reasonably large even for large values of d. We describe two formalisms of this notion, through *fatness indices*.

Definition 6.1.4. *The* h-index *of the degree distribution is the largest d such that there are at least d vertices of degree at least d.*

This is the exact analogy of the bibliometric h-index [123]. As we show in the §6.2.1, h can be approximated by $\min_d (d + N(d))/2$. A more stringent index is obtained by replacing the arithmetic mean by the (smaller) geometric mean.

Definition 6.1.5. *The* z-index *of the degree distribution is* $z = \min_{d:N(d)>0} \sqrt{d \cdot N(d)}$.

Our main theorem asserts that large h and z indices lead to a sublinear algorithm for degree distribution estimation. Theorem 6.1.3 is a direct corollary obtained by plugging in values of the indices for power laws.

Theorem 6.1.6. *For any $\varepsilon > 0$, the* SADDLES *algorithm outputs (with high probability) an $(\varepsilon, \varepsilon)$-approximation to the ccdh, and makes the following number of queries.*

- *SM:* $\widetilde{O}(n/h + m/z^2)$
- *HDM:* $\widetilde{O}(m/z)$

6.1.4 Challenges and Main Idea

The heavy-tailed behavior of the real degree distribution poses the primary challenge to computing $(\varepsilon, \varepsilon)$-estimates to the ccdh. As d increases, there are fewer and fewer vertices of that degree. Sampling uniform random vertices is inefficient when $N(d)$ is small. A natural idea to find high degree vertices to pick a random neighbor of a random vertex. Such a sample is more likely to be a high degree vertex. This is the idea behind methods like snowball sampling, forest fire sampling, random walk sampling, graph sample-and-hold, etc. [10, 88, 158, 160, 196, 202, 270]. But these lead to biased samples, since vertices with the same degree may be picked with differing probabilities.

A direct extrapolation/scaling of the degrees in the observed graph does not provide an accurate estimate. Our experiments show that existing methods always miss the head or the tail. A more principled approach was proposed recently by Zhang et al. [270], by casting the estimation of the unseen portion of the distribution as an optimization problem. From a mathematical standpoint, the vast majority of existing results tend to analyze the KS-statistic, or some ℓ_p-norm. As we mentioned earlier, this does not work well for measuring the quality of the

estimate at all scales. As shown by our experiments, none of these methods give accurate estimate for the entire ccdh with less than 5% of the vertices.

The main innovation in SADDLES comes through the use of a recent theoretical technique to simulate edge samples through vertex samples [90, 91]. The sampling of edges occurs through two stages. In the first stage, the algorithm samples a set of r vertices and sets up a distribution over the sampled vertices such that any edge adjacent to a sampled vertex may be sampled with uniform probability. In the second stage, it samples q edges from this distribution. While a single edge is uniform random, the set of edges are correlated.

For a given d, we define a weight function on the edges, such that the total weight is exactly $N(d)$. SADDLES estimates the total weight by scaling up the average weight on a random sample of edges, generated as discussed above. The difficulty in the analysis is the correlation between the edges. Our main insight is that if the degree distribution has a fat tail, this correlation can be contained even for sublinear r and q. Formally, this is achieved by relating the concentration behavior of the average weight of the sample to the h and z-indices. The final algorithm combines this idea with vertex sampling to get accurate estimates for all d.

The hidden degrees model is dealt with using birthday paradox techniques formalized by Ron and Tsur [204]. It is possible to estimate the degree d_v using $O(\sqrt{d_v})$ neighbor queries. But this adds overhead to the algorithm, especially for estimating the ccdh at the tail. As discussed earlier, we need methods that bias towards higher degrees, but this significantly adds to the query cost of actually estimating the degrees.

6.1.5 Related Work

There is a rich body of literature on generating a graph sample that reveals graph properties of the larger “true" graph. We do not attempt to fully survey this literature, and only refer to results directly related to our work. The works of Leskovec & Faloutsos [160], Maiya & Berger-Wolf [167], and Ahmed, Neville, & Kompella [10, 13] provide excellent surveys of multiple sampling methods.

There are a number of sampling methods based on random crawls: forest-fire [160], snowball sampling [167], and expansion sampling [160]. As has been detailed in previous work, these methods tend to bias certain parts of the network, which can be exploited for more accurate estimates of various properties [160, 167, 202]. A series of papers by Ahmed, Neville, and Kompella [8–10, 13] have proposed alternate sampling methods that combine random vertices and edges to get better representative samples.

All these results aim to capture numerous properties of the graph, using a single graph sample. Nonetheless, there is much previous work focused on the degree distribution. Ribiero and Towsley [202] and Stumpf and Wiuf [229] specifically study degree distributions. Ribiero and Towsley [202] do detailed analysis on degree distribution estimates (they also look at the ccdh) for a variety of these sampling methods. Their empirical results show significant errors either at the head or the tail. We note that almost all these results end up sampling up to 20% of the graph to estimate the degree distribution.

Zhang et al. observe that the degree distribution of numerous sampling methods is a random linear projection of the true distribution [270]. They attempt to invert this (ill-conditioned) linear problem, to correct the biases. This leads to improvement in the estimate, but the empirical studies typically sample more than 10% of the vertices for good estimates.

A recent line of work by Soundarajan et al. on *active probing* also has flavors of graph sampling [226, 227]. In this setting, we start with a small, arbitrary subgraph and try to grow this subgraph to achieve some coverage objective (like discover the maximum new vertices, find new edges, etc.). The probing schemes devised in these papers outperform uniform random sampling methods for coverage objectives.

Some methods try to match the shape/family of the distribution, rather than estimate it as a whole [229]. Thus, statistical methods can be used to estimate parameters of the distribution. But it is reasonably well-established that real-world degree distributions are rarely pure power laws in most instances [65]. Indeed, fitting a power law is rather challenging and naive regression fits on log-log plots are erroneous, as results of Clauset-Shalizi-Newman showed [65].

The subfield of *property testing and sublinear algorithms for sparse graphs* within theoretical computer science can be thought of as a formalization of graph sampling to estimate properties. Indeed, our description of the main problem follows this language. There is a very rich body of mathematical work in this area (refer to Ron's survey [203]). Practical applications of graph property testing are quite rare, and we are only aware of one previous work on applications for finding dense cores in router networks [114]. The specific problem of estimating the average degree (or the total number of edges) was studied by Feige [103] and Goldreich-Ron [112]. Gonen et al. and Eden et al. focus on the problem of estimating higher moments of the degree distribution [91, 113]. One of the main techniques we use of simulating edge queries was developed in sublinear algorithms results of Eden et al. [90, 91] in the context of triangle counting and degree moment estimation. We stress that all these results are purely theoretical, and their practicality is by no means obvious.

On the practical side, Dasgupta, Kumar, and Sarlos study average degree estimation in real graphs, and develop alternate algorithms [81]. They require the graph to have low mixing time and demonstrate that the algorithm has excellent behavior in practice (compared to implementations of Feige's and the Goldreich-Ron algorithm [103, 112]). Dasgupta et al. note that sampling uniform random vertices is not possible in many settings, and thus they consider a significantly weaker setting than SM or HDM. Chierichetti et al. focus on sampling uniform random vertices, using only a small set of seed vertices and neighbor queries [62].

We note that there is a large body of work on sampling graphs from a stream [175]. This is quite different from our setting, since a streaming algorithm observes every edge at least once. The specific problem of estimating the degree distribution at all scales was considered by Simpson et al. [223]. They observe many of the challenges we mentioned earlier: the difficulty of estimating the tail accurately, finding vertices at all degree scales, and combining estimates from the head and the tail.

6.2 Preliminaries

We say that the input graph G has n vertices and m edges and $m \geq n$ (since isolated vertices are not relevant here). For any vertex v, let $\Gamma(v)$ be the neighborhood of v, and d_v be the degree. As mentioned earlier, $n(d)$ is the number of vertices of degree d and $N(d) = \sum_{r \geq d} n(r)$ is the ccdh at d. We use "u.a.r." as a shorthand for "uniform at random". We stress that the all mention of probability and error is with respect to the randomness of the sampling algorithm. There is no stochastic assumption on the input graph G. We use the shorthand $A \in (1 \pm \alpha)B$ for $A \in [(1-\alpha)B, (1+\alpha)B]$. We will apply the following (rescaled)

Chernoff bound.

Theorem 6.2.1. *[Theorem 1 in [83]] Let $X_1, X_2, \ldots, X_k$ be a sequence of iid random variables with expectation μ. Furthermore, $X_i \in [0, B]$.*

- *For $\varepsilon < 1$, $\Pr[|\sum_{i=1}^{k} X_i - \mu k| \geq \varepsilon \mu k] \leq 2\exp(-\varepsilon^2 \mu k/3B)$.*
- *For $t \geq 2e\mu$, $\Pr[\sum_{i=1}^{k} X_i \geq tk] \leq 2^{-tk/B}$.*

6.2.1 More on Fatness indices

The following characterization of the h-index will be useful for analysis. Since $(d + N(d))/2 \leq \max(d, N(d)) \leq d + N(d)$, this proves that $\min_d (d + N(d))/2$ is a 2-factor approximation to the h-index.

Lemma 6.2.2. $\min_d \max(d, N(d)) \in \{h, h+1\}$

Proof. Let $s = \min_d \max(d, N(d))$ and let the minimum be attained at d^*. If there are multiple minima, let d^* be the largest among them. We consider two cases. (Note that $N(d)$ is a monotonically non-increasing sequence.)

Case 1: $N(d^*) \geq d^*$. So $s = N(d^*)$. Since d^* is the largest minimum, for any $d > d^*$, $d > N(d^*)$. (If not, then the minimum is also attained at $d > d^*$.) Thus, $d > N(d^*) \geq N(d)$. For any $d < d^*$, $N(d) \geq N(d^*) \geq d^* > d$. We conclude that d^* is largest d such that $N(d) \geq d$. Thus, $h = d^*$.

If $s \neq h$, then $d^* < N(d^*)$. Then, $N(d^* + 1) < N(d^*)$, otherwise the minimum would be attained at $d^* + 1$. Furthermore, $\max(d^* + 1, N(d^* + 1)) > N(d^*)$, implying $d^* + 1 > N(d^*)$. This proves that $h + 1 > s$.

Case 2: $d^* > N(d^*)$. So $s = d^*$. For $d > d^*$, $N(d) \leq N(d^*) < d^* < d$. For $d < d^*$, $N(d) \geq d^* > d$ (if $N(d) < d^*$, then d^* would not be the minimizer). Thus, $d^* - 1$ is the largest d such that $N(d) \geq d$, and $h = d^* - 1 = s - 1$. □

The h-index does not measure d vs $N(d)$ at different scales, and a large h-index only ensures that there are "enough" high-degree vertices. For instance, the h-index does not distinguish between two different distributions whose ccdh N_1 and N_2 are such that $N_1(100) = 100$ and $N_1(d) = 0$ for $d > 100$, and $N_2(100,000) = 100$ and $N_2(d) = 100$ for all other values of $d \geq 100$. The h-index in both these cases is 100.

The h and z-indices are related to each other.

Claim 6.2.3. $\sqrt{h} \leq z \leq h$.

Proof. Since $N(d)$ is integral, if $N(d) > 0$, then $N(d) \geq 1$. Thus, for all $N(d) > 0$, $\sqrt{\max(d, N(d))} \leq d \cdot N(d) \leq \max(d, N(d))$. We take the minimum over all d to complete the proof. □

To give some intuition about these indices, we compute the h and z index for power laws. The classic power law degree distribution sets $n(d) \propto 1/d^\gamma$, where γ is typically in $[2, 3]$.

Claim 6.2.4. *If a degree distribution is bounded below by a power law with exponent γ, then $h = \Omega(n^{\frac{1}{\gamma}})$ and $z = \Omega(n^{\frac{1}{2(\gamma-1)}})$.*

Proof. Consider $d \leq \tau n^{1/\gamma}$, where τ is defined according to Definition 6.1.2. Then, $N(d) \geq \lfloor \tau n/(\tau^{1/\gamma} n^{(\gamma-1)/\gamma}) \rfloor = \Omega(n^{1/\gamma})$. This proves the h-index bound.

Set $d^* = (\tau n)^{\frac{1}{\gamma-1}}$. For $d \leq d^*$, $N(d) \geq 1$ and $d \cdot N(d) \geq (\tau/2)n/d^{\gamma-2} = \Omega(n^{\frac{1}{\gamma-1}})$. If there exists no $d > d^*$ such that $N(d) > 0$, then $z = \Omega(n^{\frac{1}{2(\gamma-1)}})$. If there does exist some such d, then $z = \Omega(\sqrt{d^*})$ which yields the same value. □

Plugging in values, for $\gamma = 2$, both h and z are $\Omega(\sqrt{n})$. For $\gamma = 3$, $h = \Theta(n^{1/3})$ and $z = \Theta(n^{1/4})$.

6.2.2 Simulating degree queries for HDM

The Hidden Degrees Model does not allow for querying the degree d_v of a vertex v. Nonetheless, it is possible to get accurate estimates of d_v by sampling u.a.r. neighbors (with replacement) of v. This can be done by using the birthday paradox argument, as formalized by Ron and Tsur [204]. Roughly speaking, one repeatedly samples neighbors until the same vertex is seen twice. If this happens after t samples, t^2 is a constant factor approximation for d_v. This argument can be refined to get accurate approximations for d_v using $O(\sqrt{d_v})$ random edge queries.

Theorem 6.2.5. *[Theorem 3.1 of [204], restated] Fix any $\alpha > 0$. There is an algorithm that outputs a value in $(1 \pm \alpha)d_v$ with probability $> 2/3$, and makes an expected $O(\sqrt{d_v}/\alpha^2)$ u.a.r. neighbor samples.*

For the sake of the theoretical analysis, we will simply assume this theorem. In the actual implementation of SADDLES, we will discuss the specific parameters used. It will be helpful to abstract out the estimation of degrees through the following corollary. The procedure DEG(v) will be repeatedly invoked by SADDLES. This is a direct consequence of setting $\alpha - \varepsilon/10$ and applying Theorem **??** with $\delta = 1/n^3$.

Corollary 6.2.6. *There is an algorithm DEG that takes as input a vertex v, and has the following properties:*

- *For all v: with probability $> 1 - 1/n^3$, the output DEG(v) is in $(1 \pm \varepsilon/10)d_v$.*
- *The expected running time and query complexity of DEG(v) is $O(\varepsilon^{-2}\sqrt{d_v}\log n)$.*

We will assume that invocations to *DEG* with the same arguments use the same sequence of random bits. Alternately, imagine that a call to $DEG(v, \varepsilon)$ stores the output, so subsequent calls output the same value. For the sake of

analysis, it is convenient to imagine that $\mathrm{DEG}(v)$ is called once for all vertices v, and these results are stored.

Definition 6.2.7. *The output $DEG(v)$ is denoted by $\hat{d}_v$. The random bits used in all calls to DEG is collectively denoted Λ. (Thus, Λ completely specifies all the values $\{\hat{d}_v\}$.) We say Λ is* good *if $\forall v \in V$, $\hat{d}_v \in (1 \pm \varepsilon/10)d_v$.*

The following is a consequence of conditional probabilities.

Claim 6.2.8. *Consider any event $\mathcal{A}$, such that for any good Λ, $\Pr[\mathcal{A}|\Lambda] \geq p$. Then $\Pr[\mathcal{A}] \geq p - 1/n^2$.*

Proof. The probability that Λ is not good is at most the probability that for *some* v, $\mathrm{DEG}(v) \notin (1 \pm \varepsilon/10)$. By the union bound and Corollary 6.2.6, the probability is at most $1/n^2$.

Note that
$\Pr[\mathcal{A}] \geq \sum_{\Lambda \text{good}} \Pr[\Lambda] \Pr[\mathcal{A}|\Lambda] \geq p \Pr[\Lambda \text{ is good}]$. Since Λ is good with probability at least $1 - 1/n^2$, $\Pr[\mathcal{A}] \geq (1 - 1/n^2)p \geq p - 1/n^2$. □

For any fixed Λ, we set $\widehat{N_\Lambda}(d)$ to be $|\{v|\hat{d}_v \geq d\}|$. We will perform the analysis of SADDLES with respect to the $\widehat{N_\Lambda}$-values.

Claim 6.2.9. *Suppose Λ is good. For all v, $\widehat{N_\Lambda}(v) \in [N((1 + \varepsilon/9)d), N((1 - \varepsilon/9)d)]$.*

Proof. Since Λ is good, $\forall u, \hat{d}_u \in (1 \pm \varepsilon/10)d_u$, Furthermore, if $d_u \geq (1 + \varepsilon/9)d$, then $\hat{d}_u \geq (1 - \varepsilon/10)(1 + \varepsilon/9)d \geq d$. Analogously, if $d_u \leq (1 - \varepsilon/9)d$, then $\hat{d}_u \leq (1 + \varepsilon/10)(1 - \varepsilon/9)d \leq d$. Thus, $\{u|d_u \geq d(1 + \varepsilon/9)\} \subseteq \{u|\hat{d}_u \geq d\} \subseteq \{u|d_u \geq d(1 - \varepsilon/9)\}$. □

6.3 The Main Result and SADDLES

We begin by stating the main result, and explaining how heavy tails lead to sublinear algorithms. Note that D refers to a set of degrees, for which we desire an approximation to $N(d)$.

Theorem 6.3.1. *There exists an algorithm* SADDLES *with the following properties. Let c be a sufficiently large constant. Fix any $\varepsilon > 0, \delta > 0$. Suppose that the parameters of* SADDLES *satisfy the following conditions: $r \geq c\varepsilon^{-2}n/h$, $q \geq c\varepsilon^{-2}m/z^2$, $\ell \geq c\log(n/\delta)$, $\tau \geq c\varepsilon^{-2}$.*

Then with probability at least $1 - \delta$, for all $d \in D$, SADDLES *outputs an $(\varepsilon, \varepsilon)$-approximation of $N(d)$.*

The expected number of queries made depends on the model, and is independent of the size of D.

- *SM:* $O((n/h + m/z^2)(\varepsilon^{-2}\log(n/\delta)))$.
- *HDM:* $O((m/z)(\varepsilon^{-4}\log^2(n/\delta)))$.

Observe how a larger h and z-index lead to smaller running times. Ignoring constant factors and assuming $m = O(n)$, asymptotically increasing h and z-indices lead to sublinear algorithms.

We now describe the algorithm itself. The main innovation in SADDLES comes through the use of a recent theoretical technique to simulate edge samples through vertex samples [90, 91]. The sampling of edges occurs through two stages. In the first stage, the algorithm samples a set of r vertices and sets up a distribution over the sampled vertices such that any edge adjacent to a sampled vertex may be sampled with uniform probability. In the second stage, it samples q edges from this distribution.

For each edge, we compute a weight based on the degrees of its vertices and generate our estimate by averaging these weights. Additionally, we use vertex

sampling to estimate the head of the distribution. Straightforward Chernoff bound arguments can be used to determine when to use the vertex sampling over the edge sampling method.

The same algorithmic structure is used for the Standard Model and the Hidden Degrees Model. The only difference is the use the algorithm of Corollary 6.2.6 to estimate degrees in the HDM, while the degrees are directly available in the Standard Model.

The core theoretical bound: The central technical bound deals with the properties of each individual estimate $\widetilde{N}(d)[t]$.

Theorem 6.3.2. *Suppose* $r \geq c\varepsilon^{-2}n/h$, $q \geq c\varepsilon^{-2}m/z^2$, $\tau = c\varepsilon^{-2}$. *Then, for all* $d \in D$, *with probability* $\geq 5/6$, $\widetilde{N}(d)[t] \in [(1-\varepsilon/2)N((1+\varepsilon/2)d), (1+\varepsilon/2)N((1-\varepsilon/2)d]$.

The proof of this theorem is the main part of our analysis, which appears in the next section. Theorem 6.3.1 can be derived from this theorem, as we show next.

Proof. (of Theorem 6.3.1) First, let us prove the error/accuracy bound. For a fixed $d \in D$ and $t \leq \ell$, Theorem 6.3.2 asserts that we get an accurate estimate with probability $\geq 5/6$. Among the ℓ independent invocations, the probability that more than $\ell/3$ values of $\widetilde{N}(d)[t]$ lie outside $[(1-\varepsilon/2)N((1+\varepsilon/2)d), (1+\varepsilon/2)N((1-\varepsilon/2)d]$ is at most $\exp(-\ell/100)$ (by the Chernoff bound of Theorem 6.2.1). By the choice of $\ell \geq c\log(n/\delta)$, the probability is at most δ/n. Thus, with probability $> 1 - \delta/n$, the median of $\widetilde{N}(d)[t]$ gives an $(\varepsilon, \varepsilon)$ estimate of $N(d)$. By a union bound over all (at most n) $d \in D$, the total probability of error over any d is at most δ.

Algorithm 18: SADDLES(D, r, q, ℓ, τ)
Inputs:
D: set of degrees for which N(d) is to be computed
r: budget for vertex samples
q: budget for edge samples
ℓ: boosting parameter
τ: cutoff for vertex sampling
Output:
$\{N'(d)\}$: estimated $\{N(d)\}$

For $t = 1, \ldots, \ell$:
    For $i = 1, \ldots, r$:
        Select u.a.r. vertex v and add it to multiset R.
        In HDM, call DEG(v) to get $\hat{d}_v$. In SM, set $\hat{d}_v$ to d_v.
        For $d \in D$:
            If $\hat{d}_v \geq d$, set $X_{id} = 1$. Else, $X_{id} = 0$.
    Let $\hat{d}_R = \sum_{v \in R} \hat{d}_v$ and $\mathcal{D}$ denote the distribution over R where $v \in R$ is selected with probability $\hat{d}_v / \hat{d}_R$.
    For $i = 1, \ldots, q$:
        Sample $v \sim \mathcal{D}$.
        Pick u.a.r. neighbor u of v.
        In HDM, call DEG(u) to get $\hat{d}_u$. In SM, set $\hat{d}_u$ to d_u.
        For $d \in D$:
            If $\hat{d}_u \geq d$, set $Y_{id} = 1/\hat{d}_u$. Else, set $Y_{id} = 0$.
    For $d \in D$:
        If $\sum_{i \leq r} X_{id} \geq \tau$:
            $\widetilde{N}(d)[t] = \frac{n}{r} \sum_{i \leq r} X_{id}$.
        else $\widetilde{N}(d)[t] = \frac{n}{r} \cdot \frac{\hat{d}_R}{q} \sum_{i \leq q} Y_{id}$.
For $d \in D$:
    $N'(d) = median\{\widetilde{N}(d)\}$
Return $\{N'(d)\}$

Now for the query complexity. The overall algorithm is the same for both models, involving multiple invocations of SADDLES. The only difference is in DEG, which is trivial when degree queries are allowed. For the Standard Model, the number of graph queries made for a single invocation of SADDLES is simply $O(\ell(r+q)) = O(\varepsilon^{-2}(n/h + m/z^2)\log(n/\delta))$.

For the Hidden Degrees Model, we have to account for the overhead of Corollary 6.2.6 for each degree estimated. The number of queries for a single call to DEG(d) is $O(\varepsilon^{-2}\sqrt{d_v}\log n)$. The total overhead of all calls in Step 4 is $\mathbf{E}[\sum_{v\in R}\sqrt{d_v}(\varepsilon^{-2}\log n)]$. By linearity of expectation, this is $O((\varepsilon^{-2}\log n)r\mathbf{E}[\sqrt{d_v}]$, where the expectation is over a uniform random vertex. We can bound $r\mathbf{E}[\sqrt{d_v}]$ $\leq r\mathbf{E}[d_v] = O(\varepsilon^{-2}n(m/n)/h) = O(\varepsilon^{-2}n/h)$.

The total overhead of all calls in Step 11 requires more care. Note that when DEG(v) is called multiple times for a fixed v, the subsequent calls require no further queries. (This is because the output of the first call can be stored.) We partition the vertices into two sets $S_0 = \{v|d_v \leq z^2\}$ and $S_1 = \{v|d_v > z^2\}$. The total query cost of queries to S_0 is at most $O(qz) = O((\varepsilon^{-2}\log n)m/z)$. For the total cost to S_1, we directly bound by (ignoring the $\varepsilon^{-2}\log n$ factor) $\sum_{v\in S_1}\sqrt{d_v} = \sum_{v\in S_1} d_v/\sqrt{d_v} \leq z^{-1}\sum_v d_v = O(m/z)$. All in all, the total query complexity is $O((\varepsilon^{-4}\log^2 n)(n/h+m/z))$. Since $m \geq n$ and $z \leq h$, we can simplify to $O((\varepsilon^{-4}\log^2 n)(m/z))$. □

6.4 Analysis of SADDLES

We now prove Theorem 6.3.2. There are a number of intermediate claims towards that. We will fix $d \in D$ and a choice of t. Abusing notation, we use $\widetilde{N}(d)$ to refer to $\widetilde{N}(d)[t]$. The estimate of Step 16 can be analyzed with a direct Chernoff bound.

Claim 6.4.1. *The following holds with probability* $> 9/10$. *If* SADDLES(r, q) *outputs an estimate in Step 16 for a given* d, *then* $\widetilde{N}(d) \in (1 \pm \varepsilon/10)\widehat{N_\Lambda}(d)$. *If it does not output in Step 16, then* $\widehat{N_\Lambda}(d) < (2c/\varepsilon^2)(n/r)$.

Proof. Each X_i is an iid Bernoulli random variable, with success probability precisely $\widehat{N_\Lambda}(d)/n$. We split into two cases.

Case 1: $\widehat{N_\Lambda}(d) \geq (c/10\varepsilon^2)(n/r)$. By the Chernoff bound of Theorem 6.2.1, $\Pr[|\sum_{i \leq r} X_i - r\widehat{N_\Lambda}(d)/n| \geq (\varepsilon/10)(r\widehat{N_\Lambda}(d)/n)] \leq 2\exp(-(\varepsilon^2/100)(r\widehat{N_\Lambda}(d)/n) \leq 1/100$.

Case 2: $\widehat{N_\Lambda}(d) \leq (c/10\varepsilon^2)(n/r)$. Note that $\mathbf{E}[\sum_{i \leq r} X_i] \leq c/10\varepsilon^2 \leq (c/\varepsilon^2)/2e$. By the upper tail bound of Theorem 6.2.1, $\Pr[\sum_{i \leq r} X_i \geq c/\varepsilon^2] < 1/100$.

Thus, with probability at least 99/100, if an estimate is output in Step 16, $\widehat{N_\Lambda}(d) > (c/10\varepsilon^2)(n/r)$. By the first case, with probability at least 99/100, $\widetilde{N}(d)$ is a $(1 + \varepsilon/10)$-estimate for $\widehat{N_\Lambda}(d)$. A union bound completes the first part.

Furthermore, if $\widehat{N_\Lambda}(d) \geq (2c/\varepsilon^2)(n/r)$, then with probability at least 99/100, $\sum_{i \leq r} X_i \geq (1 - \varepsilon/10)r\widehat{N_\Lambda}(d)/n \geq c/\varepsilon^2 = \tau$. A union bound proves (the contrapositive of) the second part. □

We define weights of *ordered* edges. The weight only depends on the second member in the pair, but allows for a more convenient analysis. The weight of $\langle v, u \rangle$ is the random variable Y_i of Step 13.

Definition 6.4.2. *The* d*-weight of an ordered edge* $\langle v, u \rangle$ *for a given* Λ *(the randomness of DEG) is defined as follows. We set* $\mathrm{wt}_{\Lambda,d}(\langle v, u \rangle)$ *to be* $1/\hat{d}_u$ *if* $\hat{d}_u \geq d$, *and zero otherwise. For vertex* v, $\mathrm{wt}_{\Lambda,d}(v) = \sum_{u \in \Gamma(v)} \mathrm{wt}_{\Lambda,d}(\langle v, u \rangle)$.

The utility of the weight definition is captured by the following claim. The total weight is an approximation of $\widetilde{N}(d)$, and thus, we can analyze how well SADDLES approximates the total weight.

Claim 6.4.3. *If Λ is good, $\sum_{v\in V} \mathrm{wt}_{\Lambda,d}(v) \in (1\pm\varepsilon/9)\widehat{N_\Lambda}(d)$.*

Proof.

$$\begin{aligned}\sum_{v\in V} \mathrm{wt}_{\Lambda,d}(v) &= \sum_{v\in V}\sum_{u\in\Gamma(v)} \mathbb{1}_{\hat{d}_u\geq d}/\hat{d}_u \\ &= \sum_{u:\hat{d}_u\geq d}\sum_{v\in\Gamma(u)} 1/\hat{d}_u = \sum_{u:\hat{d}_u\geq d} d_u/\hat{d}_u \end{aligned} \tag{6.1}$$

Since Λ is good, $\forall u, \hat{d}_u \in (1\pm\varepsilon/10)d_u$, and $d_u/\hat{d}_u \in (1\pm\varepsilon/9)$. Applying in (**??**), $\sum_{v\in V} \mathrm{wt}_{\Lambda,d}(v) \in (1\pm\varepsilon/9)\widehat{N_\Lambda}(d)$. □

We come to an important lemma, that shows that the weight of the random subset R (chosen in Step 3) is well-concentrated. This is proven using a Chernoff bound, but we need to bound the maximum possible weight to get a good bound on $r = |R|$.

Lemma 6.4.4. *Fix any good Λ and d. Suppose $r \geq c\varepsilon^{-2}n/d$. With probability at least 9/10, $\sum_{v\in R} \mathrm{wt}_{\Lambda,d}(v) \in (1\pm\varepsilon/8)(r/n)\widehat{N_\Lambda}(d)$.*

Proof. Let $\mathrm{wt}(R)$ denote $\sum_{v\in R} \mathrm{wt}_{\Lambda,d}(v)$. By linearity of expectation, $\mathbf{E}[\mathrm{wt}(R)] = (r/n)\cdot \sum_{v\in V} \mathrm{wt}_{\Lambda,d}(v) \geq (r/2n)\widehat{N_\Lambda}(d)$. To apply the Chernoff bound, we need to bound the maximum weight of a vertex. For good Λ, the weight $\mathrm{wt}_{\Lambda,d}$ of any ordered pair is at most $1/(1-\varepsilon/10)d \leq 2/d$. The number of neighbors of v such that $\hat{d}_u \geq d$ is at most $\widehat{N_\Lambda}(d)$. Thus, $\mathrm{wt}_{\Lambda,d}(v) \leq 2\widehat{N_\Lambda}(d)/d$.

By the Chernoff bound of Theorem 6.2.1 and setting $r \geq c\varepsilon^{-2}n/d$,

$$\begin{aligned}&Pr\left[|\mathrm{wt}(R) - \mathbf{E}[\mathrm{wt}(R)]| > (\varepsilon/20)\mathbf{E}[\mathrm{wt}(R)]\right] \\ &< 2\exp\left(-\frac{\varepsilon^2\cdot(c\varepsilon^{-2}n/d)\cdot(\widehat{N_\Lambda}(d)/2n)}{3\cdot 20^2\cdot 2\widehat{N_\Lambda}(d)/d}\right) \leq 1/10\end{aligned}$$

With probability at least 9/10, $\mathrm{wt}(R) \in (1\pm\varepsilon/20)\mathbf{E}[\mathrm{wt}(R)]$. By the arguments

given above, $\mathbf{E}[\mathrm{wt}(R)] \in (1 \pm \varepsilon/9)(r/n)\widehat{N_\Lambda}(d)$. We combine to complete the proof. □

Now, we determine the number of edge samples required to estimate the weight $\mathrm{wt}_{\Lambda,d}(R)$.

Lemma 6.4.5. *Let $\widetilde{N}(d)$ be as defined in Step 17 of* SADDLES. *Assume Λ is good, $r \geq c\varepsilon^{-2}n/d$, and $q \geq c\varepsilon^{-2}m/(d\widehat{N_\Lambda}(d))$. Then, with probability $> 7/8$, $\widetilde{N}(d) \in (1 \pm \varepsilon/4)\widehat{N_\Lambda}(d)$.*

Proof. We define the random set R selected in Step 3 to be *sound* if the following hold. (1) $\mathrm{wt}(R) = \sum_{v\in R} \mathrm{wt}_{\Lambda,d}(v) \in (1 \pm \varepsilon/8)(r/n)\widehat{N_\Lambda}(d)$ and (2) $\sum_{v\in R} d_v \leq 100r(2m/n)$. By Lemma 6.4.4, the first holds with probability $> 9/10$. Observe that $\mathbf{E}[\sum_{v\in R} d_v] = r(2m/n)$, since $2m/n$ is the average degree. By the Markov bound, the second holds with probability $> 99/100$. By the union bound, R is sound with probability at least $1 - (1/10 + 1/100) > 8/9$.

Fix a sound R. Recall Y_i from Step 13. The expectation of $Y_i|R$ is $\sum_{v\in R} \Pr[v \text{ is selected}] \cdot \sum_{u\in\Gamma(v)} \Pr[u \text{ is selected}]\mathrm{wt}_{\Lambda,d}(\langle v,u\rangle)$. We plug in the probability values, and observe that for good Λ, for all v, $\hat{d}_v/d_v \in (1 \pm \varepsilon/10)$.

$$\begin{aligned}
\mathbf{E}[Y_i|R] &= \sum_{v\in R}(\hat{d}_v/\hat{d}_R) \sum_{u\in\Gamma(v)} (1/d_v)\mathrm{wt}_{\Lambda,d}(\langle v,u\rangle) \\
&= (1/\hat{d}_R)\sum_{v\in R}(\hat{d}_v/d_v) \sum_{u\in\Gamma(v)} \mathrm{wt}_{\Lambda,d}(\langle v,u\rangle) \\
&\in (1 \pm \varepsilon/10)(1/\hat{d}_R)\sum_{v\in R}\sum_{u\in\Gamma(v)} \mathrm{wt}_{\Lambda,d}(\langle v,u\rangle) \\
&\in (1 \pm \varepsilon/10)(\mathrm{wt}(R)/\hat{d}_R) \qquad (6.2)
\end{aligned}$$

Note that $\widetilde{N}(d) = (n/r)(\hat{d}_R/q)\sum_{i\leq q} Y_i$ and $(n/r)(\hat{d}_R/q)\mathbf{E}[\sum_{i\leq q} Y_i|R] \in (1 \pm \varepsilon/10)(n/r)\mathrm{wt}(R)$. Since R is sound, the latter is in $(1 \pm \varepsilon/4)\widehat{N_\Lambda}(d)$. Also, note

that

$$\mathbf{E}[Y_i|R] = \mathbf{E}[Y_1|R] \geq \frac{q\mathrm{wt}(R)}{2\hat{d}_R} \geq \frac{(r/n)\widehat{N_\Lambda}(d)}{4(100r(2m/n)} = \frac{\widehat{N_\Lambda}(d)}{800m} \tag{6.3}$$

By linearity of expectation, $\mathbf{E}[\sum_{i\leq q} Y_i|R] = q\mathbf{E}[Y_1|R]$. Observe that $Y_i \leq 1/d$. We can apply the Chernoff bound of Theorem 6.2.1 to the iid random variables $(Y_i|R)$.

$$\begin{aligned} & \Pr[|\sum_i Y_i - \mathbf{E}[\sum_i Y_i]| > (\varepsilon/100)\mathbf{E}[\sum_i Y_i]|R] \\ \leq\ & 2\exp\Big(-\frac{\varepsilon^2}{3\cdot 100^2}\cdot d\cdot q\mathbf{E}[Y_1|R]\Big) \end{aligned} \tag{6.4}$$

We use (**??**) to bound the (positive) term in the exponent is at least

$$\frac{\varepsilon^2}{3\cdot 100^2}\cdot\frac{c\varepsilon^{-2}m}{\widehat{N_\Lambda}(d)}\cdot\frac{\widehat{N_\Lambda}(d)}{800m} \geq 10.$$

Thus, if R is sound, the following bound holds with probability at least 0.99. We also apply (**??**).

$$\begin{aligned} \widehat{N_\Lambda}(d) &= (n/r)(\hat{d}_R/q)\sum_{i=1}^{q} Y_i \\ &\in (1\pm\varepsilon/100)(n/r)(\hat{d}_R/q)q\mathbf{E}[Y_i|R] \\ &\in (1\pm\varepsilon/100)(1\pm\varepsilon/10)(n/r)\mathrm{wt}(R) \in (1\pm\varepsilon/4)\widetilde{N}(d) \end{aligned}$$

The probability that R is sound is at least 8/9. A union bound completes the proof. □

The bounds on r and q in Lemma 6.4.5 depend on the degree d. We now bring in the h and z-indices to derive bounds that hold for all d. We also remove the conditioning over a good Λ.

Proof. (of Theorem 6.3.2) We will first assume that Λ is good. By Claim 6.2.9,

$\widehat{N_\Lambda}(d) \in [N((1+\varepsilon/9)d, N((1-\varepsilon/9)d)]$.

Suppose $\widehat{N_\Lambda}(d) = 0$, so there are no vertices with $\hat{d}_v \geq d$. By the bound above, $N((1+\varepsilon/9)d) = 0$, implying that $N((1+\varepsilon/2)d) = 0$. Furthermore $\widetilde{N}(d) = 0$, since the random variables X_i and Y_i in SADDLES can never be non-zero. Thus, $\widetilde{N}(d) = N((1+\varepsilon/2)d)$, completing the proof.

We now assume that $\widehat{N_\Lambda}(d) > 0$. We split into two cases, depending on whether Step 16 outputs or not. By Claim 6.4.1, with probability $> 9/10$, if Step 16 outputs, then $\widetilde{N}(d) \in (1 \pm \varepsilon/9)\widehat{N_\Lambda}(d)$. By combining these bounds, the desired bound on $\widetilde{N}(d)$ holds with probability $> 9/10$, conditioned on a good Λ.

Henceforth, we focus on the case that Step 16 does not output. By Claim 6.4.1, $\widehat{N_\Lambda}(d) < 2c\varepsilon^{-2}(n/r)$. By the choice of r and Claim 6.2.9, $\widehat{N_\Lambda}((1+\varepsilon/9)d) < h$. By the characterization of h of Lemma 6.2.2, $z^2 \leq \max(\widehat{N_\Lambda}((1+\varepsilon/9)d), (1+\varepsilon/9)d) = (1+\varepsilon/9)d$. This implies that $r \geq c\varepsilon^{-2}n/d$. By the definition of z, $z^2 \leq N(\min(d_{max}, (1+\varepsilon/9)d)) \cdot \min(d_{max}, (1+\varepsilon/9)d)$. By the Claim 6.2.9 bound in the first paragraph, $\widehat{N_\Lambda}(d) \geq N((1+\varepsilon/9)d)$. Since $\widehat{N_\Lambda}(d) > 0$, $\widehat{N_\Lambda}(d) \geq \widehat{N_\Lambda}(d_{max})$. Thus, $z^2 \leq \widehat{N_\Lambda}(d) \cdot (1+\varepsilon/9)d$. and hence, $m \leq c\varepsilon^{-2}m/(d\widehat{N_\Lambda}(d))$. The parameters satisfy the conditions in Lemma 6.4.5. With probability $> 7/8$, $\widetilde{N}(d) \in (1 \pm \varepsilon/4)\widehat{N_\Lambda}(d)$, and by Claim 6.2.9, $\widetilde{N}(d)$ has the desired accuracy.

All in all, assuming Λ is good, with probability at least $7/8$, $\widetilde{N}(d)$ has the desired accuracy. The conditioning on a good Λ is removed by Claim 6.2.8 to complete the proof. □

6.5 Experimental Results

We implemented our algorithm in C++ and performed our experiments on a MacBook Pro laptop with 2.7 GHz Intel Core i5 with 8 GB RAM. We performed our experiments on a collection of graphs from SNAP [159], including social

networks, web networks, and infrastructure networks. The graphs typically have millions of edges, with the largest having more than 100M edges. Basic properties of these graphs are presented in Table 6.1. We ignore direction and treat all edges as undirected edges.

6.5.1 Implementation Details

For the HDM, we explicitly describe the procedure DEG(v), which estimates the degree of a given vertex (v). In the algorithm DEG, a "pair-wise

Algorithm 19: DEG(v)

1 (Initialize $S = \emptyset$.) Repeatedly add u.a.r. vertex to S, until the number of pair-wise collisions is at least $k = 25$.
2 Output $\binom{|S|}{2}/k$ as estimate $\hat{d}_v$.

collision" refers to a pair of neighbor samples that yield the same vertex. The expected number of pair-wise collisions is $\binom{|S|}{2}/d_v$. We simply reverse engineer that inequality to get the estimate $\hat{d}_v$. Ron and Tsur essentially prove that this estimate has low variance [204].

Setting the parameter values. The boosting parameter ℓ is simply set to 1. (In some sense, we only introduced the median boosting for the theoretical union bound. In practice, convergence is much more rapid that predicted by the Chernoff bound.)

The threshold τ is set to 100. The parameters r and q are chosen to be typically around $0.005n$. These are not "sublinear" per se, but are an order of magnitude smaller than the queries made in existing graph sampling results (more discussion in next section).

We set $D = \{\lfloor 1.1^i \rfloor\}$, since that gives a sufficiently fine-grained approximation at all scales of the degree distribution.

Table 6.1: Graph properties: #vertices (n), #edges (m), maximum degree, h-index and z-index. The last column indicates the median number of samples over 100 runs (as a percentage of m) required by SADDLES under HDM, with $r + q = 0.01n$.

graph	#vertices	#edges	max. degree	avg. degree	H-index	Z-index	Perc. edge samples for HDM
loc-gowalla	1.97E+05	9.50E+05	14730	4.8	275	101	7.0
web-Stanford	2.82E+05	1.99E+06	38625	7.0	427	148	6.4
com-youtube	1.13E+06	2.99E+06	28754	2.6	547	121	11.7
web-Google	8.76E+05	4.32E+06	6332	4.9	419	73	6.2
web-BerkStan	6.85E+05	6.65E+06	84230	9.7	707	220	5.5
wiki-Talk	2.39E+06	9.32E+06	100029	3.9	1055	180	8.5
as-skitter	1.70E+06	1.11E+07	35455	6.5	982	184	6.7
cit-Patents	3.77E+06	1.65E+07	793	4.3	237	28	5.6
com-lj	4.00E+06	3.47E+07	14815	8.6	810	114	4.7
soc-LiveJournal1	4.85E+06	8.57E+07	20333	17.7	989	124	2.4
com-orkut	3.07E+06	1.17E+08	33313	38.1	1638	172	2.0

Code for all experiments is available here[1].

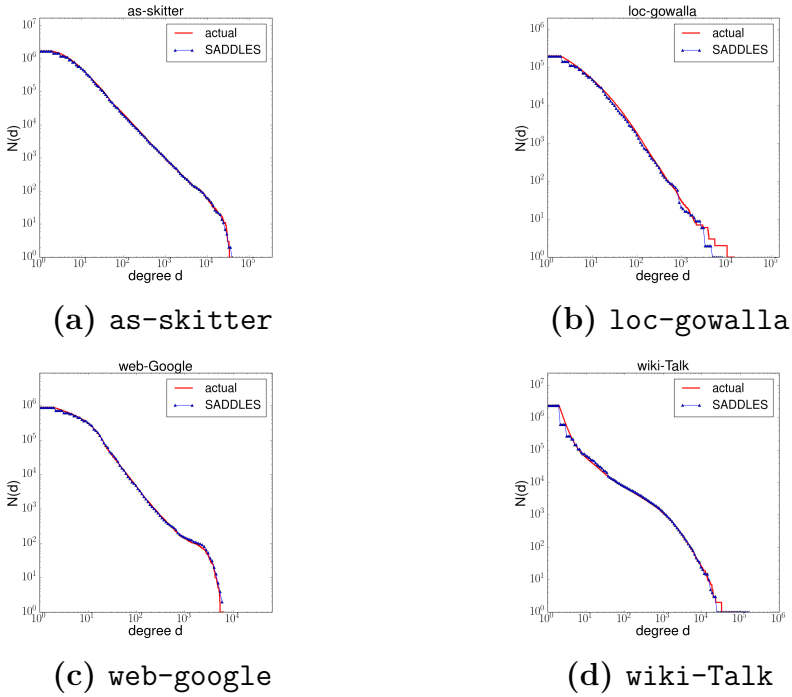
6.5.2 Evaluation of SADDLES

Accuracy over all graphs: We run SADDLES with the parameters discussed above for a variety of graphs. Because of space considerations, we do not show results for all graphs in this version. (We discovered the results to be consistent among all our experiments.) Fig. 6.1 show the results for the SM for some graphs in Tab. 6.1. For all these runs, we set $r + q$ to be 1% of the number of vertices in the graph. Note that the sample size of SADDLES in the SM is exactly $r + q$. For the HDM, we show results in Fig. 6.2. Again, we set $r + q$ to be 1%, though the number of edges sampled (due to invocations of DEG(v)) varies quite a bit. The required number of samples are provided in Tab. 6.1. Note that the number of edges sampled is well within 10% of the total, except for the `com-youtube` graph.

Visually, we can see that the estimates are accurate for all degrees, in all graphs, for both models. This is despite there being sufficient irregular behavior

[1] https://sjain12@bitbucket.org/sjain12/saddles.git

in $N(d)$. Note that the shape of the various ccdhs are different and none of them form an obvious straight line. Nonetheless, SADDLES captures the distribution almost perfectly in all cases by observing 1% of the vertices.



(a) `as-skitter` **(b)** `loc-gowalla`

(c) `web-google` **(d)** `wiki-Talk`

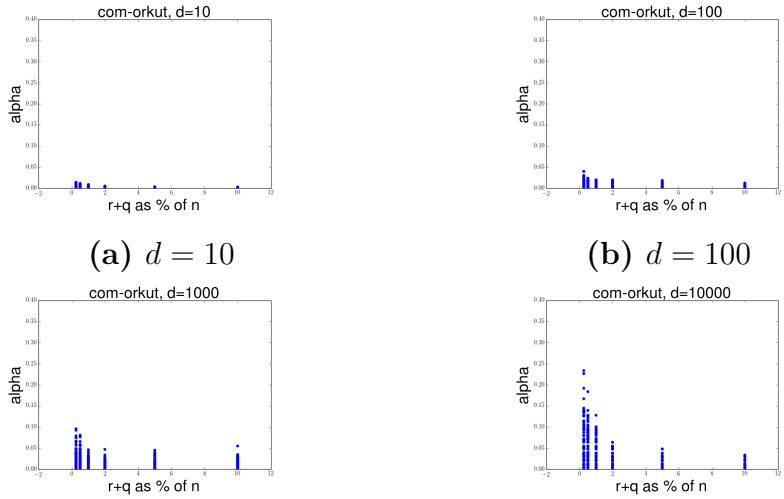
Figure 6.2: The result of runs of SADDLES on a variety of graphs, for the HDM. We set $r+q$ to be 1% of the number of vertices, for all graphs. The actual number of edges sampled varies, and is given in Tab. 6.1.

Convergence: To demonstrate convergence, we fix the graph `com-orkut`, and run SADDLES only for the degrees $10, 100$, and 1000. For each choice of degree, we vary the total number of samples $r+q$. (We set $r=q$ in all runs.) Finally, for each setting of $r+q$, we perform 100 independent runs of SADDLES.

For each such run, we compute an error parameter α. Suppose the output of a run is M, for degree d. The value of α is the smallest value of ϵ, such that $M \in [(1-\epsilon)N((1+\epsilon)d), (1+\epsilon)N((1-\epsilon)d)]$. (It is the smallest ϵ such that M is an (ϵ,ϵ)-approximation of $N(d)$.)

Fig. 6.3 shows the spread of α, for the 100 runs, for each choice of $r+q$.

Observe how the spread decreases as $r+q$ goes to 10%. In all cases, the values of α decay to less than 0.05. We notice that convergence is much faster for $d = 10$. This is because $N(10)$ is quite large, and SADDLES is using vertex sampling to estimate the value.



(a) $d = 10$ **(b)** $d = 100$

(c) $d = 1000$ **(d)** $d = 10000$

Figure 6.3: Convergence of SADDLES: We plot the values of the error parameter α (as defined in §6.5.2) for 100 runs at increasing values of $r + q$. We have a different plot for $d = 10, 100, 1000, 10000$ to show the convergence at varying portions of the ccdh.

Large value of h and z-index on real graphs: The h and z-index of all graphs is given in Tab. 6.1. Observe how they are typically in the hundreds. Note that the average degree is typically an order of magnitude smaller than these indices. Thus, a sample size of $n/h + m/z^2$ (as given by Theorem 6.3.1, ignoring constants) is significantly sublinear. This is consistent with our choice of $r + q = n/100$ leading to accurate estimates for the ccdh.

6.5.3 Comparison with previous work

There are several graph sampling algorithms that have been discussed in [13, 88, 158, 160, 196, 202, 270]. In all of these methods we collect the vertices and scale their counts appropriately to get the estimated ccdh. We describe these methods below in more detail, and discuss our implementation of the method.

- Vertex Sampling (VS, also called egocentric sampling) [10, 88, 158, 160, 196, 202]: In this algorithm, we sample vertices u.a.r. and scale the ccdh obtained appropriately, to get an estimate for the ccdh of the entire graph.
- Edge Sampling (ES) [10, 88, 158, 160, 196, 202]: This algorithm samples edges u.a.r. and includes one or both end points in the sampled network. Note that this does *not* fall into the SM. In our implementation we pick a random end point.
- Random walk with jump (RWJ) [10, 88, 160, 196, 202]: We start a random walk at a vertex selected u.a.r. and collect all vertices encountered on the path in our sampled network. At any point, with a constant probability (0.15, based on previous results) we jump to another u.a.r. vertex.
- One Wave Snowball (OWS) [10, 88, 158]: Snowball sampling starts with some vertices selected u.a.r. and crawls the network until a network of the desired size is sampled. In our implementation, we usually stop at the first level since that accumulates enough vertices.
- Forest fire (FF) [10, 88, 160]: This method generates random sub-crawls of the network. A vertex is picked u.a.r. and randomly selects a subset of its neighbors (according to a geometric distribution). The process is repeated from every selected vertex until it ends. It is then repeated from another u.a.r. vertex.

We run all these algorithms on the `amazon0601`, `web-Google`, `cit-Patents`, and `com-orkut` networks. To make fair comparisons, we run each method until

it selects 1% of the vertices. The comparisons are shown in Fig. 6.1. Observe how none of the methods come close to accurately measuring the ccdh. (This is consistent with previous work, where typically 10-20% of the vertices are sampled for results.) Naive vertex sampling is accurate at the head of the distribution, but completely misses the tail. Except for vertex sampling, all other algorithms are biased towards the tail. Crawls find high degree vertices with disproportionately higher probability, and overestimate the tail.

Note that our implementations of FF, OWS, RWJ assume access to u.a.r. vertices. Variants of these algorithms can be used in situations where we only have access to seed vertices, however, one would typically have to sample many more edges to deal with larger correlation among the vertices obtained through the random walks. Despite this extra capability to sample u.a.r. vertices in our implementation of these algorithms, they show significant errors, particularly in the tail of the distribution.

Inverse method of Zhang et al [270]: An important result of estimating degree distributions is that of Zhang et al [270], that explicitly points out the bias problems in various sampling methods. They propose a bias correction method by solving a constrained, penalized weighted least-squares problem on the sampled degree distribution. We apply this method for the sampling methods demonstrated in their paper, namely VS, OWS, and IN (sample vertices u.a.r. and only retain edges between sampled vertices). We show results in Fig. 6.1, again with a sample size of 1% of the vertices. Observe that no method gets even close to estimating the ccdh accurately, even after debiasing. Fundamentally, these methods require significantly more samples to generate accurate estimates.

The running time and memory requirements of this method grow superlinearly with the maximum degree in the graph. The largest graph processed by [270] has

a few hundred thousand edges, which is on the smaller side of graphs in Tab. 6.1. SADDLES processes a graph with more than 100M edges in less than a minute, while our attempts to run the [270] algorithm on this graph did not terminate in hours.

6.6 Future work

We presented an algorithm, SADDLES that is able to estimate the degree distribution of real world graphs by sampling only 1% of the graph which is an order of magnitude less then most other methods. While this is a first step towards understanding what can be accomplished when we do not have access to the whole graph, the ability to be able to sample vertices u.a.r. is arguably, too powerful an assumption for most real world setups. It would be interesting to see what can be accomplished when we do not have access to u.a.r. vertices but only to some seed vertices in the graph.

Chapter 7

Conclusion and future work

Clique counting is an important but challenging problem with many applications. In this thesis, we presented several different tools and techniques to address this problem. All existing methods were too slow and did not scale for cliques larger than 5 vertices. As a first step towards more efficient algorithms, we presented TuránShadow - a randomized algorithm based on Turán's theorem that was able to count cliques upto 10 vertices and was much faster and more accurate in estimating the number of cliques than existing methods.

While TuránShadow provided the first significant improvement in clique counting, it had it's limitations. Specifically, it required us to store the entire shadow - a set of small subgraphs of the given graph. For large graphs with billions of edges, the amount of space required to store the shadow can become prohibitively large. The heuristic introduced in Inverse-TS gives an estimator based on TuránShadow that is able to sample and estimate the count of cliques without requiring to store the entire shadow. This results in significant savings in space.

We also explored how the efficient mining and counting of cliques can help us in mining and counting other clique-like structures called near-cliques. Generic

pattern-finding algorithms fail to utilize the clique like structure of near-cliques and Inverse-TS gives orders of magnitude improvement over them. Using Inverse-TS we were able to estimate the counts of near-cliques (cliques missing atmost 2 edges) of upto 10 vertices in several graphs with millions of vertices and edges which was not possible using previous methods.

Finally, in a completely different approach to clique counting, we used pivoting to massively cut down the search space of cliques resulting in a tremendous improvement over all existing clique counting methods. For a graph with 100 million edges, Pivoter was able to count *all* $k-$cliques (largest k for this graph was 61) exactly in less than 2 hours, where other methods (including TuránShadow) was unable to count even in days. Remarkably, it was also able to give per-edge and per-vertex counts which no other existing methods have been able to get for graphs with millions of edges. Using Pivoter, local as well as global clique counting has become feasible for a lot of graphs for which it was infeasible before.

One of the salient features of these techniques is that they did not involve the use of parallelism. It would be useful and interesting to see how we can further push the boundaries of what is feasible using parallelism and specialized hardware like that of MapReduce. While these paradigms require different thinking when designing algorithms for them, we believe that the ideas presented here can be applied in the parallel setup and that will definitely lead to significant improvements. This would be especially useful for addressing clique counting for graphs like `com-lj` whose size and structure make clique counting difficult even for Pivoter.

It would be also useful to investigate why these algorithms do not show worst-case behaviour on real-world graphs and infact run efficiently, and what

that implies about the structure of these graphs. The fact that these algorithms run efficiently on real-world graphs is very telling and understanding this behavior can possibly lead to faster algorithms for other tasks on graphs.

Over the years, there has been a trend of looking at bigger patterns and incorporating the information of higher order structures in techniques for visualizing and analyzing graphs, and we hope that the tools provided here will open the doors for greater use of cliques and near-cliques, leading to more useful and meaningful insights.

Bibliography

[1] http://en.wikipedia.org/wiki/Scale-free_network.

[2] *Proceedings IEEE CSE'09, 12th IEEE International Conference on Computational Science and Engineering, August 29-31, 2009, Vancouver, BC, Canada*. IEEE Computer Society, 2009. 153

[3] *30th IEEE International Conference on Distributed Computing Systems Workshops (ICDCS 2010 Workshops), 21-25 June 2010, Genova, Italy*. IEEE Computer Society, 2010. 148

[4] http://facebook.com/press/info.php?statistics, 2012.

[5] D. Achlioptas, A. Clauset, D. Kempe, and C. Moore. On the bias of traceroute sampling: Or, power-law degree distributions in regular graphs. *Journal of the ACM*, 56(4), 2009. 99

[6] Alex T Adai, Shailesh V Date, Shannon Wieland, and Edward M Marcotte. Lgl: creating a map of protein function with an algorithm for visualizing very large biological networks. *Journal of molecular biology*, 340(1):179–190, 2004. 36

[7] Foto N. Afrati, Dimitris Fotakis, and Jeffrey D. Ullman. Enumerating subgraph instances using map-reduce. In *International Conference on Data Engineering (ICDE)*, pages 62–73, 2013. 17

[8] N. Ahmed, J. Neville, and R. Kompella. Space-efficient sampling from social activity streams. In *SIGKDD BigMine*, pages 1–8, 2012. 100, 108

[9] Nesreen K Ahmed, Nick Duffield, Jennifer Neville, and Ramana Kompella. Graph sample and hold: A framework for big-graph analytics. In *SIGKDD*, pages 1446–1455. ACM, ACM, 2014.

[10] Nesreen K Ahmed, Jennifer Neville, and Ramana Kompella. Network sampling: From static to streaming graphs. *TKDD*, 8(2):7, 2014. 100, 102, 104, 106, 108, 128

[11] Nesreen K. Ahmed, Jennifer Neville, Ryan A. Rossi, and Nick Duffield. Efficient graphlet counting for large networks. In *Proceedings of International Conference on Data Mining (ICDM)*, 2015. 3, 17, 30

[12] Nesreen K. Ahmed, Jennifer Neville, Ryan A. Rossi, and Nick Duffield. Efficient graphlet counting for large networks. 2015.

[13] N.K. Ahmed, J. Neville, and R. Kompella. Reconsidering the foundations of network sampling. In *WIN 10*, 2010. 100, 108, 128

[14] Kook J. Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Principles of Database Systems*, pages 5–14, 2012.

[15] Kook Jin Ahn, Graham Cormode, Sudipto Guha, Andrew McGregor, and Anthony Wirth. Correlation clustering in data streams. In *ICML*, 2015.

[16] E. A. Akkoyunlu. The enumeration of maximal cliques of large graphs. *SIAM J. Comput.*, 2:1–6, 1973. 17, 77

[17] Sinan G. Aksoy, Tamara G. Kolda, and Ali Pinar. Measuring and modeling bipartite graphs with community structure. *Journal of Complex Networks*, 2017. to appear. 99

[18] Noga Alon, T. Kaufman, and Michael Krivelevich. Testing triangle-freeness in general graphs. In *Proceedings of the 17th Annual Symposium on Discrete Algorithms (SODA)*, 2006.

[19] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[20] Noga Alon, Raphy Yuster, and Uri Zwick. Color-coding: A new method for finding simple paths, cycles and other small subgraphs within large graphs. In *Symposium on the Theory of Computing (STOC)*, pages 326–335, 1994. 9, 16, 17, 18, 46, 76

[21] J Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat, and Alessandro Vespignani. Large scale networks fingerprinting and visualization using the k-core decomposition. In *Advances in neural information processing systems*, pages 41–50, 2006. 4, 39

[22] R. Andersen and K. Chellapilla. Finding dense subgraphs with size bounds. In *Workshop on Algorithms and Models for the Web-Graph (WAW)*, pages 25–37, 2009. 25, 46

[23] L. Arge, M.T. Goodrich, and N. Sitchinava. Parallel external memory graph algorithms. In *Parallel Distributed Processing Symposium (IPDPS)*, pages 1 –11, april 2010.

[24] S. M. Arifuzzaman, M. Khan, and M. Marathe. Patric: A parallel algorithm for counting triangles and computing clustering coefficients in massive networks. Technical Report 12-042, NDSSL, 2012.

[25] Yuichi Asahiro, Refael Hassin, and Kazuo Iwama. Complexity of finding dense subgraphs. *Discrete Applied Mathematics*, 121(1-3):15–26, 2002.

[26] Haim Avron. Counting triangles in large graphs using randomized matrix trace estimation. In *KDD workshon Large Scale Data Mining*, 2010.

[27] László Babai, editor. *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*. ACM, 2004. 138

[28] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Symposium of Discrete Algorith,s*, pages 623–632, 2002.

[29] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286:509–512, October 1999. 99, 104

[30] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *KDD'08*, pages 16–24, 2008. 2, 16, 46

[31] A. Benson, D. F. Gleich, and J. Leskovec. Higher-order organization of complex networks. *Science*, 353(6295):163–166, 2016. 3, 72, 75

[32] Radu Berinde, Piotr Indyk, Graham Cormode, and Martin J. Strauss. Space-optimal heavy hitters with strong error bounds. *ACM Trans. Database Syst.*, 35(4):26, 2010.

[33] J. Berry, L. Fosvedt, D. Nordman, C. A. Phillips, and A. G. Wilson. Listing triangles in expected linear time on power law graphs with exponent at least $\frac{7}{3}$. Technical Report SAND2010-4474c, Sandia National Laboratories, 2011.

[34] Jonathan W. Berry, Luke K. Fostvedt, Daniel J. Nordman, Cynthia A. Phillips, C. Seshadhri, and Alyson G. Wilson. Why do simple algorithms for triangle enumeration work in the real world? In *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science*, ITCS '14, pages 225–234, New York, NY, USA, 2014. ACM.

[35] Jonathan W. Berry, Bruce Hendrickson, Randall A. LaViolette, and Cynthia A. Phillips. Tolerating the community detection resolution limit with edge weighting. *Phys. Rev. E*, 83:056119, May 2011. 2, 75

[36] J.W. Berry, B. Hendrickson, S. Kahan, and P. Konecny. Software and algorithms for graph queries on multithreaded architectures. In *Parallel and Distributed Processing Symposium (IPDPS)*, pages 1 –14, march 2007.

[37] Nadja Betzler, René van Bevern, Michael R. Fellows, Christian Komusiewicz, and Rolf Niedermeier. Parameterized algorithmics for finding connected motifs in biological networks. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 8(5):1296–1308, 2011. 18, 76

[38] M. Bhuiyan, M. Rahman, M. Rahman, and M. Al Hasan. Guise: Uniform sampling of graphlets for large graph analysis. In *Proceedings of International Conference on Data Mining*, pages 91–100, 2012. 18

[39] M. A. Bhuiyan and M. Al Hasan. Mirage: An iterative mapreduce based frequent subgraph mining algorithm. Technical report, arXiv, 2013. http://arxiv.org/pdf/1307.5894.pdf.

[40] Mansurul A Bhuiyan, Mahmudur Rahman, Mahmuda Rahman, and Mohammad Al Hasan. Guise: Uniform sampling of graphlets for large graph analysis. In *2012 IEEE 12th International Conference on Data Mining*, pages 91–100. IEEE, 2012.

[41] Etienne Birmel. Detecting local network motifs. *Electron. J. Statist.*, 6:908–933, 2012.

[42] I. Bordino, D. Donata, A. Gionis, and S. Leonardi. Mining large networks with subgraph counting. In *Proceedings of International Conference on Data Mining*, pages 737–742, 2008. 46

[43] Jacqueline Bourdeau, Jim Hendler, Roger Nkambou, Ian Horrocks, and Ben Y. Zhao, editors. *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11 - 15, 2016*. ACM, 2016. 142

[44] Vladimir Braverman and Rafail Ostrovsky. Approximating large frequency moments with pick-and-drop sampling. In *APPROX*, pages 42–57. Springer, 2013.

[45] Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. Motif counting beyond five nodes. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 12(4):48, 2018. 46

[46] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. *Computer Networks*, 33:309–320, 2000. 99, 104

[47] Coen Bron and Joep Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Commun. ACM*, 16(9):575–577, September 1973. 17

[48] Coen Bron and Joep Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Commun. ACM*, 16(9):575–577, September 1973. 73, 77, 78

[49] Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In *Principles of Database Systems*, pages 253–262, 2006.

[50] R. Burt. Structural holes and good ideas. *American Journal of Sociology*, 110(2):349–399, 2004. 2, 17

[51] Venkatesan T. Chakaravarthy, Michael Kapralov, Prakash Murali, Fabrizio Petrini, Xinyu Que, Yogish Sabharwal, and Baruch Schieber. Subgraph counting: Color coding beyond trees. *CoRR*, abs/1602.04478, 2016.

[52] Deepayan Chakrabarti and Christos Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Computing Surveys*, 38(1), 2006. 99

[53] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-MAT: A recursive model for graph mining. In *SDM '04*, pages 442–446, 2004.

[54] Dhruva R. Chakrabarti, Prithviraj Banerjee, Hans-J. Boehm, Pramod G. Joisha, and Robert S. Schreiber. The runtime abort graph and its application to software transactional memory optimization. In *International Symposium on Code Generation and Optimization*, pages 42–53, 2011.

[55] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Automata, Languages and Programming*, pages 693–703. Springer, 2002.

[56] Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Linear FPT reductions and computational lower bounds. In Babai [27], pages 212–221. 3, 14, 17, 30, 76

[57] Jie Chen and Yousef Saad. Dense subgraph extraction with application to community detection. *IEEE Transactions on knowledge and data engineering*, 24(7):1216–1230, 2010. 4, 39

[58] H. Chernoff. A measure of asymptotic efficiency for test of a hypothesis based on the sum of observations. *Annals of mathematical statistics*, 23:493–507, 1952.

[59] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23(4):493–507, 1952.

[60] Yi-Jen Chiang, Michael T. Goodrich, Edward F. Grove, Roberto Tamassia, Darren Erik Vengroff, and Jeffrey Scott Vitter. External-memory graph algorithms. In *Symposium on Discrete Algorithms (SODA)*, pages 139–149, 1995.

[61] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14:210–223, 1985. 9, 12, 17, 18, 19, 25, 29, 46, 66, 76, 81, 89

[62] F. Chierichetti, A. Dasgupta, R. Kumar, S. Lattanzi, and T. Sarlos. On sampling nodes in a network. In *World Wide Web (WWW)*, 2016. 102, 110

[63] S. Chu and J. Cheng. Triangle listing in massive networks and its applications. In *Knowledge Data and Discovery (KDD)*, pages 672–680, 2011.

[64] A. Clauset and C. Moore. Accuracy and scaling phenomena in internet mapping. *Phys. Rev. Lett.*, 94:018701, 2005. 99

[65] A. Clauset, C. R. Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Review*, 51(4):661–703, 2009. 101, 104, 109

[66] Jonathan Cohen. Graph twiddling in a MapReduce world. *Computing in Science & Engineering*, 11:29–41, 2009.

[67] R. Cohen, K. Erez, D. ben Avraham, and S. Havlin. Resilience of the internet to random breakdowns. *Phys. Rev. Lett.*, 85(4626âĂŞ8), 2000. 99

[68] J. Coleman. Social capital in the creation of human capital. *American Journal of Sociology*, 94:S95–S120, 1988.

[69] Robert L Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *ACM SIGGRAPH computer graphics*, volume 18, pages 137–145. ACM, 1984.

[70] Graham Cormode and Donatella Firmani. A unifying framework for ℓ_0-sampling algorithms. *Distributed and Parallel Databases*, 32(3):315–335, 2014.

[71] Graham Cormode and Marios Hadjieleftheriou. Finding frequent items in data streams. *VLDB*, 1(2):1530–1541, 2008.

[72] Graham Cormode, Flip Korn, S Muthukrishnan, and Divesh Srivastava. Diamond in the rough: Finding hierarchical heavy hitters in multi-dimensional data. In *ACM SIGMOD*, pages 155–166. ACM, 2004.

[73] Graham Cormode and S Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[74] Graham Cormode and S. Muthukrishnan. Space efficient mining of multigraph streams. In *SIGACT-SIGMOD-SIGART*, pages 271–282, 2005.

[75] Graham Cormode and S Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. *TODS*, 30(1):249–278, 2005.

[76] Graham Cormode, S Muthukrishnan, and Irina Rozenbaum. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In *VLDB*, pages 25–36. VLDB Endowment, 2005.

[77] Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 210–223. ACM, 2017. 46

[78] Maximilien Danisch, Oana Balalau, and Mauro Sozio. Listing k-cliques in sparse real-world graphs. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 589–598. International World Wide Web Conferences Steering Committee, 2018.

[79] Maximilien Danisch, Oana Denisa Balalau, and Mauro Sozio. Listing k-cliques in sparse real-world graphs. In *World Wide Web (WWW)*, pages 589–598, 2018. 12, 33, 46, 71, 72, 74, 76, 78, 81, 89, 90, 92, 94, 96

[80] Apurba Das, Michael Svendsen, and Srikanta Tirthapura. Change-sensitive algorithms for maintaining maximal cliques in a dynamic graph. *CoRR*, abs/1601.06311, 2016.

[81] A. Dasgupta, R. Kumar, and T. Sarlos. On estimating the average degree. In *World Wide Web (WWW)*, pages 795–806, 2014. 102, 110

[82] Erik D Demaine, Alejandro López-Ortiz, and J Ian Munro. Frequency estimation of internet packet streams with limited space. In *Proc. of ESA 2002*, pages 348–360. Springer, 2002.

[83] D. Dubhashi and A. Panconesi. *Concentration of Measure for the Analysis of Randomised Algorithms*. Cambridge University Press, 2012. 111

[84] Devdatt Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009. 23, 49

[85] N. Durak, T.G. Kolda, A. Pinar, and C. Seshadhri. A scalable null model for directed graphs matching all degree distributions: In, out, and reciprocal. In *Network Science Workshop (NSW), 2013 IEEE 2nd*, pages 23–30, April 2013. 99

[86] N. Durak, A. Pinar, T. G. Kolda, and C. Seshadhri. Degree relations of triangles in real-world networks and graph models. In *CIKM'12*, 2012.

[87] David A. Van Dyk and Max Welling, editors. *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, Florida, USA, April 16-18, 2009*, volume 5 of *JMLR Proceedings*. JMLR.org, 2009. 152

[88] Peter Ebbes, Zan Huang, Arvind Rangaswamy, Hari P Thadakamalla, and ORGB Unit. Sampling large-scale social networks: Insights from simulated networks. In *18th Annual Workshop on Information Technologies and Systems, Paris, France*, 2008. 100, 102, 104, 106, 128

[89] Jean-Pierre Eckmann and Elisha Moses. Curvature of co-links uncovers hidden thematic layers in the World Wide Web. *Proceedings of the National Academy of Sciences (PNAS)*, 99(9):5825–5829, 2002.

[90] T. Eden, A. Levi, D. Ron, and C. Seshadhri. Approximately counting triangles in sublinear time. pages 614–633, 2015. 101, 103, 107, 109, 115

[91] T. Eden, D. Ron, and C. Seshadhri. Sublinear time estimation of degree distribution moments: The degeneracy connection. In GRS11, editor, *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 614–633, 2017. 101, 103, 107, 109, 115

[92] Talya Eden, Shweta Jain, Ali Pinar, Dana Ron, and C Seshadhri. Provable and practical approximations for the degree distribution using sublinear graph samples. In *Proceedings of the 2018 World Wide Web Conference*, pages 449–458, 2018. 7

[93] Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theoretical Computer Science*, 326(1):57 – 67, 2004.

[94] Ethan R. Elenberg, Karthikeyan Shanmugam, Michael Borokhovich, and Alexandros G. Dimakis. Beyond triangles: A distributed framework for estimating 3-profiles of large graphs. In *Knowledge Data and Discovery (KDD)*, pages 229–238, 2015.

[95] Ethan R. Elenberg, Karthikeyan Shanmugam, Michael Borokhovich, and Alexandros G. Dimakis. Distributed estimation of graph 4-profiles. In Bourdeau et al. [43], pages 483–493. 11

[96] Ethan R Elenberg, Karthikeyan Shanmugam, Michael Borokhovich, and Alexandros G Dimakis. Distributed estimation of graph 4-profiles. In *Proceedings of the 25th International Conference on World Wide Web*, pages 483–493. International World Wide Web Conferences Steering Committee, 2016. 46

[97] David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *International Symposium on Algorithms and Computation*, pages 403–414. Springer, 2010. 17, 77

[98] David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. *ACM Journal of Experimental Algorithmics*, 18, 2013. 73, 77, 80

[99] P. Erdős. On the number of complete subgraphs and circuits contained in graphs. *Casopis Pest. Mat.*, 94:290–296, 1969. 15, 18, 21

[100] G. Fagiolo. Clustering in complex directed networks. *Phys. Rev. E*, 76:026107, Aug 2007.

[101] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999. 99, 104

[102] K. Faust. A puzzle concerning triads in social networks: Graph constraints and the triad census. *Social Networks*, 32(3):221–233, 2010. 2

[103] U. Feige. On sums of independent random variables with unbounded variance and estimating the average degree in a graph. *SIAM Journal on Computing*, 35(4):964–984, 2006. 109, 110

[104] W. Feller. *An Introduction to probability theory and applications: Vol I.* John Wiley and Sons, 3rd edition, 1968.

[105] Irene Finocchi, Marco Finocchi, and Emanuele G. Fusco. Clique counting in mapreduce: Algorithms and experiments. *ACM Journal of Experimental Algorithmics*, 20, 2015. vi, x, 14, 15, 17, 18, 25, 31, 32, 46, 50, 71, 72, 73, 76, 78, 81

[106] Peter Floderus, MirosÅĆaw Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Induced subgraph isomorphism: Are some patterns substantially easier than others? *Theoretical Computer Science*, 605:119 – 128, 2015.

[107] Eugene Fratkin, Brian T Naughton, Douglas L Brutlag, and Serafim Batzoglou. Motifcut: regulatory motifs finding with maximum density subgraphs. *Bioinformatics*, 22(14):e150–e157, 2006. 4, 39

[108] Christos Giatsidis, Fragkiskos Malliaros, Dimitrios M Thilikos, and Michalis Vazirgiannis. Corecluster: A degeneracy based graph clustering framework. In *IAAA: Innovative Applications of Artificial Intelligence*, 2014. 25

[109] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

[110] David F. Gleich and C. Seshadhri. Vertex neighborhoods, low conductance cuts, and good seeds for local community methods. In *Knowledge Data and Discovery (KDD)*, 2012.

[111] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, pages 302–343, 2002. 101

[112] O. Goldreich and D. Ron. Approximating average parameters of graphs. 32(4):473–493, 2008. 101, 109, 110

[113] M. Gonen, D. Ron, and Y. Shavitt. Counting stars and other small subgraphs in sublinear-time. 25(3):1365–1411, 2011. 101, 109

[114] Mira Gonen, Dana Ron, Udi Weinsberg, and Avishai Wool. Finding a dense-core in jellyfish graphs. *Computer Networks*, 52(15):2831–2841, 2008. 109

[115] Mira Gonen and Yuval Shavitt. Approximating the number of network motifs. *Internet Mathematics*, 6(3):349–372, 2009.

[116] Donald P Greenberg, Kenneth E Torrance, Peter Shirley, James Arvo, Eric Lafortune, James A Ferwerda, Bruce Walter, Ben Trumbore, Sumanta Pattanaik, and Sing-Choong Foo. A framework for realistic image synthesis. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 477–494. ACM Press/Addison-Wesley Publishing Co., 1997.

[117] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016. 36

[118] Sudipto Guha, Andrew McGregor, and David Tench. Vertex and hyperedge connectivity in dynamic graph streams. In *PODS*, pages 241–247, 2015.

[119] David Hales and Stefano Arteconi. Motifs in evolving cooperative networks look like protein structure networks. *NHM*, 3(2):239–249, 2008.

[120] Guyue Han and Harish Sethu. Waddling random walk: Fast and accurate mining of motif statistics in large graphs. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pages 181–190. IEEE, 2016. 46

[121] Robert A. Hanneman and Mark Riddle. *Introduction to social network methods*. University of California, Riverside, 2005. http://faculty.ucr.edu/~hanneman/nettext/. 3, 17

[122] Paul S Heckbert. Adaptive radiosity textures for bidirectional ray tracing. *ACM SIGGRAPH Computer Graphics*, 24(4):145–154, 1990.

[123] J. E. Hirsch. An index to quantify an individual's scientific research output. *Proceedings of the National Academy of Sciences*, 102(46):16569âĂŞ16572, 2005. 105

[124] Tomaž Hočevar and Janez Demšar. Combinatorial algorithm for counting small induced graphs and orbits. *PloS one*, 12(2):e0171428, 2017. 46

[125] W. Hoeffding. Probability inequalities for sums of bounded random variables. *J. American Statistical Association*, 58:13–30, 1963.

[126] P. Holland and S. Leinhardt. A method for detecting structure in sociometric data. *American Journal of Sociology*, 76:492–513, 1970. 2, 16, 46

[127] F. Hormozdiari, P. Berenbrink, N. Przulj, and S. Cenk Sahinalp. Not all scale-free networks are born equal: The role of the seed graph in ppi network evolution. *PLoS Computational Biology*, 118, 2007. 2, 18

[128] Joseph J. Pfeiffer III, Timothy La Fond, Sebastián Moreno, and Jennifer Neville. Fast generation of large scale social networks while incorporating transitive closures. In *International Conference on Privacy, Security, Risk and Trust (PASSAT)*, pages 154–165, 2012. 2

[129] Piotr Indyk and David P. Woodruff. Optimal approximations of the frequency moments of data streams. In *STOC*, pages 202–208, 2005.

[130] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph dat. In *Proceedings of Pacific-Asia KDD*, pages 13–23, 2000.

[131] Shalev Itzkovitz, Reuven Levitt, Nadav Kashtan, Ron Milo, Michael Itzkovitz, and Uri Alon. Coarse-graining and self-dissimilarity of complex networks. 71(016127), January 2005.

[132] Matthew O. Jackson. *Social and Economic Networks*. Princeton University Press, 2010. 3, 17

[133] Matthew O. Jackson, Tomas Rodriguez-Barraquer, and Xu Tan. Social capital and social quilts: Network patterns of favor exchange. *American Economic Review*, 102(5):1857?1897, 2012. 17

[134] Shweta Jain and C Seshadhri. A fast and provable method for estimating clique counts using turán's theorem. In *Proceedings of the 26th International Conference on World Wide Web*, pages 441–449. International World Wide Web Conferences Steering Committee, 2017. 6, 71, 72, 73, 76, 78, 81, 90, 91, 96

[135] Shweta Jain and C Seshadhri. A fast and provable method for estimating clique counts using turán's theorem. In *Proceedings of the 26th International Conference on World Wide Web*, pages 441–449. International World Wide Web Conferences Steering Committee, 2017. 46, 69

[136] Shweta Jain and C Seshadhri. The power of pivoting for exact clique counting. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 268–276, 2020. 7

[137] M. Jha, C. Seshadhri, and A. Pinar. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *World Wide Web (WWW)*, pages 495–505, 2015. 3, 14, 17, 18, 30, 46, 76

[138] Madhav Jha, C Seshadhri, and Ali Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. In *SIGKDD*, pages 589–597. ACM, 2013.

[139] Madhav Jha, C. Seshadhri, and Ali Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. In *Knowledge Data and Discovery (KDD)*, 2013.

[140] H. Jowhari and M. Ghodsi. New streaming algorithms for counting triangles in graphs. In *Computing and Combinatorics Conference (COCOON)*, pages 710–716, 2005.

[141] Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *SIGMOD-SIGACT-SIGART, PODS*, pages 49–58, 2011.

[142] James T Kajiya. The rendering equation. In *ACM Siggraph Computer Graphics*, volume 20, pages 143–150. ACM, 1986.

[143] Daniel M Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. Counting arbitrary subgraphs in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 598–609. Springer, 2012. 46

[144] Daniel M. Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. Counting arbitrary subgraphs in data streams. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 598–609, 2012.

[145] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *Prof. of PODS*, pages 41–5, 2010.

[146] Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Approximating matching size from random streams. In *SODA*, pages 734–751, 2014.

[147] Richard M Karp, Scott Shenker, and Christos H Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *TODS*, 28(1):51–55, 2003.

[148] Ton Kloks, Dieter Kratsch, and Haiko MÃijller. Finding and counting small induced subgraphs efficiently. *Information Processing Letters*, 74(3):115 – 121, 2000.

[149] Tamara G. Kolda, Ali Pinar, Todd Plantenga, C. Seshadhri, and Christine Task. Counting triangles in massive graphs with MapReduce. *SIAM Journal of Scientific Computing*, 2013. To appear.

[150] M. N. Kolountzakis, G. L. Miller, R. Peng, and C. Tsourakakis. Efficient triangle counting in large graphs via degree-based vertex partitioning. In *WAW'10*, 2010.

[151] Flip Korn, S Muthukrishnan, and Yihua Wu. Modeling skew in data streams. In *SIGMOD*, pages 181–192. ACM, 2006.

[152] Flip Korn, S. Muthukrishnan, and Yihua Wu. Modeling skew in data streams. In *SIGMOD*, pages 181–192. ACM, 2006.

[153] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. Trawling the web for emerging cyber-communities. *Computer networks*, 31(11-16):1481–1493, 1999. 4, 39

[154] Jérôme Kunegis. The koblenz network collection. http://konect.uni-koblenz.de, 2015.

[155] H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? In *WWW*, 2010.

[156] A. Lakhina, J. Byers, M. Crovella, and P. Xie. Sampling biases in IP topology measurements. In *Proceedings of INFOCOMM*, volume 1, pages 332–341, 2003. 99

[157] M. Latapy. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theoretical Computer Science*, 407:458–473, 2008.

[158] Sang Hoon Lee, Pan-Jun Kim, and Hawoong Jeong. Statistical properties of sampled networks. *Physical Review E*, 73(1):016102, 2006. 100, 102, 104, 106, 128

[159] Jure Leskovec. Snap stanford network analysis project. `http://snap.standord.edu`, 2015. 30, 45, 62, 123

[160] Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *Knowledge Data and Discovery (KDD)*, pages 631–636. ACM, 2006. 100, 102, 104, 106, 108, 128

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

[162] Ying Li, Bing Liu, and Sunita Sarawagi, editors. *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*. ACM, 2008.

[163] Guimei Liu and Limsoon Wong. Effective pruning techniques for mining quasi-cliques. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 33–49. Springer, 2008. 4, 39

[164] Thomas Locher. Finding heavy distinct hitters in data streams. In *Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures*, pages 299–308. ACM, 2011.

[165] L. Lovász and M. Simonovits. *On the number of complete subgraphs of a graph II*, pages 459–495. Birkhäuser Basel, Basel, 1983.

[166] Zhenqi Lu, Johan Wahlström, and Arye Nehorai. Community detection in complex networks via clique conductance. *Scientific reports*, 8(1):5982, 2018. 3, 72

[167] A. S. Maiya and T. Y. Berger-Wolf. Benefits of bias: Towards better characterization of network sampling. In *Knowledge Data and Discovery (KDD)*, pages 105–113, 2011. 100, 108

[168] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *VLDB*, pages 346–357. VLDB Endowment, 2002.

[169] George Manoussakis. The clique problem on inductive k-independent graphs. *CoRR*, abs/1410.3302, 2014.

[170] George Manoussakis. Listing all maximal cliques in sparse graphs in optimal time. *CoRR*, abs/1501.01819, 2015.

[171] Dror Marcus and Yuval Shavitt. Efficient counting of network motifs. In *ICDCS Workshops* [3], pages 92–98. 3

[172] Shawn Martin, W Michael Brown, and Brian N Wylie. Dr. l: Distributed recursive (graph) layout. Technical report, Sandia National Laboratories, 2007. 36

[173] David W Matula and Leland L Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM (JACM)*, 30(3):417–427, 1983. 9, 25, 27, 56, 81, 86

[174] Andrew McGregor. Finding graph matchings in data streams. In *APPROX and RANDOM*, pages 170–181, 2005.

[175] Andrew McGregor. Graph stream algorithms: A survey. *SIGMOD*, 43(1):9–20, 2014. 110

[176] M. Meiss, F. Menczer, S. Fortunato, A. Flammini, and A. Vespignani. Ranking web sites with real user traffic. In *WSDM*, pages 65–75. ACM, 2008.

[177] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *Database Theory-ICDT 2005*, pages 398–412. Springer, 2005.

[178] T. Milenkovic and N. Przulj. Uncovering Biological Network Function via Graphlet Degree Signatures. *arXiv*, q-bio.MN, January 2008. 2

[179] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002. 2, 16, 45

[180] Ron Milo, Shalev Itzkovitz, Nadav Kashtan, Reuven Levitt, Shai Shen-Orr, Inbal Ayzenshtat, Michal Sheffer, and Uri Alon. Superfamilies of evolved and designed networks. *Science*, 303(5663):1538–1542, 2004.

[181] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *IMC'07*, pages 29–42. ACM, 2007.

[182] M. Mitzenmacher. A brief history of generative models for power law and lognormal distributions. *Internet Mathematics*, 1(2):226–251, 2003. 99

[183] Michael Mitzenmacher, Jakub Pachocki, Richard Peng, Charalampos E. Tsourakakis, and Shen Chen Xu. Scalable large near-clique detection in large-scale networks via sampling. In *SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 815–824, 2015. 3

[184] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003. 99

[185] M. E. J. Newman, S. Strogatz, and D. Watts. Random graphs with arbitrary degree distributions and their applications. *Physical Review E*, 64:026118, 2001. 99

[186] R. Pagh and C. Tsourakakis. Colorful triangle counting and a mapreduce implementation. *Information Processing Letters*, 112:277–281, 2012.

[187] Ashwin Paranjape, Austin R Benson, and Jure Leskovec. Motifs in temporal networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 601–610. ACM, 2017. 46

[188] Panos M Pardalos and Steffen Rebennack. Computational challenges with cliques, quasi-cliques and clique partitions in graphs. In *International Symposium on Experimental Algorithms*, pages 13–22. Springer, 2010.

[189] Panos M. Pardalos and Steffen Rebennack, editors. *Experimental Algorithms - 10th International Symposium, SEA 2011, Kolimpari, Chania, Crete, Greece, May 5-7, 2011. Proceedings*, volume 6630 of *Lecture Notes in Computer Science*. Springer, 2011.

[190] Jeffrey Pattillo, Nataly Youssef, and Sergiy Butenko. Clique relaxation models in social network analysis. In *Handbook of Optimization in Complex Networks*, pages 143–162. Springer, 2012. 4, 39

[191] A. Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. Counting and sampling triangles from a graph stream. *PVLDB*, 6(14):1870–1881, 2013.

[192] D. Pennock, G. Flake, S. Lawrence, E. Glover, and C. L. Giles. Winners don't take all: Characterizing the competition for links on the web. *Proceedings of the National Academy of Sciences*, 99(8):5207–5211, 2002. 99

[193] T. Petermann and P. Rios. Exploration of scale-free networks. *European Physical Journal B*, 38:201–204, 2004. 99

[194] Ali Pinar, C Seshadhri, and Vaidyanathan Vishal. Escape: Efficiently counting all 5-vertex subgraphs. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1431–1440. International World Wide Web Conferences Steering Committee, 2017. 3

[195] Ali Pinar, C Seshadhri, and Vaidyanathan Vishal. Escape: Efficiently counting all 5-vertex subgraphs. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1431–1440. International World Wide Web Conferences Steering Committee, 2017. 46

[196] Ali Pinar, Sucheta Soundarajan, Tina Eliassi-Rad, and Brian Gallagher. Maxoutprobe: An algorithm for increasing the size of partially observed networks. Technical report, Sandia National Laboratories (SNL-CA), Livermore, CA (United States), 2015. 102, 104, 106, 128

[197] Todd Plantenga. Inexact subgraph isomorphism in mapreduce. *Journal of Parallel and Distributed Computing*, (0), 2012.

[198] Alejandro Portes. Social capital: Its origins and applications in modern sociology. *Annual Review of Sociology*, 24(1):1–24, 1998.

[199] Nataša Pržulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):e177–e183, 2007. 45, 95

[200] Natasa Przulj, Derek G. Corneil, and Igor Jurisica. Modeling interactome: scale-free or geometric?. *Bioinformatics*, 20(18):3508–3515, 2004. 2

[201] M. Rahman, M. A. Bhuiyan, and M. Al Hasan. Graft: An efficient graphlet counting method for large graph analysis. *IEEE Transactions on Knowledge and Data Engineering*, PP(99), 2014. 11, 16, 18

[202] Bruno Ribeiro and Don Towsley. On the estimation accuracy of degree distributions from graph sampling. In *Annual Conference on Decision and Control (CDC)*, pages 5240–5247. IEEE, 2012. 100, 102, 104, 106, 108, 128

[203] Dana Ron. Algorithmic and analysis techniques in property testing. *Foundations and Trends in Theoretical Computer Science*, 5(2):73–205, 2010. 109

[204] Dana Ron and Gilad Tsur. The power of an example: Hidden set size approximation using group queries and conditional sampling. *ACM Transactions on Computation Theory*, 8(4):15:1–15:19, 2016. 107, 113, 124

[205] Ryan A Rossi, David F Gleich, and Assefaw H Gebremedhin. Parallel maximum clique algorithms with applications to network analysis. *SIAM Journal on Scientific Computing*, 37(5):C589–C616, 2015. 17, 46

[206] Rahmtin Rotabi, Krishna Kamath, Jon M. Kleinberg, and Aneesh Sharma. Detecting strong ties using network motifs. In *World Wide Web (WWW)*, pages 983–992, 2017. 75

[207] Alessandra Sala, Lili Cao, Christo Wilson, Robert Zablit, Haitao Zheng, and Ben Y. Zhao. Measurement-calibrated graph models for social network experiments. In *World Wide Web (WWW)*, pages 861–870, 2010. 2

[208] Ahmet Erdem Sariyüce, C. Seshadhri, Ali Pinar, and Ümit V. Çatalyürek. Finding the hierarchy of dense subgraphs using nucleus decompositions. pages 927–937, 2015. 3, 17, 25, 46, 72, 75

[209] Ahmet Erdem Sariyuce, C Seshadhri, Ali Pinar, and Umit V Catalyurek. Finding the hierarchy of dense subgraphs using nucleus decompositions. In *Proceedings of the 24th International Conference on World Wide Web*, pages 927–937. International World Wide Web Conferences Steering Committee, 2015. 4, 39

[210] Thomas Schank and Dorothea Wagner. Approximating clustering coefficient and transitivity. *Journal of Graph Algorithms and Applications*, 9:265–275, 2005.

[211] Thomas Schank and Dorothea Wagner. Finding, counting and listing all triangles in large graphs, an experimental study. In *Experimental and Efficient Algorithms*, pages 606–609. Springer Berlin / Heidelberg, 2005.

[212] Daniel Schwabe, Virgílio A. F. Almeida, Hartmut Glaser, Ricardo A. Baeza-Yates, and Sue B. Moon, editors. *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*. International World Wide Web Conferences Steering Committee / ACM, 2013.

[213] Stephen B. Seidman. Network structure and minimum degree. *Social Networks*, 5(3):269–287, 1983. 18, 25

[214] C. Seshadhri, Tamara G. Kolda, and Ali Pinar. Community structure and scale-free collections of Erdös-Rényi graphs. *Physical Review E*, 85(5):056109, May 2012. 2, 17, 46, 99

[215] C. Seshadhri, Ali Pinar, and Tamara G. Kolda. Fast triangle counting through wedge sampling. In *Proceedings of the SIAM Conference on Data Mining*, 2013. 14, 30

[216] C. Seshadhri, Ali Pinar, and Tamara G. Kolda. Wedge sampling for computing clustering coefficients and triangle counts on large graphs. *Statistical Analysis and Data Mining*, 7(4):294–307, 2014. 76

[217] C. Seshadhri and Srikanta Tirthapura. Scalable subgraph counting: The methods behind the madness: WWW 2019 tutorial. In *Proceedings of the Web Conference (WWW)*, 2019. 2, 75

[218] Nino Shervashidze, S. V. N. Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten M. Borgwardt. Efficient graphlet kernels for large graph comparison. pages 488–495, 2009. 2

[219] Nino Shervashidze, S. V. N. Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten M. Borgwardt. Efficient graphlet kernels for large graph comparison. In Dyk and Welling [87], pages 488–495.

[220] R. Sherwin. Introduction to the graph theory and structural balance approaches to international relations. *World Event/Interaction Survey*, 1971. https://apps.dtic.mil/dtic/tr/fulltext/u2/a080476.pdf.

[221] Françis X Sillion, James R Arvo, Stephen H Westin, and Donald P Greenberg. A global illumination solution for general reflectance distributions. In *ACM SIGGRAPH Computer Graphics*, volume 25, pages 187–196. ACM, 1991.

[222] Miguel EP Silva, Pedro Paredes, and Pedro Ribeiro. Network motifs detection using random networks with prescribed subgraph frequencies. In *Workshop on Complex Networks CompleNet*, pages 17–29. Springer, 2017. 46

[223] Olivia Simpson, C Seshadhri, and Andrew McGregor. Catching the head, tail, and everything in between: a streaming algorithm for the degree distribution. pages 979–984. IEEE, 2015. 110

[224] Ann Sizemore, Chad Giusti, and Danielle S. Bassett. Classification of weighted networks through mesoscale homological features. *Journal of Complex Networks*, 10.1093, 2016. 3, 17, 46, 75

[225] S. Son, A. Kang, H. Kim, T. Kwon, J. Park, and H. Kim. Analysis of context dependence in social interaction networks of a massively multiplayer online role-playing game. *PLoS ONE*, 7(4):e33918, 04 2012. 2

[226] Sucheta Soundarajan, Tina Eliassi-Rad, Brian Gallagher, and Ali Pinar. Maxreach: Reducing network incompleteness through node probes. pages 152–157, 2016. 109

[227] Sucheta Soundarajan, Tina Eliassi-Rad, Brian Gallagher, and Ali Pinar. ϵ - WGX: adaptive edge probing for enhancing incomplete networks. In *Web Science Conference*, pages 161–170, 2017. 109

[228] Alina Stoica and Christophe Prieur. Structure of neighborhoods in a large social network. In *CSE (4)* [2], pages 26–33.

[229] Michael PH Stumpf and Carsten Wiuf. Sampling properties of random graphs: the degree distribution. *Physical Review E*, 72(3):036118, 2005. 100, 108, 109

[230] Michael PH Stumpf, Carsten Wiuf, and Robert M May. Subnets of scale-free networks are not scale-free: sampling properties of networks. *Proceedings of the National Academy of Sciences of the United States of America*, 102(12):4221–4224, 2005.

[231] Siddharth Suri and Sergei Vassilvitskii. Counting triangles and the curse of the last reducer. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 607–614, New York, NY, USA, 2011. ACM.

[232] Siddharth Suri and Sergei Vassilvitskii. Counting triangles and the curse of the last reducer. In *World Wide Web (WWW)*, pages 607–614, 2011.

[233] M. Szell, R. Lambiotte, and S. Thurner. Multirelational organization of large-scale social networks in an online world. *Proceedings of the National Academy of Sciences*, 107:13636–13641, 2010.

[234] M. Szell and S. Thurner. Measuring social dynamics in a massive multiplayer online game. *Social Networks*, 32:313–329, 2010.

[235] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. Arnetminer: Extraction and mining of academic social networks. In *KDD'08*, pages 990–998, 2008. 67

[236] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. Arnetminer: Extraction and mining of academic social networks. In *KDD'08*, pages 990–998, 2008. 96

[237] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. Arnetminer: Extraction and mining of academic social networks. In *KDD'08*, pages 990–998, 2008.

[238] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. *The Worst-Case Time Complexity for Generating All Maximal Cliques*, pages 161–170. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. 17, 77, 80, 88

[239] Amanda L Traud, Peter J Mucha, and Mason A Porter. Social structure of facebook networks. *Physica A: Statistical Mechanics and its Applications*, 391(16):4165–4180, 2012. 36

[240] C. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. Tsiarli. Denser than the densest subgraph: Extracting optimal quasi-cliques with quality guarantees. In *Knowledge Data and Discovery (KDD)*, 2013. 25, 40, 46

[241] C. Tsourakakis, P. Drineas, E. Michelakis, I. Koutis, and C. Faloutsos. Spectral counting of triangles in power-law networks via element-wise sparsification. In *ASONAM'09*, pages 66–71, 2009. 10

[242] C. Tsourakakis, M. N. Kolountzakis, and G. Miller. Triangle sparsifiers. *J. Graph Algorithms and Applications*, 15:703–726, 2011. 10, 18

[243] C.E. Tsourakakis. Fast counting of triangles in large real networks without counting: Algorithms and laws. In *International Conference on Data Mining (ICDM)*, pages 608–617, 2008.

[244] Charalampos E. Tsourakakis. The k-clique densest subgraph problem. In *Proceedings of the Conference on World Wide Web WWW*, pages 1122–1132, 2015. 3, 17, 46, 72, 75

[245] Charalampos E. Tsourakakis, U. Kang, Gary L. Miller, and Christos Faloutsos. Doulion: counting triangles in massive graphs with a coin. In *Knowledge Data and Discovery (KDD)*, pages 837–846, 2009. 10, 14, 16, 18

[246] Charalampos E. Tsourakakis, Jakub Pachocki, and Michael Mitzenmacher. Scalable motif-aware graph clustering. In *World Wide Web (WWW)*, pages 1451–1460, 2017. 3, 72, 75

[247] Charalampos E. Tsourakakis, Jakub W. Pachocki, and Michael Mitzenmacher. Scalable motif-aware graph clustering. *CoRR*, abs/1606.06235, 2016. 3, 46

[248] Paul Turán. On an extremal problem in graph theory. *Mat. Fiz. Lapok*, 48(436-452):137, 1941. 15, 18, 21

[249] Johan Ugander, Lars Backstrom, and Jon M. Kleinberg. Subgraph frequencies: mapping the empirical and extremal geography of large graph collections. In *WWW*, pages 1307–1318, 2013. 2, 46, 75

[250] Virginia Vassilevska. Efficient algorithms for clique problems. *Information Processing Letters*, 109(4):254 – 257, 2009. 17, 76

[251] J. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[252] John R Wallace, Michael F Cohen, and Donald P Greenberg. *A two-pass solution to the rendering equation: A synthesis of ray tracing and radiosity methods*, volume 21. ACM, 1987.

[253] Pinghui Wang, John Lui, Bruno Ribeiro, Don Towsley, Junzhou Zhao, and Xiaohong Guan. Efficiently estimating motif statistics of large networks. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 9(2):8, 2014. 46

[254] Pinghui Wang, Junzhou Zhao, Xiangliang Zhang, Zhenguo Li, Jiefeng Cheng, John C. S. Lui, Don Towsley, Jing Tao, and Xiaohong Guan. MOSS-5: A fast method of approximating counts of 5-node graphlets in large graphs. 30(1):73–86, 2018. 76

[255] Pinghui Wang, Junzhou Zhao, Xiangliang Zhang, Zhenguo Li, Jiefeng Cheng, John CS Lui, Don Towsley, Jing Tao, and Xiaohong Guan. Moss-5: A fast method of approximating counts of 5-node graphlets in large graphs. *IEEE Transactions on Knowledge and Data Engineering*, 30(1):73–86, 2018. 46

[256] Gregory J Ward, Francis M Rubinstein, and Robert D Clear. A ray tracing solution for diffuse interreflection. *ACM SIGGRAPH Computer Graphics*, 22(4):85–92, 1988.

[257] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[258] D. Watts and S. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998. 2

[259] B. Welles, A. Van Devender, and N. Contractor. Is a friend a friend?: Investigating the structure of friendship networks in virtual worlds. In *CHI-EA'10*, pages 4027–4032, 2010. 2

[260] Sebastian Wernicke. Efficient detection of network motifs. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 3(4):347–359, 2006. 46

[261] Elisabeth Wong, Brittany Baur, Saad Quader, and Chun-Hsi Huang. Biological network motif detection: principles and practice. *Briefings in Bioinformatics*, 13(2):202–215, 2012.

[262] Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, ICDM '02, pages 721–, Washington, DC, USA, 2002. IEEE Computer Society.

[263] Hao Yin, Austin R. Benson, and Jure Leskovec. Higher-order clustering in networks. *Phys. Rev. E*, 97:052306, 2018. 75

[264] Hao Yin, Austin R. Benson, and Jure Leskovec. The local closure coefficient: A new perspective on network clustering. pages 303–311, 2019. 3, 72, 75

[265] Hao Yin, Austin R Benson, Jure Leskovec, and David F Gleich. Local higher-order graph clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 555–564. ACM, 2017. 4, 39, 46

[266] Jin-Hyun Yoon and Sung-Ryul Kim. Improved sampling for triangle counting with MapReduce. In *Convergence and Hybrid Information Technology*, volume 6935, pages 685–689. 2011.

[267] Haiyuan Yu, Alberto Paccanaro, Valery Trifonov, and Mark Gerstein. Predicting interactions in protein networks by completing defective cliques. *Bioinformatics*, 22(7):823–829, 2006. 4, 39

[268] Raphael Yuster. Finding and counting cliques and independent sets in r-uniform hypergraphs. *Information Processing Letters*, 99(4):130 – 134, 2006.

[269] Cyber situational awareness: The netops perspective. Available at http://www.disa.mil/News/Conferences-and-Events/DISA-Mission-Partner-Conference-2012/~/media/Files/DISA/News/Conference/2012/Cyber_Situational_Awareness_NetOps.pdf.

[270] Yaonan Zhang, Eric D Kolaczyk, and Bruce D Spencer. Estimating network degree distributions under sampling: An inverse problem, with applications to monitoring social media networks. *The Annals of Applied Statistics*, 9(1):166–199, 2015. ix, 100, 102, 104, 106, 108, 128, 129, 130

[271] Z. Zhao, G. Wang, A. Butt, M. Khan, V. S. Anil Kumar, and M. Marathe. Sahad: Subgraph analysis in massive networks using hadoop. In *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS)*, pages 390–401, 2012. 18, 76

[272] Petabyte scale data at facebook. available at http://www-conf.slac.stanford.edu/xldb2012/talks/xldb2012_wed_1105_DhrubaBorthakur.pdf.