Provably and Efficiently Approximating Near-cliques using the Turán Shadow: PEANUTS

Shweta Jain
University of California, Santa Cruz
Santa Cruz, CA, USA
sjain12@ucsc.edu

C. Seshadhri
University of California, Santa Cruz
Santa Cruz, CA
sesh@ucsc.edu

ABSTRACT

Clique and near-clique counts are important graph properties with applications in graph generation, graph modeling, graph analytics, community detection among others. They are the archetypal examples of dense subgraphs. While there are several different definitions of near-cliques, most of them share the attribute that they are cliques that are missing a small number of edges. Clique counting is itself considered a challenging problem. Counting near-cliques is significantly harder more so since the search space for near-cliques is orders of magnitude larger than that of cliques.

We give a formulation of a near-clique as a clique that is missing a constant number of edges. We exploit the fact that a near-clique contains a smaller clique, and use techniques for clique sampling to count near-cliques. This method allows us to count near-cliques with 1 or 2 missing edges, in graphs with tens of millions of edges. To the best of our knowledge, there was no known efficient method for this problem, and we obtain a $10x - 100x$ speedup over existing algorithms for counting near-cliques.

Our main technique is a space efficient adaptation of the Turán Shadow sampling approach, recently introduced by Jain and Seshadhri (WWW 2017). This approach constructs a large recursion tree (called the Turán Shadow) that represents cliques in a graph. We design a novel algorithm that builds an estimator for near-cliques, using a online, compact construction of the Turán Shadow.

CCS CONCEPTS

• **Theory of computation → Theory and algorithms for application domains**; • **Mathematics of computing →** *Approximation algorithms.*

KEYWORDS

Cliques, near-cliques, near-cliques, defective-cliques, Turán Shadow, sampling, graphs

ACM Reference Format:
Shweta Jain and C. Seshadhri. 2020. Provably and Efficiently Approximating Near-cliques using the Turán Shadow: PEANUTS. In *Proceedings of The Web Conference 2020 (WWW '20), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3366423.3380264

WWW '20, April 20–24, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-7023-3/20/04.
https://doi.org/10.1145/3366423.3380264

1 INTRODUCTION

Subgraph counting is an important tool in graph analysis that goes by many names such as motif counting, graphlet analysis, and pattern counting. The aim is to count the number of occurrences of a (generally small) subgraph in a much larger graph. Among these subgraphs, cliques are arguably the most important. Even the simplest clique, the triangle, has a rich history of algorithms and applications. There has been much recent focus on counting cliques in large graphs [5, 11, 14, 16, 19, 42].

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 extremely close to being cliques. We will call these structures near-cliques but they are also known as quasi-cliques [23, 27] and defective cliques [46] 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 [45].

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 [46] 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. [2, 8, 15, 22, 31]

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. 1i 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. [38] 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 §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.

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 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 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 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.

The different types of near-cliques are shown in Fig. 2. 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).

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.

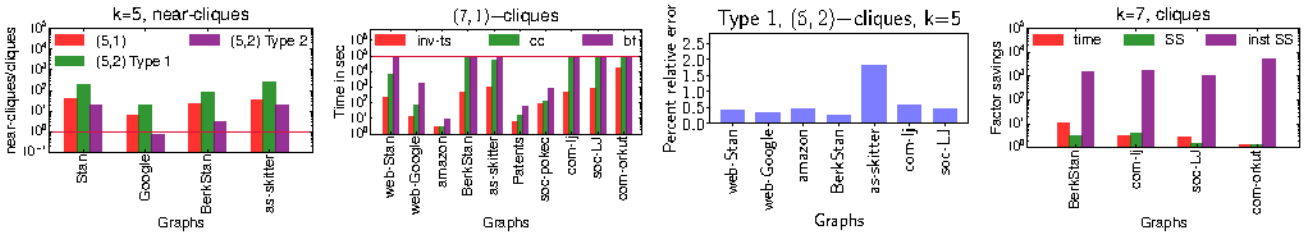
Extremely fast: PEANUTS is based on the observation that every near-clique contains a smaller clique. Thus, we can use cliques as clues for finding near-cliques. We leverage a fast clique-counting algorithm (TuránShadow) to achieve fast and accurate near-clique counting. Fig. 1ii shows the time taken 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. 1iii 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).

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).

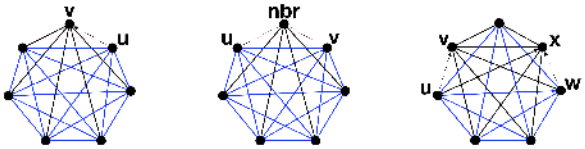
Excellent space efficiency: TuránShadow requires that the entire shadow be generated and stored, which for a graph with 100s of millions of edges can potentially require large amount of memory. Our practical implementation of Inverse-TS addresses this by removing the separation in the Shadow construction 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. 1iv 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. 1ii 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.



(i) Ratios **(ii) Timings** **(iii) Error** **(iv) TS vs Inverse-TS**

Figure 1: Fig. 1i 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. 1ii 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. Fig. 1iii 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. 1iv 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.



(i) $(7, 1)$ **(ii) Type 1** $(7, 2)$ **(iii) Type 2** $(7, 2)$

Figure 2: Near-7-cliques. Dotted lines indicate the missing edges. Blue lines mark the contained clique.

All code and data available: All the datasets we used are publicly available at [36]. In addition, we can readily make the code for our algorithm publicly available if the paper is accepted for publication.

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 [25, 29, 44], social sciences [18], spam detection [4], graph modeling [33], etc. Triangle counting, and more recently, clique counting have gained a lot of attention [11, 14, 19] due to their special role in characterizing real-world graphs. Clique counts have been employed in applications such as discovery of dense subgraphs [32, 39], in topological approaches to network analysis [35], graph clustering [45] among others. More generally, motif counts have been used in clustering [40, 45], evaluation of graph models [33, 34], classification of graphs [41] etc.

On the theoretical side, several motif-counting algorithms exist [9, 10, 45]. On the more practical side, only recently, efficient methods for counting graphlets upto size 5 [17, 20, 28, 43] have been proposed. Most of these are extensions of triangle counting methods and do not scale. For patterns of larger sizes, two widely used techniques are the MCMC [16, 42] and color-coding (CC) of [1]. However, as shown in [7], 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 [6, 21] and distributed settings [13] and in temporal networks [26].

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 [38]. 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 [32]). The algorithms of Andersen and Chellapilla [3], Rossi et al. [30], and Tsourakakis et al. [38, 39] 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.

2 MAIN IDEAS

The starting point of our result is the TuránShadow algorithm 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. 2. 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 §3 we set some basic notation. In §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 §5. Finally, in §6 we provide a detailed experimental study of Inverse-TS and its comparison with the state-of-the-art.

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 3.1. *[Theorem 1 in [12]] 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 MAIN ALGORITHM

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

Definition 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 [14], 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 [19], except that at each stage it also maintains the partial clique P.

Claim 4.2. *Given a graph G and integer k, PrefixedTuránShadowFinder returns a k-clique Prefixed-Turán-Shadow of G. Its running time is $O(|V|^{k+1})$.*

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 5.2 in [19], 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

Algorithm 1: 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 $\boldsymbol{S}$

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 2: Sample(S)
Inputs: S: $k-$clique Prefixed-Turán-Shadow of some graph G
Output: B: $k-$vertex set

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

Claim 4.3. *The probability of any k-clique K in G being returned by a call to Sample is $\frac{1}{w(S)}$.*

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(K is sampled) = Pr(E is sampled from D) $*$ Pr(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. □

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 3: 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

S = **PrefixedTurnShadowFinder(G, h)**
Let $w(\boldsymbol{S}) = \sum_{(P, S, \ell) \in \mathbf{S}} \binom{|S|}{\ell}$
For $i = 1, 2, ..., s$:
  $K = Sample(\mathbf{S})$
  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. *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 $\mathbf{E}[\hat{F}] = F$. 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 4.3, every h-clique in G has the same probability of being returned by Sample. $\mathbf{E}[X_i] = \sum_{K \in C_k(G)} \frac{f(K)}{w(\boldsymbol{S})} = \frac{F}{w(\boldsymbol{S})}$. Suppose $X_i \in [0, B]$. By Theorem 3.1, $Pr[|\sum_{i=1}^{s} X_i - s\mathbf{E}[X_i]| \geq \varepsilon s \mathbf{E}[X_i] \leq \delta$ 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 (Theorem 5.4 in [19]. The only difference is the addition of sT_f in the running time which is the time required to obtain f for s samples.

□

4.1 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^+.

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

Proof. Consider an h-clique $K \in C_h(G)$ and let v be the lowest order vertex according to degeneracy 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, $\mathbf{E}[X_i] = \sum_{v\in V} \sum_{K\in\mathcal{K}(E_v)} \frac{\Phi_v}{\Phi\phi_v} \frac{\phi_v}{\Phi_v} f(K) = \sum_{K\in C_k(G)} \frac{f(K)}{\Phi} = \frac{F}{\Phi}$

Algorithm 4: Inverse-TS($G, h, s, Func$)

1 Order G by degeneracy and convert it to a DAG DG.
2 Let M be a map, $W = 0$
3 Set probability distribution D over V where $p(v) = \sum_v \Phi_v/\Phi$.
4 For $i = 1, 2, \ldots, s$:
5 Independently sample a vertex v from D.
6 If $M[v]$ exists, set $\boldsymbol{S} = M[v]$
7 else
8 $\mathbf{S} = PrefixedTurnShadowFinder(G_{|N_v^+}, h-1)$
9 $M[v] = \mathbf{S}$
10 Let $\phi_v = \sum_{(P,S,\ell)\in\mathbf{S}} \binom{|S|}{\ell}$
11 Let $K = \{v\} \cup Sample(\mathbf{S})$
12 If K is a clique, set $X_i = \frac{\phi_v}{\Phi_v} * Func(G, K)$
13 else set $X_i = 0$
14 $W = W + X_i$
15 let $\hat{F} = \frac{W}{s}\Phi$
16 return $\hat{F}$

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 4.6. *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 $1-\delta$, $|\hat{F} - F| \le \varepsilon F$.*

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 4.5, their expectation $\mu = F/\Phi$. Suppose $X_i \in [0, B]$. By Theorem 3.1, $Pr[|\sum_{i=1}^{s} X_i - \mu s| \ge \varepsilon s\mu] \le \delta$ 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 [24]. 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 Thm. 5.4 from [19]. 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 4.2) 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 $O(min(\alpha \text{size}(S), s\alpha^k) + sT_f + m + n)$.

□

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.

5 COUNTING CLIQUES AND NEAR-K-CLIQUES

5.1 Counting $(k, 1)$-cliques

Algorithm 5: Func-$(k, 1)$-Clique(G, K)

```
1 f' = 0
2 Let u and v be two distinct vertices from K
3 Let nbrs = N_u ∪ N_v
4 For nbr ∈ nbrs:
5     If nbr is connected to all vertices in K except 1 vertex, say
      w and nbr > w, then f' = f' + 1
6 return f'
```

Definition 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 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 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 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 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, $T_f = O(d_{max})$

□

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

Algorithm 6: 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 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. 2). Thus, $\sum_{K' \in C_{k-1}(G)} f(K') = F$ = the total number of Type 1 $(k, 2)$-cliques in G. □

Claim 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 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, $T_f = O(d_{max})$. □

Algorithm 7: 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'

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

Definition 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. 2), and the lowest order $k-2$-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 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 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 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, $T_f = O(\alpha + d_{max})$. □

6 EXPERIMENTAL RESULTS

Preliminaries: 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 [36], including social networks, web networks, and infrastructure networks. The largest graph has more than 100M edges. Basic properties like degneracy, maximum degree etc. of these graphs are presented in Table 1. We consider the graph to be simple and undirected. Code for all experiments is available at: https://bitbucket.org/sjain12/counting-near-cliques

Our practical implementation differs slightly from Inverse-TS in two ways: we 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. 3. 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. 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

					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	$(k, 1)$
					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 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. 2

graph	k	revised estimate	revised % error	time	type
cit-Patents	10	648944	1.91	130	$(k, 1)$
		2.84E+06	1.06	130	$(k, 2)$ Type 1
		3.69+07	0.27	130	$(k, 2)$ Type 2
soc-pokec	5	3.91E+08	0.51	284	$(k, 1)$
		2.27E+09	0.44	371	$(k, 2)$ Type 1
soc-pokec	7	4.92E+08	0.01	288	$(k, 1)$
		1.53E+09	0.24	347	$(k, 2)$ Type 1
		6.27E+08	0.47	298	$(k, 2)$ Type 2

Table 2: Table revised estimates, revised error and time in seconds for the counts of near-cliques obtained using PEANUTS with 500K samples for the erroneous estimates in Tab. 1 (marked with *).

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. 2 shows the revised estimates obtained using PEANUTS using 500K samples, for values in Tab. 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.

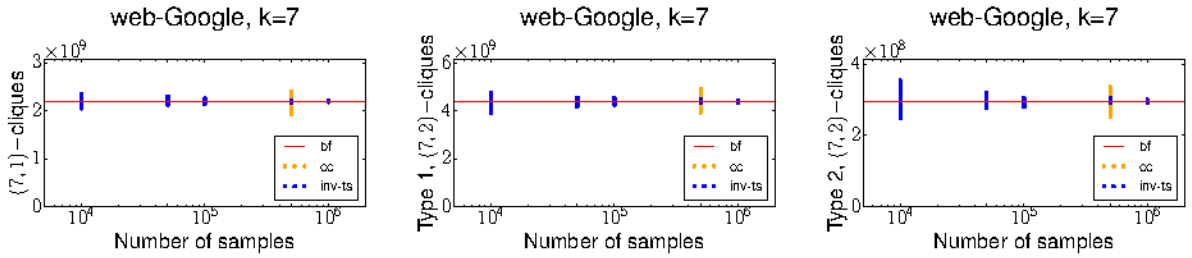
Running time: The runtimes for near-cliques of size 7 are presented in Tab. 1. We show the time for a single run in each case. In all cases except com-orkut, the algorithm terminated in minutes (for com-orkut, it took less than a day) where cc and bf did not terminate in an entire day (and in some cases, 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 [9]). 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.

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 [37]. 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



(i) $(k, 1)$-clique **(ii) Type 1, $(k, 2)$-clique** **(iii) Type 2, $(k, 2)$-clique**

Figure 3: Fig. 3i, Fig. 3ii, Fig. 3iii 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.

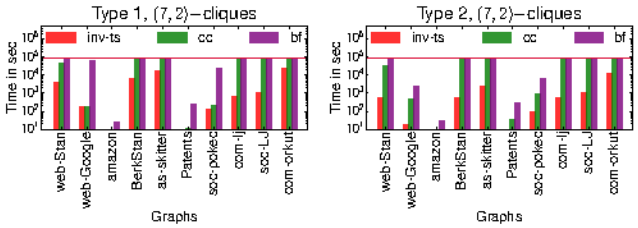


Figure 4: Figure shows the time required by Inverse-TS (inv-ts), color-coding (cc) and brute force (bf) to estimate the number of Type 1 and Type 2 $(k, 2)$-cliques resp. in 10 real world graphs for $k = 7$. The red line indicates 86400 seconds (24 hours).

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.

7 CONCLUSION AND 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.

ACKNOWLEDGMENTS

Shweta Jain and C. Seshadhri acknowledge the support of NSF Awards CCF-1740850, CCF-1813165, and ARO Award W911NF1910294.

REFERENCES

[1] Noga Alon, Raphy Yuster, and Uri Zwick. 1994. 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)* (Montreal, Quebec, Canada). 326–335. https://doi.org/10.1145/195058.195179
[2] J Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat, and Alessandro Vespignani. 2006. Large scale networks fingerprinting and visualization using the k-core decomposition. In *Advances in neural information processing systems*. 41–50.
[3] R. Andersen and K. Chellapilla. 2009. Finding Dense Subgraphs with Size Bounds. In *Workshop on Algorithms and Models for the Web-Graph (WAW)*. 25–37.
[4] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. 2008. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *KDD'08*. 16–24. https://doi.org/10.1145/1401890.1401898
[5] Mansurul A Bhuiyan, Mahmudur Rahman, Mahmuda Rahman, and Mohammad Al Hasan. 2012. Guise: Uniform sampling of graphlets for large graph analysis. In *2012 IEEE 12th International Conference on Data Mining*. IEEE, 91–100.
[6] I. Bordino, D. Donato, A. Gionis, and S. Leonardi. 2008. Mining Large Networks with Subgraph Counting. In *Proceedings of International Conference on Data Mining*. 737–742.
[7] Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. 2018. Motif Counting Beyond Five Nodes. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 12, 4 (2018), 48.

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

[9] Norishige Chiba and Takao Nishizeki. 1985. Arboricity and subgraph listing algorithms. *SIAM J. Comput.* 14 (1985), 210–223. Issue 1. https://doi.org/10.1137/0214017

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

[11] Maximilien Danisch, Oana Balalau, and Mauro Sozio. 2018. Listing k-cliques in Sparse Real-World Graphs. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 589–598.

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

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

[14] Irene Finocchi, Marco Finocchi, and Emanuele G. Fusco. 2015. Clique Counting in MapReduce: Algorithms and Experiments. *ACM Journal of Experimental Algorithmics* 20 (2015). https://doi.org/10.1145/2794080

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

[16] Guyue Han and Harish Sethu. 2016. Waddling random walk: Fast and accurate mining of motif statistics in large graphs. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE, 181–190.

[17] Tomaž Hočevar and Janez Demšar. 2017. Combinatorial algorithm for counting small induced graphs and orbits. *PloS one* 12, 2 (2017), e0171428.

[18] P. Holland and S. Leinhardt. 1970. A method for detecting structure in sociometric data. *Amer. J. Sociology* 76 (1970), 492–513.

[19] Shweta Jain and C Seshadhri. 2017. A Fast and Provable Method for Estimating Clique Counts Using Turán's Theorem. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 441–449.

[20] M. Jha, C. Seshadhri, and A. Pinar. 2015. Path Sampling: A Fast and Provable Method for Estimating 4-Vertex Subgraph Counts. In *World Wide Web (WWW)*. 495–505.

[21] Daniel M Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. 2012. Counting arbitrary subgraphs in data streams. In *International Colloquium on Automata, Languages, and Programming*. Springer, 598–609.

[22] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. 1999. Trawling the Web for emerging cyber-communities. *Computer networks* 31, 11-16 (1999), 1481–1493.

[23] Guimei Liu and Limsoon Wong. 2008. Effective pruning techniques for mining quasi-cliques. In *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 33–49.

[24] David W Matula and Leland L Beck. 1983. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM (JACM)* 30, 3 (1983), 417–427.

[25] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. 2002. Network motifs: Simple building blocks of complex networks. *Science* 298, 5594 (2002), 824–827.

[26] Ashwin Paranjape, Austin R Benson, and Jure Leskovec. 2017. Motifs in temporal networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 601–610.

[27] Jeffrey Pattillo, Nataly Youssef, and Sergiy Butenko. 2012. Clique relaxation models in social network analysis. In *Handbook of Optimization in Complex Networks*. Springer, 143–162.

[28] Ali Pinar, C Seshadhri, and Vaidyanathan Vishal. 2017. Escape: Efficiently counting all 5-vertex subgraphs. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1431–1440.

[29] Nataša Pržulj. 2007. Biological network comparison using graphlet degree distribution. *Bioinformatics* 23, 2 (2007), e177–e183.

[30] Ryan A Rossi, David F Gleich, and Assefaw H Gebremedhin. 2015. Parallel Maximum Clique Algorithms with Applications to Network Analysis. *SIAM Journal on Scientific Computing* 37, 5 (2015), C589–C616.

[31] Ahmet Erdem Sariyuce, C Seshadhri, Ali Pinar, and Umit V Catalyurek. 2015. Finding the hierarchy of dense subgraphs using nucleus decompositions. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 927–937.

[32] Ahmet Erdem Sariyüce, C. Seshadhri, Ali Pinar, and Ümit V. Çatalyürek. 2015. Finding the Hierarchy of Dense Subgraphs using Nucleus Decompositions. (2015), 927–937.

[33] C. Seshadhri, Tamara G. Kolda, and Ali Pinar. 2012. Community structure and scale-free collections of Erdös-Rényi graphs. *Physical Review E* 85, 5 (May 2012), 056109. https://doi.org/10.1103/PhysRevE.85.056109

[34] Miguel EP Silva, Pedro Paredes, and Pedro Ribeiro. 2017. Network motifs detection using random networks with prescribed subgraph frequencies. In *Workshop on Complex Networks CompleNet*. Springer, 17–29.

[35] Ann Sizemore, Chad Giusti, and Danielle S. Bassett. 2016. Classification of weighted networks through mesoscale homological features. *Journal of Complex Networks* 10.1093 (2016).

[36] SNAP [n.d.]. Stanford Network Analysis Project (SNAP). Available at http://snap.stanford.edu/.

[37] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. ArnetMiner: Extraction and Mining of Academic Social Networks. In *KDD'08*. 990–998.

[38] C. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. Tsiarli. 2013. Denser Than the Densest Subgraph: Extracting Optimal Quasi-cliques with Quality Guarantees. In *Knowledge Data and Discovery (KDD)*.

[39] Charalampos E. Tsourakakis. 2015. The K-clique Densest Subgraph Problem. In *Proceedings of the Conference on World Wide Web WWW*. 1122–1132. https://doi.org/10.1145/2736277.2741098

[40] Charalampos E. Tsourakakis, Jakub W. Pachocki, and Michael Mitzenmacher. 2016. Scalable motif-aware graph clustering. *CoRR* abs/1606.06235 (2016). http://arxiv.org/abs/1606.06235

[41] Johan Ugander, Lars Backstrom, and Jon M. Kleinberg. 2013. Subgraph frequencies: mapping the empirical and extremal geography of large graph collections. In *WWW*, Daniel Schwabe, Virgílio A. F. Almeida, Hartmut Glaser, Ricardo A. Baeza-Yates, and Sue B. Moon (Eds.). International World Wide Web Conferences Steering Committee / ACM, 1307–1318.

[42] Pinghui Wang, John Lui, Bruno Ribeiro, Don Towsley, Junzhou Zhao, and Xiaohong Guan. 2014. Efficiently estimating motif statistics of large networks. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 9, 2 (2014), 8.

[43] Pinghui Wang, Junzhou Zhao, Xiangliang Zhang, Zhenguo Li, Jiefeng Cheng, John CS Lui, Don Towsley, Jing Tao, and Xiaohong Guan. 2018. MOSS-5: A fast method of approximating counts of 5-node graphlets in large graphs. *IEEE Transactions on Knowledge and Data Engineering* 30, 1 (2018), 73–86.

[44] Sebastian Wernicke. 2006. Efficient Detection of Network Motifs. *IEEE/ACM Trans. Comput. Biology Bioinform.* 3, 4 (2006), 347–359.

[45] Hao Yin, Austin R Benson, Jure Leskovec, and David F Gleich. 2017. Local higher-order graph clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 555–564.

[46] Haiyuan Yu, Alberto Paccanaro, Valery Trifonov, and Mark Gerstein. 2006. Predicting interactions in protein networks by completing defective cliques. *Bioinformatics* 22, 7 (2006), 823–829.