YACC: A Framework Generalizing TuránShadow for Counting Large Cliques*

Shweta Jain†
University of Utah
shweta.jain@utah.edu

Hanghang Tong
University of Illinois, Urbana-Champaign
htong@illinois.edu

Abstract

Clique-counting is a fundamental problem that has application in many areas eg. dense subgraph discovery, community detection, spam detection, etc. The problem of k-clique-counting is difficult because as k increases, the number of k-cliques goes up exponentially. Enumeration algorithms (even parallel ones) fail to count k-cliques beyond a small k. Approximation algorithms, like TuránShadow have been shown to perform well upto $k = 10$, but are inefficient for larger cliques. The recently proposed Pivoter algorithm significantly improved the state-of-the-art and was able to give exact counts of all k-cliques in a large number of graphs. However, the clique counts of some graphs (for example, `com-lj`) are still out of reach of these algorithms.

We revisit the TuránShadow algorithm and propose a generalized framework called YACC that leverages several insights about real-world graphs to achieve faster clique-counting. The bottleneck in TuránShadow is a recursive subroutine whose stopping condition is based on a classic result from extremal combinatorics called Turán's theorem. This theorem gives a lower bound for the k-clique density in a subgraph in terms of its edge density. However, this stopping condition is based on a worst-case graph that does not reflect the nature of real-world graphs. Using techniques for quickly discovering dense subgraphs, we relax the stopping condition in a systematic way such that we get a smaller recursion tree while still maintaining the guarantees provided by TuránShadow. We deploy our algorithm on several real-world data sets and show that YACC reduces the size of the recursion tree and the running time by over an order of magnitude. Using YACC, we are able to obtain clique counts for several graphs for which clique-counting was infeasible before, including `com-lj`.

Keywords: cliques, TuránShadow, sampling, degeneracy

1 Introduction

Pattern counting (also known as motif counting) is an important graph analysis tool with applications in social network analysis, bioinformatics, cybersecurity, physics and many other domains [28]. One of the characteristic properties of real-world graphs is that they show very high counts of certain patterns – much higher than what one would expect from a random graph of the same size [16, 21]. Of particular interest are k-cliques: complete subgraphs on k-vertices. Cliques are the archetypal examples of dense subgraphs and have been used in many applications including spam detection, anomaly detection, community detection among others [25, 30, 34]. There is a large body of literature for counting triangles (3-cliques) [20, 3, 17, 28]. Obtaining counts of higher-order cliques is however, a difficult problem – essentially, as k increases, the count of k-cliques goes up exponentially and as a result, enumeration based algorithms are unable to count beyond small k (typically 5). In recent years, other approaches have yielded significant improvement. One such line of work (consisting of algorithms like TuránShadow [17], PEANUTS [19]) uses a combination of enumeration and sampling to give approximate clique-counts. It made clique counting feasible for k upto 10. Several parallel and distributed algorithms have also been proposed which are typically able to count upto $k = 13$. The largest improvement was arguably due to Pivoter [18] which was the first algorithm to be able to count cliques without enumerating them. It was able to obtain exact counts of k-cliques for *all* k for a large number of graphs. However, the authors of [18] point out, there are graphs like `com-lj` for which Pivoter was not able to obtain even the count of 8-cliques within a day (the parallel version of the algorithm also was unable to count beyond $k = 10$) and we discovered that there are several such examples where the existing techniques do not suffice for counting the number of k-cliques. We want to design faster clique-counting algorithms that can handle such graphs.

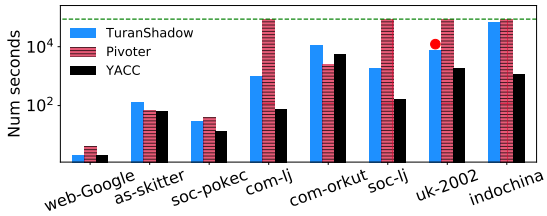
1.1 Problem Definition A k-clique is a complete graph on k vertices. Given an undirected, simple graph $G(V, E)$ and a positive integer $k \leq |V|$, we want to count all k-cliques in G. Note that these cliques need not be maximal (they could be a part of bigger cliques). We make no distributional assumption on the graph. All probabilities are over the internal randomness of the algorithm itself and are independent of the instance.

1.2 Main Contributions We present an algorithm called YACC[1] that uses the framework of the TuránShadow [17] algorithm and combines it with

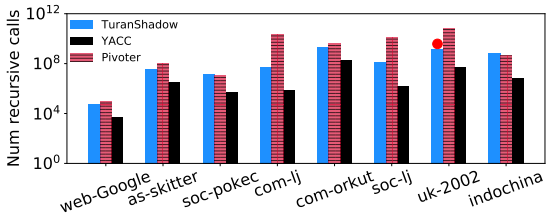
*This work is supported by NSF (1947135, 2134079, and 1939725), and by NIFA award 2020-67021-32799.

†Work done while at the University of Illinois, Urbana-Champaign.

[1] Yet Another Clique Counter



(i) Timing in seconds for counting $k = 20$ cliques



(ii) Size of the recursion tree for $k = 20$

Figure 1: Fig. 1i shows the time required by the YACC, TURÁNSHADOW and PIVOTER for estimating the number of 20-cliques. In most cases, YACC is $> 10\times$ faster. The horizontal line corresponds to 1 day and TURÁNSHADOW (PIVOTER had not terminated in 24 hours for many of the graphs). Fig. 1ii shows the number of recursive calls (size of the recursion tree) made by the three algorithms for estimating the number of 20-cliques. The red circle represents graphs for which TURÁNSHADOW crashed and we used the size of its recursion tree at the time of crashing. For algorithms that crashed, we looked at the size of the tree created in 1 day. In all cases, the size of the recursion tree goes down by at least an order or magnitude using YACC. Details about these graphs can be found in Tab. 1

several observations that better leverage the structure of real-world graphs, to obtain a faster clique-counting algorithm with provable guarantees. TURÁNSHADOW is a recursive algorithm whose running time is proportional to the size of the recursion tree. The main objective of our techniques is to reduce the size of the recursion tree, thereby giving faster algorithms for clique counting. Our main contributions can be summarized as follows:

Generalized framework incorporating Turán Shadow: TURÁNSHADOW comprises two subroutines: a recursive procedure to build an object called the Turán Shadow (a collection of dense subgraphs), and a sampling procedure that samples cliques in the Turán Shadow to give an unbiased estimate for the count of k-cliques. As noted by PEANUTS [19], sampling is cheap whereas a large part of the total running time of TURÁNSHADOW is spent in constructing the shadow. The density threshold used to determine when a subgraph is "dense" determines how much time the algorithm spends in constructing the shadow vs. how much time the algorithm spends in sampling, and there is a tradeoff in the time required for these two tasks. The TURÁNSHADOW algorithm resolves this tradeoff in a hard way (using a classic result from extremal graph theory called Turán's theorem) and is oblivious to the structure of the graph at hand. However, in many real-world scenarios, this is wasteful and indeed, minimizing this waste is critical for being able to obtain clique-counts in certain hard graphs. We provide a generalized framework that gives greater control over this tradeoff, while still maintaining the guarantees provided by TURÁNSHADOW. As a result, we are able to obtain clique counts for several graphs for which clique-counting was infeasible before.

Smaller recursion tree: TURÁNSHADOW is a recursive algorithm whose running time is proportional to the size of its recursion tree. It uses the classic Turán's theorem as a stopping condition which in turn governs the size of the recursion tree. We use several observations about real-world graphs to systematically relax this stopping condition (which results in a much smaller recursion tree than the recursion tree of TURÁNSHADOW) while providing the same guarantees as TURÁNSHADOW. Although this reduction in the size of the recursion tree, in theory, comes at the cost of requiring more samples, in practice we observe that this cost is negligible. Fig. 1ii shows the size of the recursion trees generated by TURÁNSHADOW, YACC and PIVOTER for counting the number of 20-cliques in several graphs. In many cases, we obtain at least an order of magnitude reduction in the size of the recursion tree in YACC as compared to TURÁNSHADOW (and several orders of magnitude compared to that of PIVOTER). This opens the doors for faster versions of algorithms which use TURÁNSHADOW as a subroutine, for example PEANUTS.

Faster clique-counting: The reduced size of the recursion tree directly translates into savings in runtime. Fig. 1i shows the time required by YACC, TURÁNSHADOW and PIVOTER for obtaining the counts of 20-cliques in several real-world graphs. In most cases, YACC was able to estimate the clique counts in minutes and was at least an order of magnitude faster than TURÁNSHADOW and PIVOTER. For several graphs, TURÁNSHADOW and PIVOTER had not terminated even after 24 hours, and this difference in running time was even more pronounced for $k = 40$. Because of this ability to count cliques quickly, YACC is able to obtain clique-counts for several graphs for which clique-counting was infeasible before. For example, to the best of our knowledge, *this is the first work to be able to count the number of* 40*-cliques in* `uk-2002`. This will allow the use of counts of bigger cliques in the analysis of real-world graphs.

1.3 Related Work

Works showing the importance of subgraph-counting (also known as motif-counting and graphlet-counting) in the social-sciences date back to the 70's [16]. Since then their use has been demonstrated in many different applications in network science, bioinformatics, recommendation systems etc. (refer the tutorial [28] and references therein). It started with couting cliques of size 3 (also known as triangles). Triangles are especially important as they are intimately tied to the community structure in

real-world graphs and are used in spam detection [5], graph modeling [26], and role detection [10]. Recent works have used cliques of larger sizes for graph visualization [22], dense subgraph discovery and community detection [25, 30] and several other applications [32, 24, 33]. For all such applications, having fast algorithms for counting cliques is crucial.

A long line of work has been dedicated to counting triangles and some of these have been extended for up to $k = 5$ [11, 4, 27, 20]. A number of randomized techniques like color coding [4, 35], edge sampling methods [23] and MCMC methods [6] have also been proposed. However, these do not scale beyond a small k (typically $k = 5$) for graphs with millions of vertices [20].

[15] gave a MapReduce algorithm that used orientation and sampling and was able to count up to $k = 8$. [17] proposed an approximation algorithm called TuránShadow that used the classic Turán's theorem and was able to count k-cliques for $k <= 10$, and this algorithm is the starting point for both algorithms presented in this paper. Several other parallel algorithms have been proposed that use clever orientations and remarkably, have been able to enumerate cliques (typically) up to size 13 [12, 29]. The first algorithm that was able to obtain clique-counts for larger k is the recently proposed Pivoter algorithm [18] which used the classic technique of pivoting and was able to count k-cliques for all k in a large number of graphs. However, many of these works [18, 12, 29] note the difficulty of obtaining clique-counts in several hard graphs, which are the focus of this paper.

2 Main Ideas

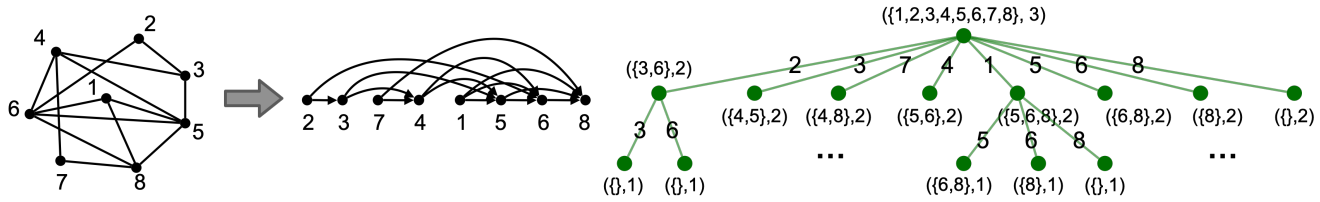
The starting point for this work is the TuránShadow algorithm proposed in [17]. We give a description of the TuránShadow algorithm and its stopping condition below, and list some of the challenges in counting k-cliques for large k. We then discuss how we address those challenges and state our main algorithm.

2.1 TuránShadow Clique-counting has traditionally been done through recursive enumeration algorithms [11, 12]. The main idea in these algorithms is that to discover all k-cliques involving a vertex v, it suffices to discover all $(k-1)$-cliques among vertices adjacent to v. By calling itself recursively on the set of vertices in the neighborhood of v to discover all $(k-1)$-cliques among them, and by combining every $(k-1)$-clique it finds in the neighborhood with v, the algorithm will have discovered every k-clique involving vertex v. By carrying out this procedure for every $v \in V$, the algorithm will have discovered every k-clique in the graph. But it will have discovered every clique multiple times. In order to get around this problem, the graph is first converted into a DAG by ordering the vertices and orienting the edges from lower to higher vertices according to the ordering, and instead of the neighborhood of every vertex v, the algorithm calls itself recursively on only the *outneighborhood* of v. One can show that this recursive procedure will discover every clique exactly once. The ordering used is often the degeneracy ordering which is formed by peeling the lowest degree vertex at every step (ties are broken by id). The largest outdegree of any vertex in this ordering is known as the degeneracy (denoted by α) of a graph. Many real-world graphs have low degeneracy (even graphs with millions of vertices have degeneracy typically in the 100s [17]). Ordering by degeneracy thus helps ensure that the outneighborhoods are small.

It would be useful to look at the recursion tree of this algorithm. At every recursive call, the algorithm has a set of vertices, say S, in which the algorithm wants to count ℓ-cliques for $\ell \leq k$. The nodes of the recursion tree (we will use 'vertices' for vertices in the graph G and 'nodes' for nodes in the tree) represent the recursive calls made by the algorithm and are labeled by the tuple (S, ℓ) (the root is labeled (V, k)). For every vertex $v \in S$, let $N_v^+(S)$ represent the outneighborhood of v in S. For every vertex $v \in S$, the algorithm calls itself recursively on $N_v^+(S)$ to count the number of $(\ell-1)$-cliques in $N_v^+(S)$. In the recursion tree, this is represented as $|S|$ children of that node, and the link to the child call corresponding to $N_v^+(S)$ is labeled by v. A pictorial representation of this tree is given in Fig. 2. Note that for any node (labeled say, S) in this tree, if we let P be the set of link labels encountered on the path from the root to that node, then P represents a clique and S is the set of vertices that lies in the common outneighborhood of all vertices in P. Moreover, $\ell = k - |P|$. Thus, in the subgraphs at level i, we are interested in counting the number of $(k-i)$-cliques. Moreover, every path P from the root to some node at level i represents a unique i-clique from G (and every i-clique from G is represented by the set of labels on the root-to-node path of a unique node at level i i.e. there is a bijection).

As the number of child nodes that can be spawned is bounded by α i.e. the degeneracy of the graph, the size of this tree can blow up as the depth of the tree increases. One could optimize the calculation of clique counts slightly by noting that if S is a clique, the number of ℓ-cliques in S is exactly $\binom{|S|}{\ell}$ and we can save on building the subtree rooted at S. However, despite this optimization, recursion trees of real-world graphs blow up in size as k increases [18]. This is why enumeration algorithms typically terminate at a certain depth and are able to count k-cliques only up to that depth.

TuránShadow remedies this by noting that as the recursion goes deeper the subgraphs S typically become denser. It uses the observation that when a subgraph is sufficiently dense, estimating the counts of cliques using random sampling is more efficient than expanding out the entire subtree. Hence, when a subgraph becomes "dense", the algorithm simply adds the subgraph to a collection of subgraphs – called the Turán Shadow, and terminates that branch. Once all the dense subgraphs have been collected, the algorithm performs random sampling on the subgraphs to give an unbiased estimate of the total count of k-cliques in the graph. If a subgraph at level i is dense then in the subgraph, the algorithm will sample for $(k-i)$-cliques. But how dense does a subgraph have to be before it can be added to the Shadow and how many samples does one need to take? To answer this question, the TuránShadow algorithm uses



(i) Convert G to DAG using degeneracy ordering.

(ii) Recursion tree for G.

Figure 2: Fig. 2i and Fig. 2ii depict the stages of obtaining the recursion tree for an example graph.

the classic Turán's theorem which gives a lower bound for the ℓ-clique density in terms of the edge density. We describe this theorem below.

We will use n for the number of vertices and m for the number of edges in the given graph $G(V, E)$. For any arbitrary graph $H = (V(H), E(H))$, we will use $\rho_\ell(H) := |C_\ell(H)|/\binom{|V(H)|}{\ell}$ for the ℓ-clique density where $C_\ell(H)$ denotes the set of ℓ-cliques in H. $\rho_2(H)$ represents the edge density. Let $ex(n, H, F)$ denote the maximum number of (non-induced) copies of a graph H that an n-vertex graph can have without having a copy of F. Turán's theorem determines this quantity for $H =$ 2-clique (edge) and $F =$ ℓ-clique. It states the following:

THEOREM 2.1. *(Turán [31]) For $F =$$\ell$-clique, and $H =$ 2-clique, $ex(n, H, F) = (1 - \frac{1}{\ell-1})\frac{n^2}{2}$.*

Turán and Erdos also showed that if the graph has density greater than the Turán edge-density for ℓ then in fact it has $\Omega(n^{\ell-2})$ ℓ-cliques.

THEOREM 2.2. *(Erdős [14]) For any graph H over n vertices, if $\rho_2(H) > 1 - \frac{1}{k-1}$, then H contains at least $(n/(k-1))^{k-2}$ k-cliques.*

The density threshold given by Turán's theorem controls how deep the algorithm needs to go before it deems a subgraph dense enough for sampling. The higher this density threshold, the deeper the algorithm has to go, and larger the size of the recursion tree. Herein lies the challenge for counting k-cliques for large k using TURÁNSHADOW.

2.2 Challenges and Main Insights As k (and correspondingly, ℓ) increases, the density threshold given by Turán's algorithm becomes very large. For $\ell = 20$, the threshold value is 0.95. This means that unless 95% of the possible edges in the subgraph are present, the subgraph will not qualify as "dense". This is a prohibitively large value that is seldom met. In such cases, the TURÁNSHADOW algorithm defaults to continuing to build the recursive subtree. As k becomes larger, this problem becomes even more pronounced and TURÁNSHADOW is unable to count k-cliques for such k.

The underlying assumption here is that if the Turán density threshold is not met by the subgraph, then we cannot give any guarantees about the number of ℓ-cliques in the subgraph. However, this is not necessarily true. If the subgraph has a dense region (which is likely to be the case if the subgraph has large cliques), then the subgraph could still have enough ℓ-cliques such that sampling would be efficient. Indeed, the Turán density is derived from highly structured and contrived, worst-case graphs that are rarely seen in practice and as a result, the Turán density threshold is unnecessarily high. For example, for $\ell = 3$, Turán's theorem says that a graph needs to have edge density $> \frac{1}{2}$ before we are guaranteed that there are triangles in it. The only graph that has this high number of edges without having a triangle is the complete bipartite graph on n vertices with each side of size $n/2$ (analogous graphs for ℓ-cliques are balanced $(\ell-1)$-partite graphs. These graphs are called Turán graphs). In real-world graphs neighborhoods of vertices are rarely bipartite (in fact, they have community structure and high clustering coefficients) and even when they don't have very high edge-density they have many triangles. How then can we capture this phenomena which clearly differs from the Turán graph, no subset of which is dense? Our main ideas essentially try to answer this question.

We are thus looking for indicators of the existence of ℓ-cliques. We do the following: if a subgraph of size s in which we are looking for ℓ-cliques has a dense region of a significant size that is guaranteed to have many ℓ-cliques then we declare the subgraph dense and add it to the Shadow. We take as input a parameter μ, $0 < \mu < 1$ that helps us control how large this dense region needs to be compared to the size of the subgraph before we declare the subgraph dense and sample in it. Note that a dense subgraph of size μs contains $\Omega((\mu s)^{\ell-2})$ ℓ-cliques. This guarantees a smaller number of ℓ-cliques compared to the number of ℓ-cliques that would have been guaranteed to exist had the entire subgraph been dense ($\Omega(s^{\ell-2})$). As a result, the number of samples required goes up by a factor of $\mu^{\ell-2}$ in theory. However, in practice we find that the increase in the number of samples required is negligible. In essence, this technique achieves the effect of declaring the subgraph dense even when the subgraph as a whole does not meet the Turàn density. But if we would have done so directly by manually reducing the threshold to a lower value (say 0.6), we would not have been able to give any guarantees about the number of samples required, nor about the probability of error or error margin.

What about the case when only a small region of the subgraph ($< \mu s$ in size) is dense? Can we extract this dense region and add it to the shadow, and build the recursion tree for only the remaining vertices? Note that naively splitting the graph into dense and sparse regions and calculating the cliques in the two regions separately (using sampling for the dense region and continuing recursive enumeration for the sparse region) would mean that we miss out on cliques going across these two regions. More importantly, how do

we efficiently find a dense region to extract? Checking for every possible subset of vertices (whether it is dense or not) would become computationally prohibitive.

We solve both these problems using properties of the degeneracy ordering. The TuránShadow algorithm, at each recursive call, orients the vertices of the subgraph S according to the degeneracy ordering *of* S. If the vertices that we remove (i.e. vertices in the dense region, say R) form a suffix of the ordering, then vertices from this dense region can still appear in the outneighborhood of vertices earlier in the ordering that are not part of the dense region i.e. vertices from R can still appear in the outneighborhood of vertices in $S \setminus R$. Any clique that goes across the dense region and the remaining subgraph (i.e. across R and $S \setminus R$) must contain a vertex from $S \setminus R$ and will be discovered by the subtree rooted at the smallest vertex in the clique according to the ordering.

Crucially, the degeneracy ordering has the property that the dense parts of the graph tend to be towards the end of the ordering. For example, one can show that if the subgraph S has a t-clique and the degeneracy of subgraph is $t-1$ (degeneracy cannot be $< t-1$), and if every subgraph that has minimum degree $t-1$ is a clique, then the suffix of size t of the ordering (last t vertices of the ordering) must form a clique. (Refer §3 for a proof).

Thus, given a subgraph S of size s in which we want to count ℓ-cliques, we form the degeneracy ordering of S and consider progressively larger suffixes as long as the subgraph induced by the suffix is "dense" (i.e. the edge density is greater than the Turán threshold). Let R represent the dense suffix. We remove a suffix R' of $|R|/\mu$ vertices and add it to the shadow. We do not explore the outneighborhoods of the vertices in R'. The outneighborhoods of the rest of the vertices $(S \setminus R')$ continue to get explored as in TuránShadow.

Finally, at every recursive call, we aggressively prune away vertices whose degree is $< \ell$. We use these techniques to obtain a Turán Shadow (the object that TuránShadow creates and on which sampling can be performed with guarantees about the solution quality). The main technical insights are geared towards obtaining this Shadow much more efficiently than TuránShadow. The rest of our algorithm (sampling from the Shadow and using the samples to estimate the count of cliques) follows the same procedure as TuránShadow. Note that the Shadow and the recursion tree are of independent interest as algorithms like PEANUTS use this Shadow (for purposes like counting near-cliques).

3 YACC

We will first show that under certain reasonable conditions, a suffix of the degeneracy ordering forms a large clique.

Theorem 3.1. *Let G be a graph with degeneracy α. For any set of vertices S, let $G|_S$ denote the subgraph induced by the vertices in S. Let $\delta(S)$ represent the minimum degree of any vertex in $G|_S$. Suppose every S for which $\delta(S) = \alpha$ is a clique. Let ϕ denote the degeneracy ordering of G (ties are broken by id) and let R represent the suffix of ϕ of length $\alpha+1$. Then R is an $(\alpha+1)$-clique.*

Proof. The degeneracy of a graph $G = \max_{S \subseteq V} \delta(S)$ [13]. Let G be a graph with degeneracy α such that every subset S of G with $\delta(S) = \alpha$ is a clique.

The degeneracy ordering of a graph is obtained through a peeling process: at every step, a vertex with the minimum degree is removed from the graph (ties are broken by id) and the degeneracy ordering is the order in which the vertices are peeled. It can be shown that if we orient all the edges from lower to higher vertices in the ordering then the maximum outdegree of any vertex in the ordering is exactly α [13].

Let K be an $(\alpha+1)$ clique in G. If $R = K$ then the claim is trivially true. If $R \neq K$ then $\exists v \in K, v \notin R$. Consider the smallest such v according to ϕ. Consider the step at which v is peeled from G. Let W represent the set of vertices not yet peeled at the start of this step. Thus, $v \in W, K \subset W, R \subset W$. Thus, $|W| \geq (\alpha+1)$. Since v is about to be peeled from W, v must be a min degree vertex in W. Moreover, since $K \subset W$, degree of v is at least α. But this violates the condition that every S for which $\delta(S) = \alpha$ is a clique (since W is not a clique because if it were, degeneracy would be $> \alpha$). Hence, proved. ∎

We will need this important corrollary from [17].

Corollary 3.1. (Corollary 3.3 from [17]) *Let $f(k) = k^{k-2}/k!$. For any graph H over n vertices, if $\rho_2(H) > 1 - \frac{1}{k-1}$, then $\rho_k(H) \geq 1/f(k)n^2$.*

Our main algorithm follows the framework of TuránShadow. Central to TuránShadow is an object called the γ-saturated Turán Shadow which the algorithm builds.

Definition 3.1. *[From [17]]: 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 $\forall (S,\ell) \in \boldsymbol{S}$, $\rho_\ell(S) \geq \gamma$.

Using the techniques described in the previous section, `YACC-Shadow-Builder` builds a γ-saturated k-clique shadow. A crucial difference from TuránShadow is that in Step 9, the algorithm recursively explores the outneighborhoods of only those vertices in S that are not in R'. Thus where the TuránShadow algorithm made $|S|$ recursive calls, YACC only makes $|S| - |R'|$ recursive calls.

Once a γ-saturated Turán Shadow is obtained, both TuránShadow and YACC sample from this shadow to estimate the count of k-cliques. The main task is to prove that the output of `YACC-Shadow-Builder` is a γ-saturated Turán Shadow (for $\gamma = 1/\max_{(S,\ell)\in\boldsymbol{S}}(f(\ell)|S|^2/\mu^{\ell-2})$). Once we have a γ-saturated Turán Shadow, all the guarantees of space, time complexity and accuracy of TuránShadow apply.

Theorem 3.2. *The output $\boldsymbol{S}$ of `YACC-Shadow-Builder`(G,k,μ) is a γ-saturated k-clique shadow, where $\gamma = 1/\max_{(S,\ell)\in\boldsymbol{S}}(f(\ell)|S|^2/\mu^{\ell-2})$.*

Algorithm 1: `YACC-Shadow-Builder`(G, k, μ)

Initialize $\boldsymbol{T} = \{(V, k)\}$ and $\boldsymbol{S} = \emptyset$
While $\exists (S, \ell) \in \boldsymbol{T}$ such that $\rho_2(S) \leq 1 - \frac{1}{\ell - 1}$
    Construct the degeneracy DAG $D(G|_S)$ and let Π denote the degeneracy ordering
    Let R be the largest suffix of Π such that $\rho_2(R) \geq 1 - \frac{1}{\ell-1}$.
    Let R' be a suffix of Π of size $min(|R|/\mu, |S|)$.
    Add (R', ℓ) to $\boldsymbol{S}$
    Let N_s^+ denote the outneighborhood (within $D(G|_S)$) of $s \in S$
    Delete (S, ℓ) from $\boldsymbol{T}$
    For each $s \in S \setminus R'$
        If $\ell \leq 2$ or $\rho_2(N_s^+) > 1 - \frac{1}{\ell-2}$
            Add $(N_s^+, \ell - 1)$ to $\boldsymbol{S}$
        Else, add $(N_s^+, \ell - 1)$ to $\boldsymbol{T}$
Output $\boldsymbol{S}$

Proof. Similar to theorem 5.2 in [17], we first prove by induction the following loop invariant for `YACC-Shadow-Builder`: $\boldsymbol{T} \cup \boldsymbol{S}$ is always a k-clique shadow.

At the start, $\boldsymbol{T} = \{(V, k)\}$ and $\boldsymbol{S} = \emptyset$. Thus the base case is valid.

For the induction step, assume that at the beginning of some iteration $\boldsymbol{T} \cup \boldsymbol{S}$ is a k-clique shadow.

The element (S, ℓ) is deleted from $\boldsymbol{T}$.

For $s \in S \setminus R'$, $(N_s^+, \ell - 1)$ is added to $\boldsymbol{S}$ or to $\boldsymbol{T}$. Also R' is added to $\boldsymbol{S}$.

Thus, it suffices to prove that there is a bijection mapping between $C_\ell(S)$ and $\bigcup_{s \in S \setminus R'} C_{\ell-1}(N_s^+) \cup C_\ell(R')$.

Let C be a ℓ-clique in S. If $C \subseteq R'$, then $C \nsubseteq C_{\ell-1}(N_s^+)$ for any $s \in S \setminus R'$ and C can be mapped to its copy in $C_\ell(R')$. If $C \notin R'$, then $\exists s \in S \setminus R', s \in C$. Let s' be the smallest vertex in C according to the degeneracy ordering of S. Then $C \setminus s'$ is an $(\ell - 1)$-clique contained exactly once and in $C_{\ell-1}(N_{s'}^+)$ thus C can be mapped to this $\ell - 1$-clique in $N_{s'}^+$.

In the other direction, every $C \in R'$ can be mapped to its own copy in S. Every $(\ell-1)$-clique J in N_s^+ for $s \in S \setminus R'$ can be mapped uniquely to exactly the clique $\{s\} \cup J$ in S. Thus, there is a bijection between the cliques in $C_\ell(S)$ and $\bigcup_{s \in S \setminus R'} C_{\ell-1}(N_s^+) \cup C_\ell(R')$.

When `YACC-Shadow-Builder` terminates, $\boldsymbol{T} \cup \boldsymbol{S}$ is a k-clique shadow. Since $\boldsymbol{T}$ must be empty, $\boldsymbol{S}$ is a k-clique shadow. Moreover, whenever a pair (R', ℓ) is added in $\boldsymbol{S}$, R' contains a dense part of size at least R'/μ. By Corollary 3.1, $\rho_\ell(R') \geq \mu^{\ell-2}/f(\ell)(|R'|)^2$. ∎

Once the Turán Shadow is obtained by `YACC-Shadow-Builder`, the sampling of cliques is carried out in the same manner as TuránShadow.

We restate the `sample` algorithm from [17] here for completeness.

φ denotes what fraction of the sample space consisting of $W(\boldsymbol{S})$ sets of vertices are cliques. Thus, the sampling procedure essentially sets up a Bernoulli distribution over $W(\boldsymbol{S})$ sets and estimates the success probability of this Bernoulli distribution. Note that if $W(\boldsymbol{S}) > 0$, the shadow definitely consists of k-cliques.

Algorithm 2: YACC$(G, k, \varepsilon, \delta, \mu)$

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

Algorithm 3: `sample`$(\boldsymbol{S}, k, \varepsilon, \delta, \gamma)$

$\boldsymbol{S}$ is γ-saturated k-clique shadow
ε, δ are error parameters

For each $(S, \ell) \in \boldsymbol{S}$, set $w(S) = \binom{|S|}{\ell}$
Let $W(\boldsymbol{S}) = \sum_{(S,\ell) \in \boldsymbol{S}} \binom{|S|}{\ell}$
Set probability distribution $\mathcal{D}$ over $\boldsymbol{S}$ where $p(S) = w(S)/W(\boldsymbol{S})$
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, $X_r = 0$
Let $\varphi = \frac{\sum_r X_r}{t}$ denote the fraction of samples that are cliques (we will also call this the success ratio)
Output $\varphi * W(\boldsymbol{S})$ as estimate for $|C_k(G)|$

By Theorem 3.2, the shadow built by YACC is a γ-saturated k-clique shadow. Hence, the results of [17] apply and our final theorem looks as follows:

Theorem 3.3. *Consider graph $G = (V, E)$ with m edges, n vertices, and degeneracy $\alpha(G)$. Let $1 \geq \mu > 0$. Let $\boldsymbol{S}$ be the γ-saturated Turán k-clique shadow of G as output by* `YACC-Shadow-Builder` *where $\gamma = 1/\max_{(S,\ell) \in \boldsymbol{S}} (f(\ell)|S|^2/\mu^{\ell-2})$.*

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

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

Tuning μ: The higher the value of μ, the denser a subgraph has to be before it is put in the shadow and hence, larger the recursion tree. On the other hand, since the subgraphs are denser, the number of samples required to reach a certain level of accuracy is smaller. Thus, there is a tradeoff between the size of the recursion tree to be explored and the number of samples required. The parameter μ helps us control this tradeoff. A small value of μ may require us to take a prohibitively large number of samples. In such a case,

increasing the value of μ can offset the time cost of taking a large number of samples.

4 Experimental results

We implemented YACC in `C++` and ran our experiments on a commodity machine equipped with a 2x Intel Xeon CPU E5-2670 processor with 32 cores and 256KB L2 cache (per core), 20MB L3 cache, and 64GB memory. Our code is available here: https://bitbucket.org/sjain12/yacc/. The codes for PIVOTER [2] and TURÁNSHADOW [1] have been made publicly available by their authors for modification and reproduction for non-commercial use.

We performed our experiments on a collection of real-world graphs from SNAP [36] and the Laboratory for Web Algorithmics [9, 8, 7], consisting of social networks, web networks, etc. For graphs that are directed, we ignore the direction. Basic properties of these graphs are presented in Tab. 1. We fix the number of samples to be 500K for YACC and 50000 for TURÁNSHADOW (based on the value used by its authors in [17]). If and when the paper gets selected for publication, we will be making our code open-source.

We focus on counting k-cliques for k ranging from 10 to 40 and showcase results for $k = 20, 40$.

4.1 Running Time Tab. 1 shows the time in seconds required to get the estimates for $k = 20, 40$ for eight different graphs using YACC. We were able to get estimates for all graphs in < 24 hours, and in many cases in minutes. For many of these graphs, these estimates have become available for the first time.

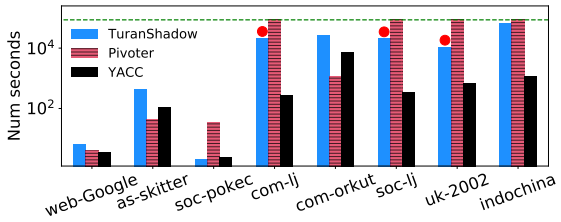
4.2 Accuracy Since YACC is the first algorithm to be able to count k-cliques for several of the graphs, the ground truths for many of our experiments are not available and as a result, we are unable to obtain the exact values of error in the estimates of our randomized algorithm. However, there are other indicators one can measure to gauge the accuracy of the experiments. As explained in [17], the sampling procedure `sample` estimates clique counts by estimating the success probability of a Bernoulli random variable, for which, the algorithm samples several sets of vertices (in this case, 500000). The fraction of these sets that happen to be cliques (which we call the success ratio and denote by φ) gives us an estimate for the fraction of $W(\boldsymbol{S})$ that are cliques. If φ is large then the estimates are more stable across different runs of the algorithm and the error in the estimates is likely to be small. As a practical rule of thumb, if the number of samples that are cliques is > 100, the estimates are likely to have less error. In other words, we want the success ratio from taking 500000 samples to be > 0.0002. When this is not the case and $W(\boldsymbol{S}) > 0$, it means that $W(\boldsymbol{S})$ is much larger than the number of cliques and hence, 500000 samples are not enough to see enough cliques. In this case, we can either increase the number of samples, or rerun the experiment with a higher value of μ. We do so for the graphs in Tab. 1 for which φ is small (and $W(\boldsymbol{S}) \neq 0$), setting $\mu = 1.0$ to get a better success ratio and hence, better estimates. Note that increasing the value of μ comes at the cost of more time. Note also, that the cost of sampling goes up linearly with the number of samples and for graphs like as-skitter and com-orkut, for $k = 40$ we do not know apriori how many more samples we need to take to have enough cliques sampled.

4.3 Convergence Fig. 4i and Fig. 4ii shows convergence over 100 runs of YACC using 50K, 500K and 5M samples for com-lj and soc-lj for $k = 20$, respectively. There is a red dot for the estimate from each of the 100 runs. Since the exact values of these clique counts are not known, we compared the spread of these outcomes with the average of the 100 runs using 5M samples, represented by the blue line. YACC has an extremely low spread and converges quickly.

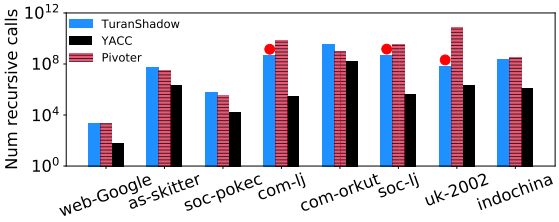
4.4 Size of Recursion Tree Fig. 1ii and Fig. 3ii show the comparison of the sizes of the recursion trees of YACC, TURÁNSHADOW and PIVOTER for several graphs for $k = 20$ and $k = 40$, resp. The red circle represents graphs for which TURÁNSHADOW crashed and we used the size of its recursion tree at the time of crashing. As we can see, in many of the cases, the size of recursion tree decreases by an order of magnitude in the case of YACC. Essentially, this is because of the relaxed stopping criteria and a finer control over the tradeoff of time spent between building the tree and sampling. The time for sampling scales linearly with the number of samples so the reduced tree size comes at the price of more samples. However, in practice this is a negligible cost compared to the time for building the tree; 500K samples (which take < 2 minutes to take) sufficed for all of our experiments involving YACC. As the proof of Theorem 5.4 in [17] shows, the size of the shadow is proportional to the size of the recursion tree and we suspect the TURÁNSHADOW algorithm crashed because of running out of memory.

4.5 Comparison with other Algorithms We compared with TURÁNSHADOW [17] and PIVOTER [18] (codes for both have been made publicly available by the authors) which are the state-of-the-art clique counting algorithms. PIVOTER is an exact clique-counting algorithm and as the authors of PIVOTER point out, it is unable to obtain the counts of cliques for graphs like com-lj. We discovered that this was the case for many more graphs like soc-lj, uk-2002, among others. For such graphs, approximation algorithms are the only feasible solution currently available. Fig. 3i shows the time required by the YACC, TURÁNSHADOW and PIVOTER for estimating the number of 40-cliques. In most cases, YACC takes an order of magnitude less time than other methods. Note that the maximum y value corresponds to 1 day and TURÁNSHADOW and PIVOTER had not terminated in 24 hours for many of the graphs.

Comparison with PEANUTS: [19] proposed a heuristic algorithm for counting cliques that uses TURÁNSHADOW as a subroutine. We replaced the TURÁNSHADOW subroutine with YACC and found significant improvement in the performance of the PEANUTS algorithm. Fig. 5 shows that the time and the number of



(i) Timing in seconds for counting $k = 40$ cliques

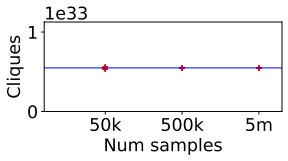


(ii) Size of the recursion tree for $k = 40$

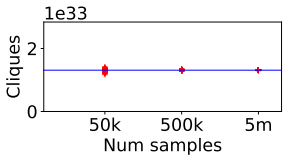
Figure 3: Comparing YACC, TURÁNSHADOW and PIVOTER for estimating the number of 40-cliques.

					k=20				k=40			
graph	**n**	**m**	α	μ	C_k	$W(\boldsymbol{S})$	φ	**time**	C_k	$W(\boldsymbol{S})$	φ	**time**
web-Google	8.7E+05	4.3E+06	44	0.5	6.8E+12	1.5E+13	0.43749	3	9.9E+5	3.7E+08	0.003	4
as-skitter	1.7E+06	1.1E+07	111	0.5	1.45E+18	8.05E+22	1.80E-05	11	0	1.61E+30	0	5
				1.0	1.29E+18	3.01E+19	0.042652	68	5.10E+19	6.25E+21	0.00816	117
soc-pokec	1.6E+06	2.2E+07	47	0.5	4.95E+07	1.24E+12	4.00E-05	11	0	0.00	0	0
				1.0	4.49E+07	1.32E+08	0.33977	12	-	-	-	-
com-lj	4.0E+06	3.4E+07	360	0.5	5.49E+32	6.88E+32	0.797518	79	2.51E+53	6.27+E53	0.399702	344
soc-LJ	4.8E+06	4.2E+07	372	0.5	1.31E+33	2.83E+33	0.463904	172	5.88E+53	5.00E+56	0.001176	384
com-orkut	3.0E+06	1.1E+08	253	0.5	2.11E+17	2.64E+22	8.00E-06	1887	0	1.32E+27	0	824
				1.0	3.38E+17	1.40E+20	2.41E-03	5469	2.61E+13	1.19E+18	2.20E-05	6665
indo-2004	7.4E+06	1.9E+08	13642	0.5	2.92E+58	3.06E+58	0.954504	1233	5.44E+105	6.40E+105	0.849698	1564
uk-2002	1.8E+07	2.9E+08	1885	0.5	1.07E+41	1.07E+41	1	1860	5.29E+70	5.29E+70	1	768

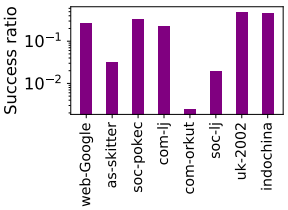
Table 1: Table shows the sizes and degeneracy of the graphs, the counts of 20 and 40 cliques (denoted by C_k), $W(\boldsymbol{S})$, success ratio φ obtained using YACC, and time in seconds required to get the estimates. By default, we used $\mu = 0.5$ and number of samples = 500000. Whenever the success ratio φ was very low (< 0.0002) and $W(\boldsymbol{S}) \neq 0$, we reran the algorithm setting $\mu = 1.0$.



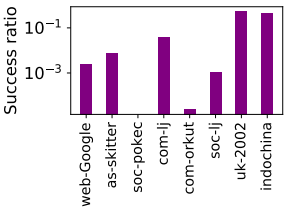
(i) Convergence of YACC for com-lj for $k = 20$



(ii) Convergence of YACC for soc-lj for $k = 20$



(iii) Success ratio k=20



(iv) Success ratio k=40

Figure 4: Convergence and success ratio of YACC.

recursive calls goes down significantly using YACC.

References

[1] https://bitbucket.org/sjain12/cliquecounting/.

[2] https://bitbucket.org/sjain12/pivoter/.

[3] N. K. Ahmed, J. Neville, R. A. Rossi, and N. Duffield. Efficient graphlet counting for large networks. In *Proceedings of International Conference on Data Mining (ICDM)*, 2015.

[4] N. Alon, R. Yuster, and U. 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.

[5] 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.

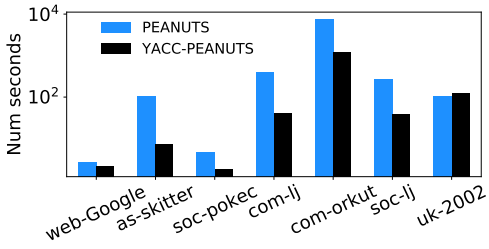
[6] M. Bhuiyan, M. Rahman, M. Rahman, and M. A. Hasan. Guise: Uniform sampling of graphlets for large graph analysis. In *International Conference on Data Mining (ICDM)*, pages 91–100, 2012.

[7] P. Boldi, B. Codenotti, M. Santini, and S. Vigna. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice & Experience*, 34(8):711–726, 2004.

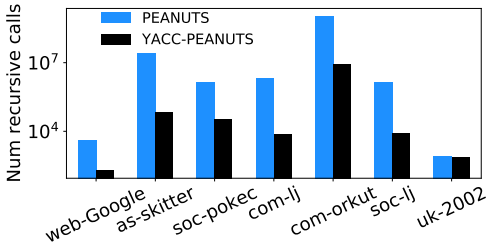
[8] P. Boldi, M. Rosa, M. Santini, and S. Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *World Wide Web (WWW)*, pages 587–596, 2011.

[9] P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. In *World Wide Web Conference (WWW 2004)*, pages 595–601, 2004.

[10] R. Burt. Structural holes and good ideas. *American*



(i) Timing for YACC-PEANUTS



(ii) Recursion tree size for YACC-PEANUTS

Figure 5: Figure shows the time and recursion tree size for PEANUTS and YACC-PEANUTS.

Journal of Sociology, 110(2):349–399, 2004.

[11] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14:210–223, 1985.

[12] M. Danisch, O. D. Balalau, and M. Sozio. Listing k-cliques in sparse real-world graphs. In *World Wide Web (WWW)*, pages 589–598, 2018.

[13] D. Eppstein, M. Löffler, and D. Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *International Symposium on Algorithms and Computation*, pages 403–414. Springer, 2010.

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

[15] I. Finocchi, M. Finocchi, and E. G. Fusco. Clique counting in mapreduce: Algorithms and experiments. *ACM Journal of Experimental Algorithmics*, 20, 2015.

[16] P. Holland and S. Leinhardt. A method for detecting structure in sociometric data. *American Journal of Sociology*, 76:492–513, 1970.

[17] S. Jain and C. Seshadhri. A fast and provable method for estimating clique counts using turán's theorem. In *World Wide Web*, pages 441–449, 2017.

[18] S. Jain and C. Seshadhri. The power of pivoting for exact clique counting. In *Web Search and Data Mining (WSDM)*, pages 268–276, 2020.

[19] S. Jain and C. Seshadhri. Provably and efficiently approximating near-cliques using the turán shadow: Peanuts. In *The Web Conference 2020*, pages 1966–1976, 2020.

[20] M. Jha, C. Seshadhri, and A. Pinar. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *WWW*, pages 495–505, 2015.

[21] 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.

[22] H. Nassar, C. Kennedy, S. Jain, A. R. Benson, and D. Gleich. Using cliques with higher-order spectral embeddings improves graph visualizations. In *The Web Conference 2020*, pages 2927–2933, 2020.

[23] M. Rahman, M. A. Bhuiyan, and M. A. Hasan. Graft: An efficient graphlet counting method for large graph analysis. *IEEE Transactions on Knowledge and Data Engineering*, PP(99), 2014.

[24] R. Rotabi, K. Kamath, J. M. Kleinberg, and A. Sharma. Detecting strong ties using network motifs. In *World Wide Web (WWW)*, pages 983–992, 2017.

[25] A. E. Sariyüce, C. Seshadhri, A. Pinar, and Ü. V. Çatalyürek. Finding the hierarchy of dense subgraphs using nucleus decompositions. In *World Wide Web (WWW)*, pages 927–937. ACM, 2015.

[26] C. Seshadhri, T. G. Kolda, and A. Pinar. Community structure and scale-free collections of Erdös-Rényi graphs. *Physical Review E*, 85(5):056109, May 2012.

[27] C. Seshadhri, A. Pinar, and T. G. Kolda. Wedge sampling for computing clustering coefficients and triangle counts on large graphs. *Statistical Analysis and Data Mining*, 7(4):294–307, 2014.

[28] C. Seshadhri and S. Tirthapura. Scalable subgraph counting: The methods behind the madness: WWW 2019 tutorial. In *The Web Conference (WWW)*, 2019.

[29] J. Shi, L. Dhulipala, and J. Shun. Parallel clique counting and peeling algorithms. In *SIAM Conference on Applied and Computational Discrete Algorithms (ACDA21)*, pages 135–146. SIAM, 2021.

[30] C. E. Tsourakakis, J. Pachocki, and M. Mitzenmacher. Scalable motif-aware graph clustering. In *World Wide Web (WWW)*, pages 1451–1460, 2017.

[31] P. Turán. On an extremal problem in graph theory. *Mat. Fiz. Lapok*, 48(436-452):137, 1941.

[32] J. Ugander, L. Backstrom, and J. M. Kleinberg. Subgraph frequencies: mapping the empirical and extremal geography of large graph collections. In *WWW*, pages 1307–1318, 2013.

[33] H. Yin, A. R. Benson, and J. Leskovec. Higher-order clustering in networks. *Phys. Rev. E*, 97:052306, 2018.

[34] H. Yin, A. R. Benson, and J. Leskovec. The local closure coefficient: A new perspective on network clustering. In *Web Search and Data Mining*, pages 303–311, 2019.

[35] Z. Zhao, G. Wang, A. Butt, M. Khan, V. S. A. Kumar, and M. Marathe. Sahad: Subgraph analysis in massive networks using hadoop. In *Parallel and Distributed Processing Symposium (IPDPS)*, pages 390–401, 2012.

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