A SURVEY OF TECHNIQUES FOR APPROXIMATE SUBGRAPH COUNTING

Richard Gong
gongy@mit.edu

Timothy Leplae-Arthur
tkleplae@mit.edu

Eshaan Nichani
eshnich@mit.edu

1 Introduction

An important problem in the analysis of large graphs G is in counting the number of times some subgraph H appears. This problem has been studied in the case for specific subgraphs such as triangles or cliques which are needed in practical network analysis, as well as in the case where H is arbitrary and we are trying to develop a general purpose bound. Exact counting is often inefficient, particularly when G is large, which motivates us to find approximate counting algorithms which run in time sublinear in the size of G.

In recent literature, approximate counting algorithms rely on randomization in two main ways. The first technique is that of *sparsification*, in which we sparsify our graph by independently keeping each edge with probability p. We form an estimate for the occurrences of H by counting H in the smaller, sparisified graph, and scaling up accordingly. The second technique we consider is *sampling*. In sampling we search for copies of H by sampling sets of vertices/edges which act as candidates for H, and base our estimator on the fraction of these sampled candidates which are isomorphic to H. Algorithms must use a variety of techniques in order to reduce the size of this candidate set or reduce the variance of their estimator.

This paper is organized as follows. We first examine the technique of *sparsification* and motivate its application to counting the number of butterflies and triangles in a graph by studying the respective analyses. Next, we explore the alternative technique of *sampling*, where we see analytical similarities and differences to sparsification when applied to the same problems of counting triangles and butterflies. Finally, we outline a newer approach that extends the approach for triangles to longer odd cycles, followed by a state of the art sampling-based algorithm that generalizes this to approximate the number of occurrences of an arbitrary subgraph.

2 Definitions

We rely on the following notation throughout this paper. Our undirected graph G has vertex set V and edge set E. We define n, m so that $n = |V|$ and $m = |E|$. For a vertex $v \in V$, we denote by d_v the degree of the vertex, by Γ_v the set of v's neighbors, and by E_v the set of edges incident to v. We let Δ denote the maximum degree of the graph.

We seek to count the number of occurences of a subgraph H, with vertex set V_H and edge set E_H. The size of H is assumed to be constant, so we can suppress any asymptotic dependence on H. A subgraph H is also referred to as a *motif*.

We say that some estimate $\hat{x}$ is a $(1 \pm \varepsilon)$-approximation of the true quantity x if $(1-\varepsilon)x \leq \hat{x} \leq (1+\varepsilon)x$. An event occurs *with high probability* if its probability is $1 - \text{poly}(n^{-1})$. We also use $\tilde{O}$ notation to ignore polylog factors of our input, as well as terms polynomial in ε^{-1}.

The following two lemmas will be useful in our analyses:

Lemma 1. *Let Y be a random variable which is an unbiased estimator for the quantity μ, i.e. $\mathbb{E}[Y] = \mu$. Then, we can form a $(1 \pm \varepsilon)$-approximation with high probability for μ with $O(\frac{Var(Y)\log n}{\mu^2\epsilon^2}) = \tilde{O}(\frac{Var(Y)}{\mu^2})$ independent samples of Y.*

Proof. Consider k independent samples $Y_1, \ldots, Y_k$ of Y, and define $Z = \frac{1}{k}\sum_{i=1}^{k} Y_i$. We see that $\text{Var}(Z) = \frac{1}{k}\text{Var}(Y)$, so by Chebyshev's inequality

$$Pr(|Z - \mu| > \varepsilon\mu) \leq \frac{\text{Var}(Z)}{\varepsilon^2\mu^2} = \frac{\text{Var}(Y)}{k\varepsilon^2\mu^2} \leq \frac{1}{8}$$

if we set $k \geq \frac{8\text{Var}(Y)}{\varepsilon^2\mu^2}$. Therefore, with probability $7/8$, Z is a $(1 \pm \epsilon)$ approximation for μ. To amplify this to be a high probability estimate, we create $r = O(\log n)$ independent copies $Z_1, \ldots, Z_r$ of Z. Let C be the number of Z_i which fall in the interval $[(1-\varepsilon)\mu, (1+\varepsilon)\mu]$; C is the sum of r i.i.d Bernoulli random variables with success probability $7/8$, and thus $\mathbb{E}[C] = \frac{7}{8} \cdot r$. By the Chernoff bound,

$$Pr(C \leq \frac{1}{2} \cdot r) \leq \exp(-O(r)) = n^{-O(1)},$$

so with high probability at least half of the Zs - and thus their median - lie in the interval $[(1-\varepsilon)\mu, (1+\varepsilon)\mu]$. Thus w.h.p the median is a $(1 \pm \varepsilon)$-approximation. We require $k \cdot r = O(\frac{\text{Var}(Y)\log n}{\mu^2\epsilon^2})$ samples of Y. □

Remark. *The above proof should be familiar to the 6.856 student, as this lemma was featured on one of our problem sets.*

Lemma 2. *Say we want to estimate the probability p of some event A. If we take k independent samples $X_1, \ldots, X_k$, where X_i is the indicator whether A was observed in sample i, then with high probability the quantity*

$$\hat{p} = \frac{1}{k}\sum_{i=1}^{k} X_i$$

is a $(1 \pm \varepsilon)$-approximation for p if $k = \Theta(\frac{1}{p} \cdot \frac{\log n}{\varepsilon^2}) = \tilde{O}(\frac{1}{p})$.

Remark. *We omit the proof as this lemma was presented in class; it is a simple application of the Chernoff bound.*

Finally, we must define a query model on our graph G. We assume that we can complete the following queries in constant time:

- **Degree query:** Obtain the degree d_v of any vertex v.
- **Neighbor query:** Given a vertex v and an integer $i \leq d_v$, obtain the ith neighbor of v.
- **Pair query:** Given two vertices u, v, test whether $(u, v) \in E$.
- **Edge-sample query:** Sample an edge e uniformly at random from E.

3 Technique 1: Sparsification

3.1 Idea and motivation

Graph sparsification is the concept of extracting information about a graph G by examining a random subgraph $G(p)$ that we construct by including each edge independently with probability p, which is known as the sparsification parameter. Since the expected number of edges in $G(p)$ is mp, this technique produces an approximation algorithm with expected run-time polynomial in mp if we run an exact polynomial (in m) algorithm on the reduced graph $G(p)$.

We have seen graph sparsification used in class to approximate min-cuts quickly, based on the fact that the value of cuts remain close to their expected value in $G(p)$. This idea translates very effectively to the problem of counting subgraphs. Suppose we want to count the number of occurrences of some subgraph H with k edges in G. The probability that a particular occurrence persists in $G(p)$ is p^k since each edge is preserved independently. It follows that if there are t occurrences of H in G, the expected number of occurrences of H in $G(p)$ is tp^k, and hence running an exact counting algorithm for H on $G(p)$ and scaling by p^{-k} acts as an estimator for t.

What determines if this simple estimator is useful? Denote the count of H in $G(p)$ by the random variable C, which is the sum of indicator variables X_i which specify if the i-th occurrence of H in G survives in $G(p)$. Then our estimator for the number of occurrences of H in G, t, is $Y = p^{-k}C$. Since $\mathbb{E}[Y] = p^{-k}\mathbb{E}[C] = t$, Y is an unbiased estimator.

By Lemma 1, we can achieve a $(1 \pm \varepsilon)$ estimator w.h.p with $O(\frac{\text{Var}(Y)\log n}{\mu^2\epsilon^2})$ independent samples of Y. Therefore we are motivated to make the variance of Y small, in order to minimize the number of trials of Y.

Proposition 1. *The variance of Y is positively correlated with the amount of dependence (covariance) between the occurrences of motifs in our input graph, and negatively correlated with the sparsification parameter p.*

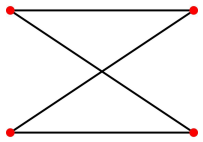


Figure 1: An example of a Butterfly subgraph

Proof.

$$\begin{aligned}
\text{Var}(Y) &= p^{-2k}\text{Var}\left[\sum_i^t X_i\right] \\
&= p^{-2k}\sum_i \text{Var}(X_i) + p^{-2k}\sum_{i,j}\text{Cov}(X_i, X_j) \\
&= p^{-2k}\cdot t(p^k - p^{2k}) + p^{-2k}\sum_{i,j}\mathbb{E}[X_iX_j] - \mathbb{E}[X_i]\mathbb{E}[X_j] \\
&= t(p^{-k}-1) + \sum_{i,j} p^{-2k}\mathbb{E}[X_iX_j] - 1 \qquad (1)
\end{aligned}$$

□

This produces a tradeoff between making p large enough such that our variance is small, but also small enough that our algorithm runs quickly.

The two applications we will examine are similar in that they are mostly useful in real-world data-sets like databases and rely on the number of copies of subgraphs sharing common edges being insignificant. In 3.2, we see a direct approach to choosing p by decomposing the variance as in Proposition 1, while in 3.3, we see an insightful indirect technique for triangles.

3.2 Application to Butterfly Counting in Bipartite Graphs

[1] applies this sparsification technique to counting "butterflies", which are 2×2 bi-cliques (i.e. a 4-cycle), in a bipartite graph G; see Figure 1 for an example. This subgraph has $k = 4$ edges so we use $Y = p^{-4}C$ as an estimator, where C is the exact number of butterflies in $G(p)$. This yields the following simple algorithm for approximately counting butterflies.

Algorithm 1 Approximating Butterflies in Bipartite Graphs with *Sparsification*

procedure BUTTERFLYSPARSIFER(G, ϵ)
  $p \leftarrow \frac{1}{\epsilon^2}\max\left(\sqrt[4]{\frac{24}{t}}, \frac{24p_1}{t}, \frac{\sqrt{24p_2}}{t}\right)$
  $G(p) \leftarrow$ **Sparsify** (G, p)
  $Y \leftarrow p^{-4} \cdot$ **ExactButterfly** $(G(p))$
  return Y

Note that if two butterflies share two vertex-disjoint edges or more than two edges, they are not distinct. Therefore we can classify each pair of distinct butterflies (i, j) as one of 3 types:

- type 0: no shared edge, so X_i and X_j are independent,
- type 1: one shared edge so $\mathbb{E}[X_iX_j] = \text{Pr}[X_i = 1]\text{Pr}[X_j = 1|X_i = 1] = p^7$, or
- type 2: two shared edges with a common endpoint, so $\mathbb{E}[X_iX_j] = \text{Pr}[X_i = 1]\text{Pr}[X_j = 1|X_i = 1] = p^6$.

Let t be the number of butterflies and let p_i be the number of pairs of type i. Then, following from Proposition 1:

$$\text{Var}(Y) = t(p^{-4}-1) + p_1(p^{-1}-1) + p_2(p^{-2}-1) \le tp^{-4} + p_1p^{-1} + p_2p^{-2}.$$

If we set $p > \frac{1}{\varepsilon^2}\max\left(\sqrt[4]{\frac{24}{t}}, \frac{24p_1}{t^2}, \frac{\sqrt{24p_2}}{t}\right)$, then

$$\text{Var}(Y) \leq tp^{-4} + p_1p^{-1} + p_2p^{-2} \leq \varepsilon^8 t \cdot \frac{t}{24} + \varepsilon^2 p_1 \cdot \frac{t^2}{24p_1} + \varepsilon^4 p_2 \cdot \frac{t^2}{24p_2} \leq \frac{\varepsilon^2 t^2}{8}.$$

By Lemma 1, we can therefore obtain a $(1\pm\varepsilon)$-approximation w.h.p from $O(\frac{\varepsilon^2 t^2 \log n}{\varepsilon^2 t^2}) = O(\log n) = \tilde{O}(1)$ independent samples of Y.

Proposition 2. *We have an exact butterfly counting algorithm on a bipartite graph $G = (V = L \cup R, E)$ that runs in $O\left(\sum_{i\in L} d_i^2\right)$ time.*

Proof. For $i, j \in R$ let $n_{i,j}$ denote the number of vertices $k \in L$ adjacent to both i and j. The total number of butterflies is then $\binom{n_{i,j}}{2}$. To compute $n_{i,j}$, iterate through each $k \in L$ and for every pair of edges (i, k) and (j, k), increment $n_{i,j}$. This takes $O\left(\sum_{i\in L} d_i^2\right)$ time. □

Proposition 3. *The expected run-time of the algorithm on the sparsified graph is $O\left(p^2 \sum_{i\in L} d_i^2\right)$.*

Proof. Let D_i be the binomial random variable for the degree of vertex i in the sparisified graph with expectation pd_i and variance $d_ip(1-p)$. Then the run-time is in expectation

$$O\left(\mathbb{E}\left[\sum_{i\in L} D_i^2\right]\right) = O\left(\sum_{i\in L} \text{var}(D_i) + \mathbb{E}\left[D_i\right]^2\right) = O\left(p^2 \sum_{i\in L} d_i^2\right).$$

□

We can upper bound $\sum_i d_i^2$ by $O(mn)$ so the randomized butterfly counting algorithm runs in $O(p^2mn)$. To express in terms of m and n, note that the number of pairs of butterflies sharing an edge is at most $O(n^2t)$, as we can first choose a butterfly and one of its 4 edges, and next choose the two additional vertices. Therefore $p_1, p_2 \leq O(n^2t)$. Plugging in these quantities to get p, we obtain the following bound.

Claim 1. *The approximate butterfly counting algorithm via sparsification runs in time*

$$\tilde{O}(p^2mn) = \tilde{O}\left(\max\left\{\frac{mn}{\sqrt{t}}, \frac{mn^3}{t}, \frac{mn^5}{t^2}\right\}\right).$$

This worst-case bound is not exciting, which makes sense as the authors of [1] were more interested in the practical performance of the algorithm on real datasets with limited dependence (where p_1, p_2 are small) rather than asymptotic guarantees.

3.3 Application to Triangle Counting

We can use an identical sparsification tactic to approximate the number of triangles in a graph:

Algorithm 2 Approximating Triangles with *Sparsification*

```
procedure TRIANGLESPARSIFIER(G, ε)
    p ← (D log n / ε² t)^{1/3}
    G(p) ← Sparsify (G, p)
    Y ← p^{-3} · ExactTriangle (G(p))
    return Y
```

The analysis of this algorithm, however, is more involved. Earlier we were able to develop some intuition about why our attempts to reduce the variance of our estimator caused our sparsification parameter p to be lower bounded by the amount of interaction between the motifs in the graph. In [2], this characteristic naturally carries over to triangle counting, where it is shown that $p^3 \geq \tilde{O}(\frac{D}{t})$ is enough to get a high probability $(1 \pm \varepsilon)$ estimate to t, where D is the maximum number of triangles an edge is contained in and t is the number of triangles. We present this analysis below.

The motivation for the analysis comes from trying to abstract away the dependence between triangles that share edges to remove the covariance part of the bound in equation 1. We do so by partitioning the occurrences of triangles into

sets where no two triangles in a set share an edge. This allows us to use the Chernoff bound on triangles in each set, followed by the union bound to prove a general error bound.

We define the auxiliary graph T such that each triangle in G corresponds to a vertex in T. Two vertices in T have an edge between them if and only if the corresponding triangles in G share an edge. Our partitioning relies on an insightful application of the Hajnal-Szemerédi theorem [3] (Theorem 1) on T. Note that the maximum degree in T is equal to D as defined above, and that $D \leq n$.

Theorem 1. *(Hajnal-Szemerédi) A graph with n vertices and maximum degree Δ can be partitioned into $\Delta + 1$ independent sets of size at least $\lfloor \frac{n}{\Delta+1} \rfloor$.*

Corollary 1. *The triangles can be partitioned into $O(D)$ sets with at least $\Omega(\frac{t}{D})$ independent triangles (no shared edges) in each set.*

Consider such a set of independent triangles of size $k = \Omega(\frac{t}{D})$, and let X_i be the indicator that the ith triangle remains intact after sparsification. By construction the X_i are independent Bernoulli random variables with expectation p^3, and thus we can use the Chernoff bound:

$$\Pr\left[\left|\frac{1}{k}\sum_{i=1}^{k} X_i - p^3\right| > \varepsilon p^3\right] \leq 2e^{-\varepsilon^2 p^3 k/2}$$

If $p^3\varepsilon^2 k \geq 4d \log n$, for some constant d, this probability is upper bounded by n^{-d}. By the union bound, the probability that any set exceeds $(1 \pm \varepsilon)$ multiplicative error is at most $n^{-d}D \leq n^{1-d}$, so the probability that the sum of $\sum_i X_i$ over the sets is within $(1 \pm \varepsilon)$ of its expected value (and thus the probability that our estimator is within $1 \pm \varepsilon$ of its expectation) is at least $1 - n^{-2}$ for $d = 3$. To achieve this high probability of success it is sufficient to have:

$$p^3 \geq O(\frac{\log n}{\varepsilon^2 k}) \geq O\left(\frac{D \log n}{\varepsilon^2 t}\right) = \tilde{O}\left(\frac{D}{t}\right).$$

For such a value of p, the triangle sparsification algorithm is able to produce a $(1 \pm \epsilon)$-approximation w.h.p for t.

Proposition 4. *A simple exact triangle counting algorithm runs in $O(\sum d_i^2)$ on G and $O(p^2 \sum d_i^2)$ on $G(p)$.*

Proof. For each vertex, count the pairs of neighbours that are adjacent. The total divided by 3 is the number of triangles. Notice that the run-time is very similar to butterfly counting (Proposition 2). In fact, the run-time on the sparsified graph shares the analysis in Proposition 3. □

Proposition 4 demonstrates that the runtime of the exact triangle counting algorithm can be upper bounded by $O(mn)$. The sparsification algorithm gives a p^2 speedup, where we take $p = \tilde{O}((\frac{D}{t})^{\frac{1}{3}})$ as shown above.

Claim 2. *The approximate triangle counting algorithm via sparsification runs in time $\tilde{O}(\frac{mn^{5/3}}{t^{2/3}})$.*

4 Technique 2: Sampling

In an attempt to improve the runtime bounds in the previous section we turn to our second theme in approximate subgraph counting, sampling. At its simplest level, the technique of sampling involves picking a random subgraph of G and using copies of H within that subgraph as an estimator.

4.1 Naive Triangle Counting

Let us consider the simplest possible algorithm for estimating the number of triangles in a graph. Consider sampling a triple of vertices at random from the set of vertices of V. We then check whether these vertices form a triangle. If there are t triangles on our graph, the probability we sample a triangle is $q = \frac{t}{\binom{n}{3}}$. Therefore estimating t is equivalent is equivalent to estimating the probability that a randomly sampled triple is a triangle, q. We can apply Lemma 2 and deduce that $\tilde{O}(\frac{1}{q}) = \tilde{O}(\frac{n^3}{t})$ samples are needed to form a $(1 \pm \varepsilon)$-approximation for q w.h.p. Unfortunately if t is small, say $o(n^2)$, then this algorithm will not be sublinear. Can we sample smarter to improve this bound?

4.2 A Better Triangle Counting Algorithm

Motivation for a more efficient sampling algorithm comes from the DNF counting problem [4]. In DNF counting the naive sampling approach was too inefficient due to the relative infrequency of satisfying assignments. Instead they restricted the set from which samples were drawn to a smaller set which still contained all the satisfying assignments; sampling from this smaller set could produce an estimate much more efficiently.

Rather than arbitrarily sampling a triple of vertices, we want to sample from a smaller set of triples S which still contains all triangles. The following technique from [2] demonstrates how to choose a small S. For each u, we define two distinct ways we could construct a set S_u; S is then defined to be $\cup_u S_u$. Given a vertex u, we can form the set S_u of triples (u, v, w) by:

1. Picking v, w to be a pair of neighbors of u. Then $|S_u| = \binom{d_u}{2}$.
2. Picking v, w such that vw is an edge of G. Then $|S_u| = m$.

In both cases S_u contains all triangles which contain u. To minimize $|S|$, we consider the smaller case for each u. If $d_u \leq \sqrt{m}$, then we consider the first case, and otherwise consider the second. This produces Algorithm 3.

Algorithm 3 Approximating Triangles with *Sampling*

procedure TRIANGLESAMPLER(G)
  $s \leftarrow 0$
  for u in V **do**
    if $d_u < \sqrt{m}$ **then**
      $s_u \leftarrow \binom{d_u}{2}$
      $s \leftarrow s + s_u$
    else
      $s_u \leftarrow m$
      $s \leftarrow s + s_u$
  $u \leftarrow$ **VertexSampler** $\left(V, \frac{s_1}{s}, \ldots, \frac{s_n}{s}\right)$ ▷ samples u with probability $\frac{s_u}{s}$
  if $d_u < \sqrt{m}$ **then**
    $v, w \leftarrow (v, w)$ sampled uniformly from pairs of neighbors of u
  else
    $v, w \leftarrow (v, w)$ sampled uniformly from set of edges E
  return True iff $\{u, v, w\}$ forms a triangle in G.

We calculate the size of S. The sum of the size of sets corresponding to the small degree vertices is

$$\sum_{u:d_u \leq \sqrt{m}} |S_u| \leq \sum_{u:d_u \leq \sqrt{m}} d_u^2 \leq 2\sqrt{m} \cdot (\sqrt{m})^2 = 2m^{3/2},$$

where the last inequality is due to $\sum_u d_u = 2m$ and the fact that the sum of squares is maximized when the sum is concentrated in as few terms as possible. For the high degree vertices, since the sum of all degrees is $2m$, there are at most $2\sqrt{m}$ vertices with high degree. Then

$$\sum_{u:d_u \geq \sqrt{m}} |S_u| \leq \sum_{u:d_u \geq \sqrt{m}} m \leq 2\sqrt{m} \cdot m = 2m^{3/2}.$$

Altogether, letting $S = \cup_u S_u$, we have that $|S| \leq 4m^{3/2}$. Now, we can sample triples from S in order to estimate the number of triangles. Our new success probability is $t/|S|$, and therefore by Lemma 2 we can achieve a $(1 \pm \varepsilon)$-approximation with $\tilde{O}(\frac{|S|}{t}) = \tilde{O}(\frac{m^{3/2}}{t})$ samples. There is also a $O(n)$ overhead to compute the $|S_u|$.

Claim 3. *Our approximate triangle counting algorithm via sampling runs in time* $\tilde{O}(n + \frac{m^{3/2}}{t})$.

This improves on the previous triangle counting approximation, and gives motivation for the more sophisticated sampling algorithms we will see next.

4.3 Butterfly Counting via Edge Sampling

Reducing the sample space provides a way to improve upon our naïve sampling algorithms. Another sampling method is to produce estimators with low variance, as we can obtain a strong approximation with fewer samples. This method is illustrated in [1] in estimating the number of butterflies in a bipartite graph.

Our estimator involves the idea of edge sampling, where we pick an edge at random from E, and then exactly count the number of butterflies containing that edge. We can compute this by, for our sampled edge (u, v), looking at Γ_u (which has size d_u) as well as Γ_w for each chosen point $w \in \Gamma_u$ (which has size at most Δ, the max degree). Therefore we can count the number of butterflies in $O(\Delta d_u)$ time, which has expectation $O(\frac{m\Delta}{n})$ when taken over the entire graph.

Our estimate Y for the number of butterflies t is as follows: Pick some edge e uniformly at random for E. Let C be the number of butterflies containing e. Then $Y = \frac{m}{4} \cdot C$. If we define X_i to be the indicator random variable for whether the ith butterfly is incident on e, then we get that $\mathbb{E}[X_i] = \frac{4}{m}$ (there are 4 edges in this butterfly we could choose), and since $C = \sum_{i=1}^{t} X_i, \mathbb{E}[C] = t \cdot \frac{4}{m}$ and thus $\mathbb{E}[Y] = t$. This algorithm is given in Algorithm 4.

Algorithm 4 Approximating Butterflies in Bipartite Graphs with *Sampling*

```
procedure ButterflySampler(G, ε)
    e ← SampleEdge (E)
    C ← ButterflyCount (e)
    Y ← (m/4) C
    return Y
```

Bounding the variance of Y is trickier, and relies on a similar analysis as in Section 3.2. We can expand the variance of Y as follows:

$$\begin{aligned}\text{Var}(Y) &= \text{Var}\left(\frac{m}{4}\sum_{i=1}^{t} X_i\right) \\ &= \frac{m^2}{16}\left[\sum_{i=1}^{t}\text{Var}(X_i) + \sum_{i\neq j}\text{Cov}(X_i, X_j)\right]\end{aligned}$$

First, see that X_i is a Bernoulli random variable with probability $4/m$, and thus $\text{Var}(X_i) = \frac{4}{m}(1 - \frac{4}{m}) \leq \frac{4}{m}$. To bound the covariances we split pairs of distinct butterflies (i, j) up into the types defined in Section 3.2.

1. If (i, j) is type 0, then X_i and X_j cannot share an edge and thus $X_iX_j = 0$. Therefore $\text{Cov}[X_iX_j] = \mathbb{E}[X_iX_j] - \mathbb{E}[X_i][X_j] = -\frac{16}{m^2} < 0$.
2. If (i, j) is type 1, then X_i and X_j share one edge. $X_iX_j = 1$ if this edge is chosen, which occurs with probability $1/m$, and thus $\mathbb{E}[X_iX_j] - \mathbb{E}[X_i][X_j] = \frac{1}{m} - \frac{16}{m^2} < \frac{1}{m}$
3. If (i, j) is type 2, then X_i and X_j share two edges. $X_iX_j = 1$ if one of these edges is chosen, which occurs with probability $2/m$, and thus $\mathbb{E}[X_iX_j] - \mathbb{E}[X_i][X_j] = \frac{2}{m} - \frac{16}{m^2} < \frac{2}{m}$.

If p_1, p_2 are the number of type 1, type 2 pairs, and $p_e = p_1 + p_2$ is defined as the number of pairs of butterflies sharing an edge, then

$$\text{Var}(Y) \leq \frac{m^2}{16}\left[\frac{4}{m}t + \frac{1}{m}p_1 + \frac{2}{m}p_2\right] \leq \frac{m}{4}(t + p_e).$$

We can now invoke Lemma 1, and thus obtain a $(1 \pm \epsilon)$-approximation with $\tilde{O}(\frac{\text{Var}(Y)}{t^2}) = \tilde{O}(\frac{m}{t}(1 + \frac{p_e}{t}))$ samples. Each sample requires a runtime of $O(\frac{m\Delta}{n})$, so the entire algorithm runs in time $\tilde{O}(\frac{m^2\Delta}{nt}(1 + \frac{p_e}{t}))$.

As with the previous algorithms, the runtime of this algorithm is very dependent on specific properties of graphs such as maximum degree and p_e, which may be small in practice (and in fact [1], from which this algorithm was taken, wasn't so much concerned with asymptotic runtime but rather its performance on practical datasets). We, however, want to upper bound on this quantity as a point of comparison. The maximum degree Δ can be $\Theta(n)$, and as discussed in Proposition 3, $p_e \leq \Theta(n^2t)$. Substituting, we get the following:

Claim 4. *Our approximate butterfly counting algorithm via sampling runs in time* $\tilde{O}(\frac{m^2n^2}{t})$.

5 An Algorithm for Approximately Counting Arbitrary Subgraphs

5.1 A naïve approach

Up until now our algorithms for approximately counting the number of subgraphs have focused on very specific subgraphs, such as the triangle or the butterfly. One interesting question is whether a general purpose algorithm or

bound exists for arbitrary subgraphs H of constant size. One potential issue with the approaches we've used thus far is that in proving bounds on the variance we need some way to analyze the dependence two copies of the subgraph have on each other. While for butterflies and triangles there were a fixed number of ways two subgraphs could interact which could be analyzed by casework, there are no general statements we can make about an arbitrary subgraph. This suggests that we may need a different method of analysis.

A naïve approach to estimating the number of occurrences of H is based on that of Section 4.1, where we sample a set of $|V_H|$ vertices from V, called *candidate subgraph*. Then, we check if the appropriate edges exist between vertices of this candidate to form a copy of H in G. We call such a candidate *valid*. If there are t copies of H then our probability of *success*, defined as our candidate being valid, is $t/\binom{n}{|V_H|}$, and thus by Lemma 2 we require $\tilde{O}(\frac{n^{|V_H|}}{t})$ samples to achieve a $(1 \pm \varepsilon)$-approximation. Unfortunately this quantity grows much larger than sublinear. Clearly, this method will not work for general subgraphs, so we must explore a different approach.

5.2 Motivation for Improvement

One motivating idea is the AGM bound, which allows us to decompose arbitrary H into a combination of smaller subgraphs which we sample instead of vertices. Before presenting the bound we consider the following definitions:

Definition 1. *Assign a real weight between* 0 *and* 1 *to each edge of* H *such that for each vertex, the sum of the weights on edges incident to that vertex is at least* 1*. The fractional edge cover* $\rho(H)$ *is defined to be the minimum possible total weight of all edges over all such weight assignments.*

Equivalently, $\rho(H)$ is the optimum of the following LP:

$$\begin{aligned} \rho(H) = \text{min.} \sum_{e \in E_H} x_e \\ \text{s.t.} \sum_{e \in E_H : v \in e} x_e \geq 1 \forall v \in V \\ 0 \leq x_e \leq 1 \end{aligned} \tag{2}$$

It is easy to deduce that the fractional edge cover of the triangle is 1.5 (by setting x_e for each edge to 0.5) and that the fractional edge cover of the butterfly is 2 (by setting the weight of two non-adjacent edges to 1).

Theorem 2. *(AGM Bound) The number of copies of the subgraph* H *within* G *is at most* $O(m^{\rho(H)})$.

The proof of the AGM bound is beyond the scope of this paper. Since this gives an upper bound on the number of copies of H, we may hope to utilize this bound to improve our sampling algorithm. In Section 4.2 we discussed how reducing the sample space can improve efficiency; therefore, if we could somehow utilize the AGM bound to construct a sample space of size $O(m^{\rho(H)})$ from which we sample *candidates* (as defined in 5.1) for H we could build a more efficient sampler, as we could estimate the number of occurrences from the fraction of valid candidates. If the number of occurrences of H is $\#H$, then the success probability of a candidate would be $O(\frac{\#H}{m^{\rho(H)}})$, and therefore by Lemma 2 we could form a $(1 \pm \varepsilon)$ approximation in at most $\tilde{O}(\frac{m^{\rho(H)}}{\#H})$ queries. The question then becomes: how would we construct such a set of candidates and sample from it efficiently?

One promising idea comes from the following lemma on the decomposition of the fractional edge cover.

Lemma 3. *For any subgraph* H *and its corresponding LP (Equation 2), there exists an optimal solution* x *such that its support*[1] *supp*(x) *is the union of vertex disjoint odd cycles and stars such that*

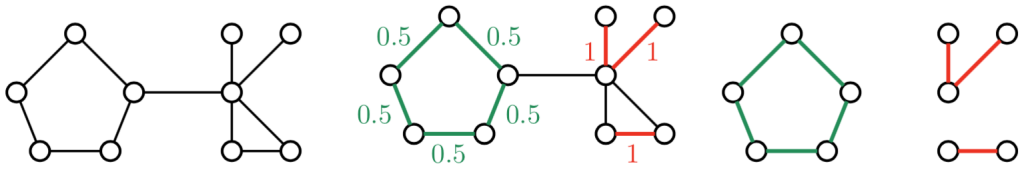
- *For odd cycles* $C \in$ *supp*(x), $x_e = 1/2$ *for all* $e \in C$
- *For stars* $S \in$ *supp*(x), $x_e = 1$ *for all* $e \in S$,

where a star is a subgraph with one center vertex u*,* l *leaf vertices* $v_1, \ldots, v_l$*, and edge set* $\{(u, v_i) \forall i\}$.

The proof of this lemma comes from standard results in Linear Programming stating that the optimum of a LP must be at a vertex.

This lemma tells us that we can decompose H into vertex disjoint odd cycles $C_1, \ldots, C_o$ and stars $S_1, \ldots, S_s$ as depicted in Figure 2. This decomposition is very naturally related to the fractional edge cover; by definition we have that

[1]The support supp(x) of x is the set of edges e such that $x_e > 0$.



(a) The subgraph H. **(b)** An optimal edge-cover of H with $\rho(H) = 5.5$. **(c)** Decomposition of H.

Figure 2: (a) An example of a subgraph H. (b) Its fractional edge cover ρ. (c) A decomposition of H into an odd cycle of size 5, a star with 1 leaf, and a star with 2 leaves. This figure is from [5].

$\rho(H) = \sum_{i=1}^{o} \rho(C_i) + \sum_{j=1}^{s} \rho(S_j)$. The AGM bound tells us that there are at most $O(m^{\rho(C_i)})$ copies of C_i and at most $O(m^{\rho(S_j)})$ copies of S_j in G. If we can somehow sample $O(m^{\rho(C_i)})$ candidates for C_i and $O(m^{\rho(S_j)})$ candidates for S_j, we could sample a candidate for H by taking the union of sample candidates for odd cycles isomorphic to $C_1, \ldots, C_o$ and stars isomorphic to $S_1, \ldots, S_s$. This would give $O(\prod_{i=1}^{o} m^{\rho(C_i)} \cdot \prod_{j=1}^{s} m^{\rho(S_j)}) = O(m^{\rho(H)})$ total candidates for H, which we desired. We then check if each if candidate is *valid*; that is, forms a valid copy of H.

Altogether, this suggests the following algorithm for counting copies of H. We decompose H into cycles and stars via the fractional edge cover $\rho(H)$, and using efficient algorithms for sampling cycles and stars, sample potential candidates for H. Such a problem is appealing as it has reduced the complicated problem of counting arbitrary H to a much simpler one, sampling odd cycles and stars.

In [5], the authors present an elegant algorithm for doing such a sampling and thus provide a general algorithm for estimating the number of subgraphs in $\tilde{O}(\frac{m^{\rho(H)}}{\#H})$ time. Similar to the previous sections, this algorithm relies on constructing a low-variance estimator in addition to the method of sampling from a candidate set. We outline the algorithm in the special cases of the odd cycle and the star separately before analyzing the estimator for arbitrary graphs.

5.3 The Odd-Cycle Sampler

Algorithm 5 The Candidate Odd-Cycle Sampler

procedure CANDIDATEODDCYCLESAMPLER(G, C_{2k+1})
  $(e_1, \ldots, e_k) \leftarrow$ **EdgeSampler** (G, k) ▷ samples $(e_1, \ldots, e_k)$ randomly with replacement
  $d \leftarrow d_{u_1}$ ▷ $e_1 = (u_1, v_1)$
  for $i = 1, \ldots, r = d/\sqrt{m}$ **do**
    $w_i \leftarrow$ **VertexSampler** (Γ_{u_1}) ▷ Samples random vertex from Γ_{u_1}
  return $\{(w_1, u_1, v_1, \ldots, u_k, v_k), (w_2, u_1, v_1, \ldots, u_k, v_k), \ldots, (w_r, u_1, v_1, \ldots, u_k, v_k)\}$, a set of r candidates for C_{2k+1}.

We first present a method to count odd cycles in a graph by sampling *candidates* for odd cycles of length $2k+1$ as follows and estimating the fraction of valid candidates. First sample with replacement k directed edges $\mathbf{e} := (\overrightarrow{e_1}, \ldots, \overrightarrow{e_k})$ (by first picking an undirected edge from E and then selecting its direction), where $\overrightarrow{e_i} = (u_i, v_i)$, with the constraint that $d_{u_1} < d_{v_1}$. Let $r = \lceil d_{u_1}/\sqrt{m} \rceil$. Sample with replacement r vertices $(w_1, \ldots, w_r)$ from the neighbours of u_1. Append each to a copy of $\mathbf{e}$ to make r odd-cycle candidates in the form $(w_i, u_1, v_1, \ldots, u_k, v_k)$ for $1 \leq i \leq r$. Recall from 5.1 that the i-th candidate is *valid* if the edges of the subgraph $(w_i, u_1), (u_1, v_1), \ldots, (v_k, w_i)$ exist. For each candidate i, define the random variable X_i such that if i is a valid candidate, then $X_i = 1/\Pr[\text{candidate } i \text{ chosen}] = d_{u_1} \cdot (2m)^k/2$, the inverse probability of i being sampled as a candidate. Otherwise, $X_i = 0$. The estimator returns the mean $Y = \frac{1}{r} \sum_{i \in [r]} X_i$.

Lemma 4. $\sum_{(u,v) \in E} \min(d_u, d_v) \leq 5m^{3/2}$.

This result is from [5] and its proof is beyond the scope of this paper. It follows that $\mathbb{E}[d_{u_1}] = 5\sqrt{m}$, and thus

Corollary 2. *The expected number of candidates* $\mathbb{E}[r]$ *is* $O(1)$.

From this corollary, we verify the validity of a constant number of candidates. For each candidate, we check $O(k) = O(1)$ edges, so the run-time is constant.

Lemma 5. $\mathbb{E}[Y] = \#C_{2k+1}$, *the number of cycles of length* $2k+1$, *and var*$(Y) \leq (2m)^k\sqrt{m} \cdot \mathbb{E}[Y]$.

Proof. We use the law of total expectation and only enumerate over candidates with non-zero X_i. Note that the probability cancels with its reciprocal, X_i.

$$\mathbb{E}[Y] = \mathbb{E}[X_i] = \sum_{\text{any candidate } i} \Pr[\text{candidate } i \text{ chosen}] \cdot X_i = \sum_{\text{successful candidate } i} 1 = \#C_{2k+1}.$$

Next, we use the law of total variance to write $\text{var}(Y) = \mathbb{E}[\text{var}(Y|\mathbf{e})] + \text{var}(\mathbb{E}[Y|\mathbf{e}])$ and bound each of the terms separately. First, since X_i's are conditionally independent and identically distributed given $\mathbf{e}$,

$$\mathbb{E}[\text{var}(Y|\mathbf{e})] = \frac{2}{(2m)^k} \sum_{e \in \vec{E}^k} \text{var}(Y|\mathbf{e}) = \frac{2}{(2m)^k} \sum_{e \in \vec{E}^k} \frac{1}{t^2} \cdot t \cdot \text{var}(X_1|\mathbf{e}) \leq \frac{2}{(2m)^k} \sum_{e \in \vec{E}^k} \frac{1}{t} \cdot \mathbb{E}[X_1^2|\mathbf{e}].$$

Now we bound $\mathbb{E}[X_1^2|\mathbf{e}]$. Let $W_{\mathbf{e}}$ be the subset of the neighbours w of u_1 such that $(\mathbf{e}, w)$ is a valid candidate. Then,

$$\mathbb{E}[X_1^2|\mathbf{e}] = \sum_{w \in W_{\mathbf{e}}} \frac{1}{d_{u_1}} X_i^2 = \sum_{w \in W_{\mathbf{e}}} \frac{1}{d_{u_1}} \left(\frac{d_{u_1}(2m)^k}{2} \right)^2 \leq \frac{(2m)^{2k}}{4} \cdot d_{u_1} \cdot |W_{\mathbf{e}}|.$$

Now, $\sum_{\mathbf{e}} |W_{\mathbf{e}}|^2 \leq (\sum_{\mathbf{e}} |W_{\mathbf{e}}|)^2 = (\#C_{2k+1})^2$, and the AGM bound gives $\#C_{2k+1} \leq m^k\sqrt{m}$, so

$$\text{var}(\mathbb{E}[Y|\mathbf{e}]) = \text{var}(|W_{\mathbf{e}}|) \leq \mathbb{E}[|W_{\mathbf{e}}|^2] \leq \frac{2}{(2m)^k} \sum_{\mathbf{e}} |W_{\mathbf{e}}|^2 \leq \frac{2(\#C_{2k+1})^2}{(2m)^k} \leq \#C_{2k+1}\sqrt{m}.$$

Substituting gives

$$\begin{aligned} \text{var}(Y) = \mathbb{E}[\text{var}(Y|\mathbf{e})] + \text{var}(\mathbb{E}[Y|\mathbf{e}]) &\leq \frac{(2m)^k \#C_{2k+1}\sqrt{m}}{2} + \#C_{2k+1}\sqrt{m} \\ &\leq (2m)^k \#C_{2k+1}\sqrt{m} = (2m)^k\sqrt{m} \cdot \mathbb{E}[Y]. \end{aligned}$$

□

Applying Lemma 1, we obtain a $(1 \pm \varepsilon)$-approximation w.h.p for $\#C_{2k+1}$ using $\tilde{O}(\frac{\text{Var}(Y)}{\#C_{2k+1}^2}) = \tilde{O}(\frac{m^{k+1/2}}{\#C_{2k+1}})$ samples.

5.4 The Star Sampler

Algorithm 6 The Candidate Star Sampler

```
procedure CandidateStarSampler(G, S_l)
    v ← VertexDegreeSampler (G)          ▷ samples v with probability d_v/2m
    d ← d_v
    for i = 1, ..., l do
        w_i ← VertexSampler (Γ_v)        ▷ Samples random vertex from Γ_v
    return (l, w_1, ... w_l), a candidate for S_l
```

We want to count the number of stars with l edges. First sample a vertex v with probability $d_v/2m$. Take l vertices $\mathbf{w} = (w_1, \ldots, w_l)$ from its neighbours uniformly at random without replacement. The union of these vertices forms the candidate star, and let the random variable X satisfy $X = 1/\Pr[\text{choose this candidate}]$ if the candidate is valid, and $X = 0$ otherwise. Note that it is valid if and only if v has l neighbours.

Lemma 6. $\mathbb{E}[X] = \#S_l$, *the number of stars with* l *leaves, and Var*$(X) \leq 2m^l \cdot \mathbb{E}[X]$.

Proof. The expectation follows similarly to the proof of Lemma 5.

$$\mathbb{E}[X] = \sum_{v, d_v \geq l} \Pr[\text{choose } v] \cdot \sum_{\mathbf{w}} \frac{1}{\Pr[\text{choose } v \text{ and } \mathbf{w}]} \cdot \Pr[\text{choose } \mathbf{w} \mid \text{v}] = \sum_{v, d_v} \sum_{\mathbf{w}} 1 = \#S_l.$$

We can also bound the variance as follows,

$$\text{var}(X) \leq \mathbb{E}[X^2] = \sum_{v, d_v \geq l} \sum_{\mathbf{w}} \frac{d_v}{2m} \cdot \frac{1}{\binom{d_v}{l}} \cdot \left(\frac{2m}{d_v} \cdot \binom{d_v}{l} \right)^2 \leq 2m^l \cdot \#S_l = 2m^l \cdot \mathbb{E}[X].$$

□

Lemma 1 yields a high probability, $(1 \pm \varepsilon)$-approximation in $\tilde{O}(\frac{m^l}{\#S_l})$ samples.

5.5 The Subgraph Sampler

We have shown how to obtain a decomposition of H into stars and odd-cycles, $D(H) = \{C_1, \ldots, C_o, S_1, \ldots, S_s\}$. We have also presented how to sample and estimate the counts of odd-cycles and stars. We want to somehow use our odd-cycle and star sampler to estimate $\#H$. To gain some intuition, suppose that $D(H) = \{C\}$ for some odd cycle C. In this case, we can use our odd-cycle sampler to obtain a candidate cycle of length $|C|$. Then, we can complete an additional check to verify that the edges of H not included in the decomposition, $E_H \setminus D(H)$, are contained in the subgraph of G induced by our candidate. This presents a method for sampling candidates of H.

Algorithm 7 The Candidate Subgraph Sampler

procedure CANDIDATESUBGRAPHSAMPLER($j, G, C_j, \ldots, C_o, S_1, \ldots, S_s$)
  $B \leftarrow \{\emptyset\}$
  if $j < o$ **then**
    $\mathbf{e}, w_1, \ldots, w_r \leftarrow$ **CandidateOddCycleSampler** (G, C_j)
  else
    $\mathbf{e}, w_1, \ldots, w_r \leftarrow$ **CandidateStarSampler** (G, S_{j-o})
  for $i = 1, \ldots, r$ **do**
    $B_i \leftarrow \{\emptyset\}$
    for b in **CandidateSubgraphSampler** $(j+1, G, C_{j+1}, \ldots, C_o, S_1, \ldots, S_s)$ **do**
      $B_i \leftarrow B_i + (\mathbf{e}, w_i, b)$
    $B \leftarrow B + B_i$
  return S

Such an algorithm generalizes to sampling candidates of arbitrary H. We can sample candidates for odd cycles $C_1, \ldots, C_o$ and stars $S_1, \ldots, S_s$ in order to obtain a candidate for H. We can then verify whether these candidates do, in fact, make vertex disjoint cycles and stars, and then finally check whether the edges of H not included in the decomposition exist in the subgraph of G induced by our sampled cycles and stars - this tells us whether or not our candidate was a successful copy of H.

The algorithms for estimating the counts of odd-cycles and stars relied on the definition of a random variable which was 0 for failed candidates and equal to the inverse probability of being sampled for successful candidates; we wish to do something similar for the general graph. In order to demonstrate the full counting algorithm, we must introduce the Subgraph Sampler Recursion Tree.

5.5.1 The Subgraph Sampler Recursion Tree

To set the stage for the description of the recursive tree, we first visualize the outputs of the odd-cycle sampler and the star sampler in terms of *elementary* trees. Recall that for the odd-cycle sampler the potential cycles are $(\mathbf{e}, w_i)$ for $i = 1, \ldots, r$. We define the corresponding tree as illustrated in figure $3b$ whose root contains the edges $\mathbf{e}$ and the ith children containing the node w_i for $i = 1, \ldots, r$. Similarly, for star-cycle sampler, there is only one potential star $(v, \mathbf{w})$. We define the corresponding tree as illustrated in figure $3c$ and $3d$ whose root contains the vertex v and whose one child contains the vertices $\mathbf{w}$. Given these trees, we now present the subgraph sampler tree.

First, we consider the odd cycles $C_1, \ldots, C_o$ in $D(H)$. We initialize the tree by calling the cycle-sampler on C_1 to obtain the cycle sampler tree. For each leaf-node α we *independently* run cycle sampler on C_2 to obtain a new tree, and concatenate the corresponding tree to α thereby extending the height of the tree by 2. We repeat this recursively for all odd cycles in $D(H)$. Once we have iterated over all the odd cycles, we switch to processing stars. For each leaf node α of the current tree, we independently run star sampler on S_1 and concatenate the star tree at the end of each leaf, extending the height of the tree by 2. We do this for all stars. At termination, the height of the tree is $2s + 2o$. Figure $3a$ shows an example of such a tree.

To build our estimator, note that each path from the root to a leaf α corresponds to a set of odd-cycle and star candidates, and thus a candidate for H, which we can check if is a copy of H. Similar to the analysis for odd-cycles and stars, we can define the random variable X_α to be 0 if this is an unsuccessful candidate of H and $1/P[\alpha]$ otherwise, where $P[\alpha]$ is the probability that this particular set of odd-cycles and stars was sampled over the successive iterations of our intermediate samples. Finally, we let $Y = \frac{1}{T}\sum_{\alpha=1}^{T} X_\alpha$, where the sum is taken over all leaves of our tree, be our estimator for $\#H$. Remember that since H is a graph of constant size, we can compute the value of our estimator Y in constant time.

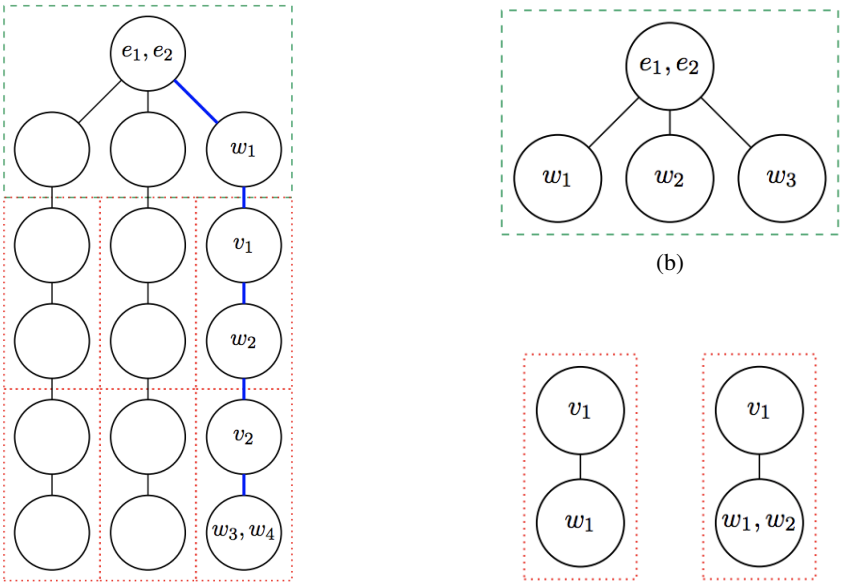


Figure 3: (a) shows the recursion tree of **CandidateSubgraphSampler** (C_5, S_1, S_2). (b) shows the recursion tree of **CandidateOddCycleSampler** (C_5). (c) shows the recursion tree of **CandidateStarSampler** (S_1). (d) shows the recursion tree of **CandidateStarSampler** (S_2). Figure from [5].

In order for Y to be a good estimator it, as in the previous sections, must be unbiased and have low variance. The following two lemmas from [5] show that this is indeed the case.

Lemma 7. *For the random variable Y for the subgraph-sampler, $\mathbb{E}[Y] = \#H$*

Proof Sketch 1. *This proof relies on the same ideas as Lemmas 5 and 6. We can define the value of a leaf node α to be X_α, and the value of an internal node V to be the average of the values of its children multiplied by the probability of reaching this node. We can then show via induction that $\mathbb{E}[V]$ for an internal node V is the number of copies of H containing the cycles and stars in the path from the root to the node H.*

Lemma 8. *For the random variable Y for the subgraph-sampler, $Var[Y] = O(m^{\rho(H)})\mathbb{E}[Y]$*

Proof Sketch 2. *We can again use induction, splitting up the desired variance based on the Law of Total Variance as in Lemmas 5 and 6; this proof is quite mathematically involved and can be viewed in its totality in [5].*

We can use the preceeding two lemmas in conjunction with Lemma 1 to obtain our final, $(1 \pm \varepsilon)$-approximation for $\#H$ with high probability using $\tilde{O}(\frac{\text{Var}(Y)}{\#H^2}) = \tilde{O}\left(\frac{m^{\rho(H)}}{\#H}\right)$.

Claim 5. *The general algorithm for counting arbitrary subgraphs via sampling runs in time $\tilde{O}\left(\frac{m^{\rho(H)}}{\#H}\right)$.*

One original goal was to produce a sublinear algorithm, and we see that this is achieved if $\#H = \Omega(m^{\rho(H)-1})$. This ends up being true in most graphs, where the AGM bound is close to tight.

Let us compare our runtime to our previous algorithms for triangle and butterfly counting. $\rho(H)$ for a triangle and butterfly are $3/2$ and 2, and thus our general purpose algorithm runs in $O(\frac{m^{3/2}}{t})$ time for the triangle and $O(\frac{m^2}{t})$ time for the butterfly, which are superior to the results derived earlier. In fact, in [5] it was shown that this bound is optimal for large classes of graphs.

6 Conclusion

In this paper we have surveyed a variety of techniques for estimating the number of occurrences of a subgraph in a larger graph. We focused on two main approximation techniques - sparsification and sampling - and analyzed their effectiveness on a variety of specific subgraphs such as triangles and butterflies. Finally, we presented a general purpose algorithm for estimating the number of occurrences of an arbitrary subgraph [5], which was shown to be optimal in this setting.

References

[1] A. E. Sariyuce S.-V. Sanei-Mehri and S. Tirthapura. Butterfly counting in bipartite networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, page 2150–2159. ACM, 2018.

[2] R. Peng M. Kolountzakis, G. Miller and C. Tsourakakis. Efficient triangle counting in large graphs via degree-based vertex partitioning. *Algorithms and Models for the Web-Graph*, pages 15–24, 2010.

[3] András Hajnal and E Szemerédi. Proof of a conjecture of p. erdős. *Colloq Math Soc János Bolyai*, 4, 01 1970.

[4] M. Luby R.M. Karp and N. Madras. Monte carlo approximation algorithms for enumeration problems. In *Journal of Algorithms 10*, page 429–448, 1989.

[5] M. Kapralov S. Assadi and S. Khanna. A simple sublinear-time algorithm for counting arbitrary subgraphs via edge sampling. *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*, pages 6:1–6:20, 2019.