Sampling Tools for Analyzing Triangles in Large Networks

Raunak Kumar

April 21, 2019

Contents

1 Introduction **2**

2 Geometric Mean of Edge Weights **2**

2.1 Problem 2

2.2 Algorithm 2

2.3 Analysis 3

2.4 Sampling in $O(\log(\text{degree}))$ - Rejection Sampling 5

2.5 Sampling in $O(\log(\text{degree}))$ - Alternative 5

3 Arithmetic and Harmonic Mean of Edge Weights **6**

3.1 Problem 6

3.2 Algorithm 6

3.2.1 Approach 1 6

3.2.2 Approach 2 6

3.2.3 Approach 3 6

3.3 Analysis 7

4 Number of Simplices (Closed Triangles) **7**

4.1 Problem 7

4.2 Algorithm 7

4.3 Analysis 7

5 Number of Simplices (All Triangles) 9

5.1 Problem 9

5.2 Algorithm 9

5.3 Analysis 9

6 Common 4^{th} Neighbor 10

6.1 Problem 10

6.2 Algorithm and Analysis 10

1 Introduction

Analyzing triangles is an important task in network analysis. We present some sampling based algorithms for tackling such problems; e.g. estimating the top-k triangles in a weighted, undirected graph ordered by the mean of the edge weights, estimating the top-k triangles in a weighted, undirected graph ordered by the number of simplices the triangles appear in, etc.

2 Geometric Mean of Edge Weights

2.1 Problem

Given a weighted graph $G = (V, E)$ and an integer k, output a list of the top-k triangles ordered by the mean of their edge weights. For now, we will focus on simple undirected graphs with positive edge weights, and we will consider the case where we order triangles according to the *geometric* mean of the edge weights.

Given an edge $e = (u, v) \in E(G)$, let $w_{uv} = w_e$ denote the weight of this edge. The geometric mean of a triangle defined by vertices a, b, c is $(w_{ab}w_{bc}w_{ca})^{1/3}$. Since the cube is a monotonic function, we will instead consider ordering the triangles by the cube of their geometric means, $w_{ab}w_{bc}w_{ca}$, for simplicity.

2.2 Algorithm

Our overall approach is similar to diamond sampling for the maximum all-pairs dot-product search problem [1]. While the overall approach is similar we do not have an analogue of a "diamond" in our case and thus, the dependency on the number of samples needed is worse than in [1]. Nevertheless, our approach is a promising one and improving upon it using an analogue of a "diamond" could be the next step.

At a high level, we sample edges $(a, b), (a, c)$ and (b, c') such that $b \neq c$ and $a \neq c'$ with probability proportional to the product of their weights, $w_{ab}w_{bc'}w_{ca}$. Thus, these edges will either form a 3-path or a triangle. If $c \neq c'$, then these edges do not form a triangle and we discard this sample. If $c = c'$, then these edges form a triangle and we increment a counter x_{abc} associated with this triangle. Intuitively, we expect that triangles whose product of edge weights is higher will have a higher counter value, and we can output the triangles associated with the top-k counters.

For a vertex $a \in V$ let $N(a)$ denote the set of its neighbors in G, ie. $N(a) = \{b \in V : (a, b) \in E\}$. Set

$$\tilde{p}_{ab} = w_{ab} \sum_{a' \in N(a) \setminus b} w_{aa'} \sum_{b' \in N(b) \setminus a} w_{bb'} \tag{1}$$

$$Z = \sum_{(a,b) \in E} \tilde{p}_{ab} \tag{2}$$

$$Z(a, b) = \sum_{a' \in N(a) \setminus b} w_{aa'} \tag{3}$$

Algorithm 1: Sampling Triangles

Input : An undirected, weighted graph $G = (V, E)$, number of samples s.
Output: Counters x initialized to 0.

for $l = 1 \dots s$ **do**
  Sample edge (a, b) with probability $\tilde{p}_{ab}/Z$.
  Sample $c \in N(a) \setminus b$ with probability $w_{ac}/Z(a, b)$.
  Sample $c' \in N(b) \setminus a$ with probability $w_{bc'}/Z(b, a)$.
  $x_{abc} \leftarrow x_{abc} + I[c = c']$.
end
return x

Algorithm 2: Postprocessing

Input : Counters x, a budget $k' \in Z_+$, number of triangles $k \in Z_+$ with $k \leq k'$.
Output: List of top-k triangles.

$J \leftarrow$ the top-k' counters from x.
$S = \emptyset$.
for $x_{abc} \in J$ **do**
  $S \leftarrow S \cup (w_{ab} w_{bc} w_{ca}, \{a, b, c\})$.
end
return *Top-k triangles from S.*

2.3 Analysis

A naïve algorithm for this problem might be to sample triangles uniformly at random and then pick the top-k of them based on the product of their edge weights. However, intuitively we would expect that sampling triangles proportional to the product of their edge weights would perform better. We will show that this is indeed the case. Recall that x_{abc} is a counter associated with the triangle formed by vertices a, b and c and represents the number of times this triangle was sampled. We will show that the expected value of x_{abc} is equal to the product of the edge weights of the corresponding triangle upto a multiplicative factor. Then, we will show that depending on the desired error and error probability, if we take a sufficient number of samples, then this expectation is tightly concentrated. Our analysis is very similar to that in [1].

Lemma 1. *Let $\delta_{abcc'}$ be the event of choosing edges $(a, b), (a, c)$ and (b, c'). Then, $P(\delta_{abcc'}) = w_{ab} w_{bc'} w_{ca}/Z$.*

Proof. Event $\delta_{abcc'}$ occurs when we first choose edge (a, b), and conditioned on that we choose edges (a, c)

and (b, c'). Thus, using equations 1, 2 and 3 we get

$$\begin{aligned}
P(\delta_{abcc'}) &= P(\text{choosing } (a,b))\ P(\text{choosing } (a,c) \mid (a,b))\ P(\text{choosing } (b,c') \mid (a,b)) \\
&= \frac{\tilde{p}_{ab}}{Z} \frac{w_{ac}}{Z(a,b)} \frac{w_{bc'}}{Z(b,a)} \\
&= \frac{w_{ab} \sum_{a' \in N(a) \setminus b} w_{aa'} \sum_{b' \in N(b) \setminus a} w_{bb'}}{Z} \frac{w_{ac}}{\sum_{a' \in N(a) \setminus b} w_{aa'}} \frac{w_{bc'}}{\sum_{b' \in N(b) \setminus a} w_{bb'}} \\
&= \frac{w_{ab} w_{bc'} w_{ca}}{Z}.
\end{aligned}$$

□

Let T denote the number of triangles sampled out of a total of s samples.

Lemma 2. $E[x_{abc}] = \frac{T}{Z} w_{ab} w_{bc} w_{ca}$.

Proof. Let X_{abcl} be a random variable defined as follows. In iteration l, if edges (a, b) and (a, c) are sampled, then X_{abcl} is equal to $I[c = c']$, where c' is the endpoint of the third sampled edge (b, c'). Otherwise, it is equal to 0. Note that $x_{abc} = \sum_l X_{abcl}$. Using linearity of expectation, we get that $E[x_{abc}] = \sum_{l=1}^{s} E[X_{abcl}] = \sum_{l=1}^{s} \sum_{c'} P(\delta_{abcc'}) I[c = c'] = \frac{T}{Z} w_{ab} w_{bc} w_{ca}$. □

Lemma 3. *Let $\epsilon \in (0, 1)$ be the error and $\delta \in (0, 1)$ be the error probability. If the number of sampled triangles*

$$T \geq 3Z \frac{\log(2/\delta)}{\epsilon^2 w_{ab} w_{bc} w_{ca}},$$

then

$$P(|x_{abc} - \frac{T}{Z} w_{ab} w_{bc} w_{ca}| > \epsilon \frac{T}{Z} w_{ab} w_{bc} w_{ca}) \leq \delta.$$

Proof. Note that $x_{abc} = \sum_{l=1}^{s} X_{abcl}$ and X_{abcl} is equal to 1 if the sampled edges have $c = c'$, 0 otherwise. These indicator variables are also independent across the s iterations. Thus, using the Chernoff bound, we get $P(x_{abc} - \frac{T}{Z} w_{ab} w_{bc} w_{ca} > \epsilon \frac{T}{Z} w_{ab} w_{bc} w_{ca}) \leq \exp(-\epsilon^2 \frac{T}{Z} w_{ab} w_{bc} w_{ca}/3)$. Given our choice of T, the right hand side is at most $\delta/2$. Using the Chernoff bound again for the other tail and using the union bound, we obtain the desired result. □

Lemma 4. *Let $\tau > 0$ be some threshold and $\delta \in (0, 1)$ be the error probability. If the number of sampled triangles*

$$T \geq 12Z \frac{\log(n^3/\delta)}{\tau},$$

then with probability at least $1-\delta$, the following holds for all triangles $\{a, b, c\}$ and $\{a', b', c'\}$: if $w_{ab} w_{bc} w_{ca} > \tau$ and $w_{a'b'} w_{b'c'} w_{c'a'} < \frac{\tau}{15}$, then $x_{abc} > x_{a'b'c'}$.

Proof. Consider a triangle $\{a, b, c\}$ with $w_{ab} w_{bc} w_{ca} > \tau$. Applying lemma 3 with error $\epsilon = \frac{1}{2}$ and error probability $\frac{\delta}{n^3}$, we get that $x_{abc} > \frac{1}{2} \frac{T}{Z} w_{ab} w_{bc} w_{ca} \geq \frac{T}{Z} \frac{\tau}{2}$. Now consider a triangle $\{a', b', c'\}$ with $w_{a'b'} w_{b'c'} w_{c'a'} < \frac{\tau}{15}$. Applying the lower tail bound of Theorem 1.1 part 3 from [2], we get $\forall t > 2eE[x_{a'b'c'}]$, $P(x_{a'b'c'} > t) < 2^{-t}$. Applying lemma 2 and using the assumption that the weight of this triangle is less than $\frac{\tau}{3}$, we get $E[x_{a'b'c'}] = \frac{T}{Z} w_{a'b'} w_{b'c'} w_{c'a'} \leq \frac{T\tau}{16Z}$. Since we require $2eE[x_{a'b'c'}] < t$, or equivalently, $2e \frac{T\tau}{16Z} < t$, we can set $t = \frac{T\tau}{2Z}$. Using our assumption on the number of sampled triangles T, we get that $t \geq 12Z \frac{\log(n^3/\delta)}{\tau} \frac{\tau}{2Z}$.

Simplifying this expression yields $t \geq 6 \log \frac{n^3}{\delta}$. Thus, $P(x_{a'b'c'} > \frac{T}{Z}\frac{\tau}{2}) < \frac{\delta}{n^3}$. Taking a union bound over at most n^3 triangles we get the following result. With probability at least $1-\delta$ if triangle $\{a,b,c\}$ has $w_{ab}w_{bc}w_{ca} > \tau$, then $x_{abc} \geq \frac{T}{Z}\frac{\tau}{2}$, and if triangle $\{a,b',c'\}$ has $w_{a'b'}w_{b'c'}w_{c'a'} < \frac{\tau}{15}$, then $x_{a'b'c'} \leq \frac{T}{Z}\frac{\tau}{2}$. □

2.4 Sampling in $O(\log(\textbf{degree}))$ - Rejection Sampling

In algorithm 1 once we sample the edge (a,b), the normalization constant for the probability of sampling c from $N(a) \setminus b$, $Z(a,b)$, depends on b. This precludes us from sampling c in $O(\log(d_a))$ time, where d_a refers to the degree of vertex a, unless we use $O(d_a^2)$ space to store d_a probability vectors of length $O(d_a)$ each, one for each possible choice of b for vertex a. Since this might be prohibitive, we would have to settle for sampling $c \in N(a) \setminus b$ in $O(d_a)$ time instead.

A possible solution to this problem is to simply sample $c \in N(a)$ and use rejection sampling, i.e. discard $c = b$ and keep sampling till we sample $c \neq b$. In order for this solution to be efficient, we would like guarantees on how many trials we would need to ensure we encounter a vertex $c \neq b$ from $N(a)$. Let $Z' = \sum_{a' \in N(a)} w_{aa'}$ and $p = 1 - \frac{w_{ab}}{Z'}$. Observe that the number of trials needed to choose $c \neq b$ follows a geometric distribution, Geom(p), with success probability p as defined above referring to the probability of choosing a neighbor of a other than b. Thus, $P(\text{number of trials needed} = x) = (1-p)^{x-1}p = (\frac{w_{ab}}{Z'})^{x-1}(1 - \frac{w_{ab}}{Z'})$. As long as w_{ab} does not account for a huge fraction of the weights on the edges incident to b a moderate number of trials should suffice to sample $c \neq b$. The lower the probability of sampling b the lower the probability of needing x trials. If $\frac{w_{ab}}{Z'} = 0.9$, then $P(x=5) = 0.0651$. If $\frac{w_{ab}}{Z'} = 0.1$, then $P(x=5) = 9e-05$.

Empirically, we observe that the algorithm 1 with rejection sampling is generally an order of magnitude faster than without rejection sampling. The quality of the results is either comparable or even better with rejection sampling, especially for larger graphs.

2.5 Sampling in $O(\log(\textbf{degree}))$ - Alternative

We can also sample a neighbor of a from $N(a) \setminus b$ in $O(d_a)$ time without using rejection sampling by using the following algorithm. For each vertex a, create array of the cumulative sums of the weights of the neighbors of a. Since the weights are assumed to be non-negative this array will be sorted in increasing order. For ease of presentation, denote the weights as $w_1, w_2, \ldots, w_d$ and suppose we would like to sample a neighbor from $[1,d]$ excluding neighbor i. Thus, the array of cumulative sums have have the values $c_1 = w_1, c_2 = w_1 + w_2, \ldots, c_d = w_1 + w_2 + \cdots + w_d$. First, sample a uniform random number between 0 and $Z(a,b)$, where $Z(a,b) = \sum_{a' \in N(a) \setminus b} w_{aa'}$. If u does not land in interval i, return the interval it landed in. Otherwise, do the following. Flip a coin with probability of heads equal to c_{i-1}. If heads, generate a random number between 1 and $i-1$. Otherwise, generate a random number between $i+1$ and d.

Paul has a different algorithm for doing the same. In order to sample $c \in N(a) \setminus b$, first sample a uniform random number u between 0 and $Z(a,b)$. Thinking about the weights of the edges $w_{aa'}$ as intervals, check which interval u lies in. If it lies in the interval corresponding to w_{ab} implicitly divide it into $d_a - 1$ subintervals, each corresponding to each of the other neighbors of a and return the neighbor corresponding to the subinterval that u lies in. If this interval is the largest one, precompute a data structure that can return the answer efficiently. If not, then just do rejection sampling. In expectation, this will take at most 2 trials since the interval we want to exclude will be at most half of the total weight. Otherwise, return the interval u lies in.

3 Arithmetic and Harmonic Mean of Edge Weights

3.1 Problem

Given a weighted graph $G = (V, E)$ and an integer k, output a list of the top-k triangles ordered by the arithmetic or harmonic mean of their edge weights. For now, we will focus on simple undirected graphs with positive edge weights. For simplicity, for a triangle defined by vertices a, b, c we will consider the sum of the edge weights or the sum of the inverse of the edge weights instead.

3.2 Algorithm

3.2.1 Approach 1

One approach could be to reduce this problem to the one for geometric mean. If we consider the exponential of the edge weights then we would sample a triangle with probability proportional to $\exp(w_{ab})\exp(w_{bc})\exp(w_{ca}) = \exp(w_{ab} + w_{bc} + w_{ca})$. However, this will lead to overflow since some of the weights in real world networks are quite large. One possible solution is to divide the weights by the maximum edge weight so that all the weights are at most e. One possible issue is that this would place too "squeeze" the weights into a narrow region and bias the sampling probabilities in unintended ways. Instead, we consider the following algorithm.

3.2.2 Approach 2

The algorithm is the same as the one in the section on ordering triangles by geometric mean of edge weights with the difference being the sampling probabilities for sampling an edge (a, b) and neighbors c and c'. The following probabilities will sample triangles with probability proportional to the sum of edge weights, i.e. for the arithmetic mean case. For harmonic mean, replace the edge weights with the inverse of the edge weights.

1. Sample (a, b) with probability $\propto \left((d_a - 1) \cdot (d_b - 1) \cdot w_{ab} + (d_b - 1) \sum_{a' \in N(a) \setminus b} w_{aa'} + (d_a - 1) \sum_{b' \in N(b) \setminus a} w_{bb'}\right)$.
2. Sample $c \in N(a) \setminus b$ with probability $\propto \left((d_b - 1) \cdot (w_{ac} + w_{ab}) + \sum_{b' \in N(b) \setminus a} w_{bb'}\right)$.
3. Sample a neighbor $c' \in N(b) \setminus a$ with probability $\propto (w_{ac} + w_{ab} + w_{bc'})$.

Normalize the probabilities over all edges, all vertices in $N(a) \setminus b$ and all vertices in $N(b) \setminus a$ respectively.

3.2.3 Approach 3

Another simpler algorithm is the following. In each iteration, sample an edge with probability proportional to its weight. For each sampled edge, iterate over all neighbors of the endpoints of the edge and count the number of triangles that contain this edge. Then, a triangle defined by vertices a, b and c is counted with probability $w_{ab} + w_{bc} + w_{ca}$. The time complexity to count all triangles for a given edge is $d_u + d_v$, but this only needs to be done once at the end of the sampling loop.

3.3 Analysis

The analysis is the same as for the geometric mean with the product of edge weights replaced by the sum of edge weights or the sum of the inverse edge weights for arithmetic and harmonic mean respectively.

4 Number of Simplices (Closed Triangles)

4.1 Problem

Given a set of p simplices $\{S_i : i \in [p]\}$ and an integer k, output a list of the top-k closed triangles ordered by their weight, where weight of a triangle is defined as how many simplices the triangle appears in. A triangle $\{a, b, c\}$ is closed if all three vertices appear together in some simplex. For example, if $S = \{\{a, b, c, d\}, \{a, b, c, e\}, \{a, d\}\}$ then the weight of triangle $\{a, b, c\}$ is 2 since it appears in 2 simplices, S_1 and S_2.

4.2 Algorithm

Our approach is similar to diamond sampling for the maximum all-pairs dot-product (MAD) search problem [1]. At a high level, the algorithm works as follows. Given the list of simplices S and the projected graph G_S, we create a tripartite graph with a set of nodes A for edges in G_S, a set of nodes B for nodes in the input, and a set of nodes C for simplices. Let $a \in A, b \in B$ and $c \in C$. Since A represents edges in the projected graph, let $a = (u, v) \in E(G_S)$. There is an edge between a and c iff edge $a = (u, v)$ appears in simplex c and there is an edge between b and c iff node b appears in simplex c. We will set the sampling probabilities appropriately, and sample a "diamond" from this tripartie graph similar to [1]. If the "diamond" corresponds to a closed triangle, we increment a counter associated with this closed triangle. Intuitively, we expect that closed triangles which appear in more simplices will have a higher counter value, and we can output the closed triangles associated with the top-k counters.

Let $a = (u, v) \in A, b \in B$ and $c \in C$. The weight of the edge between a and c, w_{ac}, is equal to 1 if a appears in simplex c, 0 otherwise. The weight of the edge between b and c, w_{bc}, is equal to 1 if b appears in simplex c. Normalize the weights to get sampling probabilities $p_{ac} = w_{ac}/W_1$ and $p_{bc} = w_{bc}/W_2$, where W_1 and W_2 are the normalizing constants.

Then, we use algorithm 2 to postprocess and output the result.

4.3 Analysis

The analysis is almost identical to [1] with a few of minor modifications. x_{ab} is a counter associated with the closed triangle defined by edge a and third vertex b, and $\sum_c w_{ac} w_{bc}$ is equal to the number of simplices that the closed triangle defined by edge a and third vertex b appears in. Additionally, the dependence is on the number of closed triangles sampled, T, and not the number of samples S since some of the samples might sample a vertex b which is an endpoint of the sampled edge a and this is not a triangle.

The number of nodes in this tripartite graph is $O(m + n + |S|)$. Each of the two sets of edges will be sparse. Each simplex contains a small number of vertices and thus, the set of edges between B and C will be sparse. The distribution of edge weights in the projected graph is skewed towards 1, with only a few edges having very high weight. In our construction, the number of outgoing edges from $a \in A$ is equal to the weight of

Algorithm 3: Sampling Closed Triangles

Input : A set of simplices S, number of samples s.
Output: Counters x initialized to 0.

Let $A = \{a = (u, v) : \exists S_i \in S, u, v \in S_i\}$.
Let B be the set of vertices in the input.
Let C be the set of simplices in the input.
for $l = 1 \dots s$ **do**
    Sample edge (a, c) with probability p_{ac}.
    Sample edge (b, c) with probability p_{bc}.
    if $a = (u, v)$ *and* b *is equal to* u *or* v **then**
        continue
    end
    Sample edge (a, c') with probability $p_{ac'}$.
    $x_{ab} \leftarrow x_{ab} + w_{bc'}$.
end
return x

the edge corresponding to node a in the projected graph, and thus, the set of edges between A and C will be sparse.

5 Number of Simplices (All Triangles)

5.1 Problem

Given a set of p simplices $\{S_i : i \in [p]\}$ and an integer k, output a list of the top-k triangles ordered by their weight, where weight of a triangle is defined as the number of simplices which contain at least one edge of the triangle. For example, if $S = \{\{a, b, x\}, \{b, c, y\}, \{a, c, z\}, \{a, u, v\}\}$ then the weight of triangle $\{a, b, c\}$ is 3 since S_1 contains edge $\{a, b\}$, S_2 contains $\{b, c\}$ and S_3 contains $\{a, c\}$.

5.2 Algorithm

Our approach is similar to diamond sampling for the maximum all-pairs dot-product (MAD) search problem [1] and algorithm 3 for ordering closed triangles by the number of simplices they appear in as stated in section 4. At a high level, the algorithm works as follows. Given the list of simplices S and the projected graph G_S, we create a 4-partite graph with a three sets of nodes A, B and D for edges in G_S, and a set of nodes C for simplices. Let $a, b, c, d \in A, B, C, D$. There is an edge from $x = a, b, d$ to c iff at least one endpoint of the edge represented by $x = a, b, d$ is in simplex c. We will set the sampling probabilities appropriately, and sample a "diamond" from this 4-partite graph. If the "diamond" corresponds to a triangle, we increment a counter associated with this triangle. Intuitively, we expect that triangles which appear in more simplices will have a higher counter value, and we can output the closed triangles associated with the top-k triangles.

Similar to the previous section, we will set the weight of each edge to be 1 and normalize them to get sampling probabilities.

Algorithm 4: Sampling All Triangles

Input : A set of simplices S, number of samples s.
Output: Counters x initialized to 0.

Let $A = \{a = (u, v) : \exists S_i \in S, u, v \in S_i\}$.
Let $B = D = A$. Let C be the set of simplices in the input.
for $l = 1 \dots s$ **do**
  Sample edge (a, c) with probability p_{ac}.
  Sample edge (b, c) with probability p_{bc}.
  Sample edge (d, c) with probability p_{dc}.
  if *edges a, b, d do not form a valid triangle* **then**
    continue
  end
  Sample edge (a, c') with probability $p_{ac'}$.
  $x_{abd} \leftarrow x_{abd} + w_{bc'} w_{dc'}$.
end
return x

Then, we use algorithm 2 to postprocess and output the result.

5.3 Analysis

The analysis is similar to the previous section with the following modification: $\sum_c w_{ac} w_{bc} w_{dc}$ is equal to the number of simplices which contain at least 1 edge of the triangle defined by the edges a, b and d. One concern is that we might need quite a few trials to sample a valid triangle in each iteration. We will observe the empirical performance of the algorithm and see how well it performs.

6 Common 4th Neighbor

6.1 Problem

Given a list of triangles and an integer k, output a list of the top-k triangles ordered by the number of common 4^{th} neighbors, or equivalently, the number of 4-cliques the triangle is part of.

6.2 Algorithm and Analysis

A simple deterministic algorithm for this problem is to store the neighbors of every vertex in the graph as a sparse vector, and for each triangle compute the sparse dot product between the 3 sparse vectors representing the neighbors of the 3 vertices of the triangle. This will output the number of common 4^{th} neighbors for each triangle and we can output the triangles corresponding to the k largest of these. This will take $O(T * d_{\max} + T * \log k)$ time where $d_{\max}$ is the maximum degree of the vertices of the graph.

However, we would like a sampling based algorithm whose time complexity is not linear in the degree. A simple sampling based algorithm is to iterate through the triangles and for each triangle, randomly sample some neighbors of the vertices of the triangle and see how many of them are common 4^{th} neighbors. Fix some vertex of the triangle and let d denote its degree. (We may want to pick the vertex of the triangle with the lowest degree.) Suppose we pick s samples from these d vertices uniformly at random. How many of these would be common 4^{th} neighbors? Let X_i be an indicator random variable that is equal to 1 if sample i is a common 4^{th} neighbor, 0 otherwise. Let the estimator be $\tilde{C} = \frac{d}{s}\sum_{i=1}^{s} X_i$. Let C be the true number of common 4^{th} neighbors for the chosen triangle.

Lemma 5. *$E[\tilde{C}] = C$.*

Proof. By linearity of expectation, we have that $E[\tilde{C}] = \frac{d}{s} \times s \times E[X_1] = dP(X_1 = 1)$. Since we sample a vertex uniformly at random from d vertices in the neighbor list out of which C are common 4^{th} neighbors, $P(X_1 = 1) = \frac{C}{d}$. Thus, $E[\tilde{C}] = C$ and $\tilde{C}$ is an unbiased estimator. □

Lemma 6. *Let $\epsilon \in (0, 1)$ be the accuracy parameter and let $\delta \in (0, 1)$ be the confidence parameters. If $s \geq \frac{\ln(2/\delta)}{(2\epsilon)^2}$, then $P(|\tilde{C} - C| \geq \epsilon) \leq \delta$.*

Proof. $\tilde{C} = d \times \frac{1}{s}\sum_{i=1}^{2} X_i$, where X_i is an indicator random variable that is equal to 1 if sample i is a common 4^{th} neighbor, 0 otherwise. Thus, the result follows by Hoeffding's inequality. □

Note that we would like $s \ll d$, otherwise we might as well use the deterministic solution.

References

[1] Grey Ballard, Tamara G. Kolda, Ali Pinar, and C. Seshadhri. Diamond sampling for approximate maximum all-pairs dot-product (mad) search. *International Conference on Data Mining (ICDM)*, 2015.

[2] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms.* Cambridge University Press, 2012.