COUNTING STREAMED SUB-HYPERGRAPHS THROUGH NEIGHBORHOOD SAMPLING

BENJAMIN CHEN AND PATRICK GUO

ABSTRACT. We generalize Pavan *et al.*'s idea of neighborhood sampling for counting cliques [20] to design a one-pass cash-register streaming algorithm for approximating the number of occurrences of any arbitrary sub(hyper)graph H in massive graph G. Compared to (what we believe to be) the only other such general algorithm by Sun [21], neighborhood sampling produces a space bound that is situationally better, particularly on massive graphs of low maximal degree, low counts of subgraph H, and H with more edges than vertices.

1. INTRODUCTION

As large networks and graphs become more common in computing, the problem of counting the number of times a small subgraph appears in a massive graph stream has become an active area of research both for theorists and systems researchers. Important applications, as detailed in [17, 22], include motif detection in protein interaction networks [18], finding hidden structure in the web [7], analyzing social networks [23], and evaluating large graph models [15].

The most basic version of this problem, estimating the number of triangles in a streamed graph, has seen numerous creative algorithms invented in the past decade, but the natural generalization to counting other motifs has received relatively less attention. Our project addresses the question of counting occurrences of arbitrary sub(hyper)graph H in massive graph G.

1.1. Preliminaries.

Throughout we take $G = (V, E)$ to be the massive graph stream of interest with $n = |V|, m = |E|$, and maximal degree Δ. We assume G is streamed with the adjacency stream model, i.e. the cash register streaming of edges $f_1, \cdots, f_m$ where the

Date: December 2018.

order of edges is arbitrary and potentially adversarial. Let $\#(H, G)$ denote the number of occurrences of subgraph H in G, and let $T := K_3$ be the triangle graph.

1.2. **Previous Related Work.** Triangle counting in the streaming setting was first studied by Bar-Yossef et al. in [3]. They provided the first (δ, ϵ)-approximation algorithm which required $O\left(\frac{1}{\epsilon^3}\log\frac{1}{\delta}\frac{m^3n^3}{\#(T,G)^3}\log n\right)$ space (we suppress the $\epsilon, \delta, \log n$ terms for the rest of this section). More importantly, they showed a lower bound of $\Omega(n^2)$ space required to approximate $\#(T, G)$ in worst case graphs, thus showing that triangle approximating algorithms could not be sublinear in full generality under the one-pass cash register streaming model (this was later generalized to $\Omega(m)$ in [5], e.g. given n, m, one can construct a graph where triangle approximation requires $\Omega(m)$ space).

Nevertheless, these lower bounds were only shown in edge cases where all triangles shared a common edge. The majority of graphs behave differently, and Bar-Yossef et al's upper bound has seen significant improvement. Jowhari and Ghodsi provided a sampling algorithm using $O\left(\frac{m\Delta^2}{\#(T,G)}\right)$ space in the cash-register model as well as a linear estimator with space $O\left(\frac{m^3}{\#(T,G)^2}\right)$ that applied to streams in the turnstile model (i.e. edge deletions supported as well) [14]. Buriol et al.[6] refine Jowhari and Ghodsi's sampling method to attain a bound of $O\left(\frac{mn}{\#(T,G)}\right)$, and Pavan et al. [20] introduce *neighborhood sampling*, where edges are sampled from those that are adjacent to already-sampled edges, to approximate triangles in $O\left(\frac{m\Delta}{\#(T,G)}\right)$ space. Pagh and Tsourakakis [19] then present a static triangle approximating algorithm which can be adapted to a streaming algorithm using $O\left(\frac{m}{\sqrt{\#(T,G)}} + \frac{m\Delta_E}{\#(T,G)}\right)$ space, where Δ_E is the maximum number of triangles sharing an edge. Most recently, Kallaugher and Price [10] develop an algorithm based on sampling vertices that achieves space $O\left(m\log\#(T,G)\left(\frac{1}{\#(T,G)^{2/3}} + \frac{\Delta_E}{\#(T,G)} + \frac{\sqrt{\Delta_V}}{\#(T,G)}\right)\right)$ where Δ_V is the maximum number of triangles sharing a vertex.

Moving beyond triangles, one-pass algorithms have also been given for cycles [1] and stars [9], and Pavan et al. [20] extend their algorithm to any fixed-size clique. The first general subgraph counting algorithm was given by Kane et al. [12] who generalized the estimator of Jowhari and Ghodsi. Sun [21] further generalized this estimator to work

for hypergraphs as well. Kallaugher and Price [10] show that their triangle counting algorithm extends to bounded-degree subgraphs and hypergraphs [11], which is the state of the art for one-pass algorithms so far. Algorithms allowing for multiple passes have been considered as well, which achieve better space bounds [2], [4].

1.3. **Our Contribution.** We use Pavan *et al.*'s idea of *neighborhood sampling* to design what we believe to be the second (after [21]) fully generalized, one-pass algorithm for counting streamed sub-hypergraphs. The space bound is situationally better than that of [21] and [12], particularly on graphs G of low maximal degree, low counts of $\#(H, G)$, and H with more edges than vertices. We analyze the performance on the special case of counting tetrahedra.

2. Neighborhood Sampling

We summarize the main technique of [20] to count triangles. A naïve sampling method would be to use reservoir sampling to uniformly at random sample a set of 3 edges from the stream, and check if they form a triangle at the end. The idea of neighborhood sampling is to sample the first edge uniformly at random from all edges, but then subsequent edges are considered from the substream consisting of the edges adjacent to previous edges (i.e. their "neighborhood," hence the name). This produces an estimator with lower variance since we are only interested in cases where the edges eventually form a triangle. However, triangles are no longer sampled uniformly, as can be seen in Figure 1, since the probability of each triangle being found at the end of the stream is proportional to the the size of its neighborhood, so the estimator needs to be scaled by the product of sizes of substreams considered.

3. Counting General Subgraphs

The basic idea for H other than cliques is for reservoir sampling of subsequent edges to occur only on the substream of edges that could possibly form H with the previous

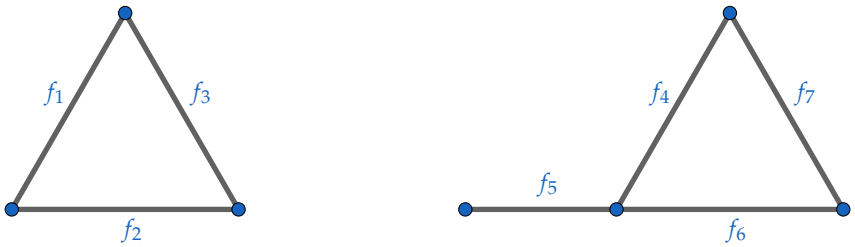

FIGURE 1. Triangle $\{f_1, f_2, f_3\}$ sampled with probablity $\frac{1}{14}$: there is a $\frac{1}{7}$ chance of sampling f_1 as the first edge in the triangle and a $\frac{1}{2}$ chance of sampling f_2 is from the substream of the 2 neighbors of f_1. However, triangle $\{f_4, f_6, f_7\}$ is sampled with probability $\frac{1}{21}$, with a $\frac{1}{7}$ chance of sampling e_4 but only a $\frac{1}{3}$ chance of sampling f_6 as the second edge since f_4 has 3 neighbors.

edges. We will only consider connected subgraphs H, since if H is not connected, we can reduce the problem to counting copies of each of the connected components of H.

3.1. **Edge Automorphisms and Permutations.** In particular, for a subgraph H with a vertices and b edges, there are $b!$ possible orderings for the edges in the stream, so a natural idea is to have $b!$ classes of samplers for copies of the subgraph depending on the edge ordering in the stream. However, the issue with this is that symmetries based on the structure of the subgraph H can cause a single copy of H in G to be consistent with multiple orderings, resulting in overcounting.

To deal with this overcounting, we consider $\text{Aut}^*(H)$, the edge automorphism group of H, which is the group of bijective maps from the set of edges of H to itself so that the images of two edges share a vertex if and only if the two original edges shared a vertex.

Let $\gamma_H := |\text{Aut}^*(H)|$. We maintain $\frac{b!}{\gamma_H}$ types of estimators, each corresponding to a subset of permutations of the edges of H.[1] The way we define the classes corresponds each class to a set of γ_H permutations of the edges, so that all $b!$ possible permutations are partitioned into exactly one class.

[1]We know that $\gamma_H | b!$ because $\text{Aut}^*(H)$ is a subgroup of the permutation group of the edges of H, which has order $b!$, and we can then apply Lagrange's Theorem.

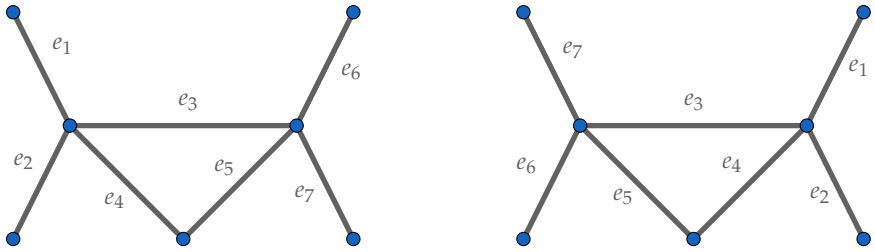

FIGURE 2. This graph has 8 edge automorphisms (generated by swapping e_1 with e_2, (e_1, e_2, e_4) with (e_6, e_7, e_5), and e_6 with e_7). The 2 edge orderings above are isomorphic and thus belong to the same class, so there will not be separate estimators for orderings $e_1, e_2, e_3, e_4, e_5, e_6, e_7$ and $e_7, e_6, e_3, e_5, e_4, e_1, e_2$.

Two permutations are in the same class if they induce an edge automorphism of H. To be more precise, let $e_1, e_2, \ldots, e_b$ and $e_{\sigma(1)}, e_{\sigma(2)}, \ldots, e_{\sigma(b)}$ be two permutations of the b edges of H. Then, they are in the same class if and only if the map that sends e_i to $e_{\sigma(i)}$ is an edge automorphism of H. For an example, see Figure 2

Lemma 1. *The relation defined above is an equivalence relation.*

Proof. This follows directly from the group structure of $\mathrm{Aut}^*(H)$. This relation is symmetric because the inverse of an edge automorphism is also an edge automorphism. It is reflexive because the identity automorphism is an edge automorphism. Finally, it is transitive because the composition of two edge automorphisms is also an edge automorphism. □

Thus, it is clear that each of these classes corresponds to a coset of $\mathrm{Aut}^*(H)$ as a subgroup of the full permutation group of the edges of H and has size γ_H.

3.2. **Generalized Subgraph Neighborhood Sampling.** Now, we can generalize the neighborhood sampling method in [20].[2] Consider the equivalence class of some permutation

[2]We believe there is a minor error in [20] in the explanation of how to generalize their triangle-counting method to K_4. In particular, the algorithm they omit to save space in the analysis of type-II K_4 subgraphs cannot exist because if r_1 and r_2 are both sampled uniformly at random from the m edges, it cannot be

of the edges of H, say $e_1, e_2, \ldots, e_b$. We use reservoir sampling to maintain a random candidate r_1 for e_1 chosen uniformly from the edges of G. Then, there are two possibilities: either e_2 shares an edge with e_1, or e_2 does not share an edge with e_1. In the first case, we maintain a random candidate r_2 for e_2 chosen uniformly from the substream of edges that share an edge with r_1. In the second case, we maintain a random candidate r_2 for e_2 randomly chosen from the remaining edges which do not share an edge with r_1.

Note that, in the first case, we have now fixed 3 of the vertices in H, while in the second, we have fixed 4. In general, a new edge that does not share a vertex with previous edges determines 2 new vertices, a new edge that shares a vertex with a previous edge determines 1 new vertex, and an edge between two vertices that both are endpoints of edges already stored contributes no new vertices. Call the first type of edge an *unbound* edge, the second type a *bound* edge, and the third type a *redundant* edge. We always have at least one unbound edge because the first edge is unbound.

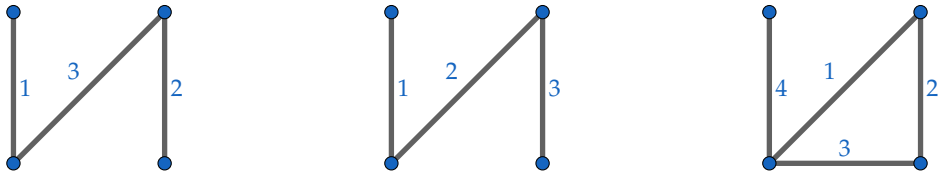

FIGURE 3. Unbound, Bound, and Redundant Edge Examples

For example, in Figure 3, in the left subgraph/ordering, edge 2 is unbound and 3 is redundant. In the middle subgraph/ordering, 2 is bound and 3 is also bound. In the right subgraph/ordering, 2 is bound, 3 is redundant, and 4 is bound. In any subgraph/ordering, 1 is always unbound.

We continue this process, storing in total b sample edges $r_1, r_2, \ldots, r_b$. We have that $2p + q = a$, where p is the number of unbound edges and q is the number of bound edges,

guaranteed that r_1 and r_2 share no vertex. We correct this error by adding a counter for each "unbound" edge reservoir as well; it is also possible to simply abandon the condition that r_1 and r_2 share no vertices altogether.

so that $b - p - q$ is the number of redundant edges. Note also that $p + q < 2p + q = a$ because $p \geq 1$.

Algorithm 1 Neighborhood Sampling for estimator with edge ordering $e_1, \ldots, e_b$ of H

```
1: r_1, ..., r_b ← ∅
2: C_1, ..., C_b ← 0
3: for Receive edge f_i do
4:     C_1 ← C_1 + 1
5:     if rand(1/C_1) then                ▷ rand(x) returns true with probability x
6:         r_1 ← f_i
7:         r_2, ..., r_b ← ∅
8:         C_2, ..., C_b ← 0
9:     else if f_i satisfies adjacency conditions for e_2 with r_1 then
10:        C_2 ← C_2 + 1
11:        if rand(1/C_2) then
12:            r_2 ← f_i
13:            r_3, ..., r_b ← ∅
14:            C_3, ..., C_b ← 0
15:        else if f_i satisfies adjacency conditions for e_3 with r_1, r_2 then
16:            ⋮
17: if r_b ≠ ∅ then
18:     return C_1 C_2 ... C_b
19: else
20:     return 0
```

The first unbound edge is sampled uniformly from among all m edges of G. The other unbound edges are sampled uniformly from the remaining edges of G that do not share an edge with an edge currently in the reservoir. For each unbound edge except the first, we maintain a counter P_j for the jth unbound edge. Each counter keeps track of how many edges do not share any vertices with current edges in the reservoir (starting from the last edge currently in the reservoir), so that we can correctly used this layered reservoir sampling on the substream of edges that do not share vertices with edges currently in the reservoir. The counters are reset every time we update an edge r_s where $s < j$.

The bound edges are sampled uniformly from the set of all edges that satisfy the appropriate incidence criteria with the preceding edges, which is done using a counter for each substream of such edges for each reservoir. In other words, in addition to the b edges we store, we also store a counter Q_j for each bound edge e_{i_j} that counts the number of

possible candidates for e_{i_j} that we have seen, where $i_1 < i_2 < \cdots < i_q$ are the indices of the bound edges. The counter starts at 0 when we take an edge as a candidate for e_{i_j-1} and is incremented by 1 every time we see an edge that satisfies the constraints for being a candidate for e_{i_j}. The counter is reset to 0 every time a new candidate for e_{i_j-1} is placed into the reservoir (in particular, if any of $r_1, r_2, \ldots, r_{i_j-1}$ are changed, the counter is reset since taking in a new value for r_i automatically resets r_s to $\varnothing$ for $s > i$).

The redundant edges are sampled uniformly in the same way as the unbound edges, with their respective counters R_i. However, since they are bound, note that there are at most 4 possible candidates for each redundant edge, that is, $R_i \leq 4$.

In Algorithm 1, C_i is the counter for e_i, that is, it's either P_i, Q_i, or R_i depending on the type of edge e_i is ($C_1 = P_1$ is simply a running tally of the number of edges so far). These counter quantities are important because the algorithm does not necessarily sample uniformly from among all copies of H in G; we will need them to normalize this bias in order to construct an unbiased estimator for $\#(H, G)$.

The conditional test to see if f_i satisfies the conditions for e_j can be easily determined from the structure of H. For unbound edges, simply check for adjacency on $r_1, r_2, \ldots, r_{j-1}$. For bound and redundant edges, check for adjacency with the relevant previous edges based on vertex. For bound edges, one vertex should not be on any of $r_1, r_2, \ldots, r_{j-1}$, while the other should be incident to the relevant previous edges. The redundant case is similar, except we have to check for both possible orientations.

3.3. **Sampling Bias Analysis.** Consider some copy of H in G whose edges $w_1, w_2, \ldots, w_b$ are streamed in an order consistent with $e_1, e_2, \ldots, e_b$. We want to determine the probability that the above algorithm samples this copy of H, that is, $r_i = w_i$ for each i. In order for this to happen, first we must have $r_1 = w_1$, which occurs with probability $\frac{1}{m}$. Every other unbound edge matches with probability $\frac{1}{P_i}$. Every bound edge matches with probability $\frac{1}{Q_i}$. Every redundant edge matches with probability $\frac{1}{R_i}$.

It follows that the probability that our algorithm samples this particular copy of H is

$$\frac{1}{m \prod_{2 \le i \le p} P_i \prod_{1 \le i \le q} Q_i \prod_{1 \le i \le b-p-q} R_i},$$

where Q_i is the counter for the ith bound edge.

If σ is an equivalence class of the permutations of the edges of H, then define τ_σ to be the number of copies of H in G whose edges arrive in the data stream in an order in σ. Then, define $\hat{\tau}_\sigma$ to be a random variable based on our algorithm output:

$$\hat{\tau}_\sigma = \begin{cases} m \prod_{2 \le i \le p} P_i \prod_{1 \le i \le q} Q_i \prod_{1 \le i \le b-p-q} R_i & \text{a copy of } H \text{ is found by the algorithm} \\ 0 & \text{otherwise} \end{cases}$$

It is easy to see that $\hat{\tau}_\sigma$ is an unbiased estimator of τ_σ, as

$$E[\hat{\tau}_\sigma] = \sum_{H \in \mathcal{H}_\sigma} \left(\left(m \prod_{2 \le i \le p} P_i \prod_{1 \le i \le q} Q_i \prod_{1 \le i \le b-p-q} R_i \right) P[\hat{H} = H] \right) =$$

$$\sum_{H \in \mathcal{H}_\sigma} 1 = |\mathcal{H}_\sigma| = \tau_\sigma,$$

where $\mathcal{H}_\sigma$ is the set of copies of H in G consistent with σ and $\hat{H}$ is the random variable of the copy of H sampled by our algorithm ($\hat{H} = \varnothing$ when no copy of H is sampled).

It follows that $\hat{\tau} := \sum_\sigma \hat{\tau}_\sigma$ is an unbiased estimator for $\#(H, G)$, the total number of copies of H in G.

3.4. **Concentration Bounds.** Now, note that $P_i \le m$, $Q_i \le 2\Delta$, and $R_i \le 4$. Using these bounds, we can bound the support of this estimator.

$$0 \le \hat{\tau}_\sigma \le m^p (2\Delta)^q 4^{b-p-q}$$

A uniform bound over all σ is then

$$0 \le \hat{\tau}_\sigma \le \max_{i=1}^{|M|} m^i (2\Delta)^{a-2i} 4^{b-a+i}$$

where $|M|$ is the cardinality of a maximal matching on H (e.g. an upper bound for p, the number of unbound edges). To establish concentration of this estimator, we use Chernoff bounds/Hoeffding's inequality:

Lemma 2. *[Chernoff Bounds] If $X_1, \cdots, X_n$ are independent random variables supported on $[0,1]$, then for $X = X_1 + \cdots + X_n$ and $\mu = E[X]$ we have for any $\epsilon > 0$ that*

$$P[X > (1+\epsilon)\mu] \leq e^{-\frac{\epsilon^2}{3}\mu}$$

$$P[X < (1-\epsilon)\mu] \leq e^{-\frac{\epsilon^2}{2}\mu}$$

Therefore, by keeping enough independent copies of the estimator and taking the average, we can get a good approximation with high probability. In particular,

Theorem 3. *Let H be any graph with a vertices, b edges, and a maximal matching of cardinality $|M|$. There exists a one-pass, cash-register $O(r * \frac{b!}{\gamma_H} \log n)$ space algorithm that (ϵ, δ)-approximates the number of occurrences of H in an arbitrary graph G with m edges and maximal degree Δ, provided that $r \geq \max_{i=1}^{|M|} \frac{3}{2\epsilon^2} \log \frac{2}{\delta} \frac{m^i (2\Delta)^{a-2i} 4^{b-a+i}}{\#(H,G)}$*

Proof. Let $Z = \max_{i=1}^{|M|} m^i (2\Delta)^{a-2i} 4^{b-a+i}$. For each edge-automorphism class σ, let $X_1^\sigma, \cdots, X_r^\sigma$ be r independent copies of $\frac{\hat{\tau}_\sigma}{Z}$, and let $X = \sum_\sigma \sum_{j=1}^r X_j^\sigma$ so that $E[X] = \frac{r\#(H,G)}{Z}$. Applying Lemma 2, we have that

$$P[|\hat{\tau} - \#(H,G)| > \epsilon\#(H,G)] = P\left[\left|X - \frac{\#(H,G)}{Z}\right| > \frac{\epsilon\#(H,G)}{Z}\right]$$

$$\leq e^{-\frac{2\epsilon^2}{3}\frac{r\#(H,G)}{Z}} \leq \delta.$$

□

For fixed size H, note that a,b, and γ_H are constant and each of the $r * \frac{b!}{\gamma_H}$ estimators requires $O(\log n)$ bits to store values up to $m^p (2\Delta)^q 4^{b-p-q}$ so the space upper bound is

$$O\left(\max_{i=1}^{|M|} \frac{1}{\epsilon^2} \log \frac{1}{\delta} \frac{m^i \Delta^{a-2i}}{\#(H,G)} \log n\right).$$

3.5. **Counting General Sub-hypergraphs.** This algorithm applies in the hypergraph case as well with the same space bound of $O\left(\max_{i=1}^{|M|} \frac{1}{\epsilon^2} \log \frac{1}{\delta} \frac{m^i \Delta^{a-2i}}{\#(H,G)} \log n\right)$.

4. Bound Comparisons

To get a sense for how well our algorithm performs, we can first compare it to the naïve sampling method of reservoir sampling b edges from the entire stream. The probability of sampling an occurrence of H is $\frac{\#(H,G)}{m^b}$, so a similar Chernoff bound gives space requirement (suppressing $\epsilon, \delta, \log n$ terms) for this method of $O\left(\frac{m^b}{\#(H,G)}\right)$. The bound given by the other general one-pass algorithm in [12], [21] is similarly dependent on b, at $O\left(\frac{m^b}{\#(H,G)^2}\right)$. The bound from neighborhood sampling of $O\left(\max_{i=1}^{|M|} \frac{m^i \Delta^{a-2i}}{\#(H,G)}\right)$ is then a significant improvement for H with $a < b$ (fewer vertices than edges), for G with bounded degree and H with small maximal matching, and for graphs where $\#(H, G)$ is not large.

4.1. **Performance on Tetrahedra.** Tetrahedra are the natural extension of triangles to 3-hypergraphs and work nicely with neighborhood sampling due to full edge symmetry and maximal matching of size 1. Theorem 3 then gives a one-pass algorithm to approximate these using $O\left(\frac{m\Delta^2}{\#(H,G)}\right)$ space, which is better than Sun's bound for the tetrahedra case of $O\left(\frac{m^4}{\#(H,G)^2}\right)$ so long as $d^2\#(H, G) = o(m^3)$. Note that $|\#(H, G)| = O(mn)$ (each of m edges part of at most n tetrahedra), and $d = O(n)$, so this is an improvement anytime $m = \omega(n^{3/2})$, as well as when $|\#(H, G)|, d$ are small. We also remark that this is sublinear when $d^2 = o(\#(H, G))$, whereas Sun's general algorithm never yields sublinear space.

5. Conclusion

We have generalized the neighborhood sampling method of Pavan *et al.* used for counting triangles and their suggested extension to cliques to the general case of arbitrary streamed sub(hyper)graphs in the cash-register model. We investigate the special case of tetrahedra counting, where our generalized algorithm gives sublinear space under certain conditions whereas Sun's never does.

There is a possibility the proof for the algorithm could be improved by incorporating the idea of arbitrarily assigning an orientation to H and sampling from a directed version of G (where a single undirected edge is read instead as two directed edges of opposite orientations), which is an idea used by Kane *et al*. [12] in the proof of their bound. This is one avenue of possible improvement and further research for generalizing neighborhood sampling.

Another extension involves non-arbitrary ordering of streamed edges. For example, McGregor *et al*. provide an algorithm for counting triangles in the adjacency list model of graph streaming where edges incident to the same node are streamed consecutively [17]; there may be space-efficient ways to apply neighborhood sampling in this model since neighborhoods can stop being tracked once the appropriate vertices stop having edges.

Finally, another possible direction for further investigation would be to consider applying our algorithm with the sliding window model. As Pavan states in [20], the method of neighborhood sampling is quite versatile and can likely be applied to a wide range of variations of this problem.

REFERENCES

[1] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. Algorithmica, 17(3):209–223, 1997

[2] Assadi, Sepehr, Michael Kapralov, and Sanjeev Khanna. "A Simple Sublinear-Time Algorithm for Counting Arbitrary Subgraphs via Edge Sampling." arXiv preprint arXiv:1811.07780 (2018).

[3] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proc. 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 623–632, 2002.

[4] S. K. Bera and A. Chakrabarti. Towards tighter space bounds for counting triangles and other substructures in graph streams. In 34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany, pages 11:1–11:14, 2017.

[5] Braverman, Vladimir, Rafail Ostrovsky, and Dan Vilenchik. "How hard is counting triangles in the streaming model?." International Colloquium on Automata, Languages, and Programming. Springer, Berlin, Heidelberg, 2013.

[6] Buriol, Luciana S., et al. "Counting triangles in data streams." Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. ACM, 2006.

[7] J.-P. Eckmann and E. Moses. Curvature of co-links uncovers hidden thematic layers in the world wide web. Proceedings of the National Academy of Sciences, 99(9):5825–5829, 2002.

[8] R. Etemadi, J. Lu, and Y. H. Tsin. Efficient Estimation of Triangles in Very Large Graphs. *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. Pages 1251-1260.

[9] M. Gonen, D. Ron, and Y. Shavitt. Counting stars and other small subgraphs in sublinear-time. SIAM J. Disc. Math., 25(3):1365–1411, 2011.

[10] Kallaugher, John, and Eric Price. "A hybrid sampling scheme for triangle counting." Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, 2017.

[11] Kallaugher, John, Michael Kapralov, and Eric Price. "The sketching complexity of graph and hypergraph counting." arXiv preprint arXiv:1808.04995 (2018).

[12] D. Kane *et al*. "Counting arbitrary subgraphs in data streams." International Colloquium on Automata, Languages, and Programming. Springer, Berlin, Heidelberg, 2012.

[13] M. Jha, C. Seshadhri, and A. Pinar. A space-efficient streaming algorithm for estimating transitivity and triangle counts using the birthday paradox. *ACM Trans. Knowl. Discov. Data*, 9(3):15:1–15:21, Feb. 2015.

[14] H. Jowhari, M. Ghodsi. (2005) New Streaming Algorithms for Counting Triangles in Graphs. In: Wang L. (eds) *Computing and Combinatorics*. COCOON 2005. Lecture Notes in Computer Science, vol 3595. Springer, Berlin, Heidelberg

[15] J. Leskovec, L. Backstrom, R. Kumar, and A. Tomkins. Microscopic evolution of social networks. In Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008, pages 462–470, 2008.

[16] A. McGregor. 2014. Graph stream algorithms: a survey. *ACM SIGMOD* 43, 1 (2014), 9–20.

[17] McGregor, Andrew, Sofya Vorotnikova, and Hoa T. Vu. "Better algorithms for counting triangles in data streams." Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems. ACM, 2016.

[18] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: simple building blocks of complex networks. In Science, pages 824–827, 2002.

[19] Pagh, Rasmus, and Charalampos E. Tsourakakis. "Colorful triangle counting and a mapreduce implementation." Information Processing Letters 112.7 (2012): 277-281.

[20] A. Pavan., K. Tangwongsan., S. Tirthapura., and K. Wu. 2013. Counting and sampling triangles from a graph stream. *Proc. VLDB Endow.* 6, 14 (September 2013), 1870-1881.

[21] H. Sun. Counting hypergraphs in data streams. CoRR, abs/1304.7456, 2013.

[22] C. E. Tsourakakis, M. N. Kolountzakis, and G. L. Miller. Triangle sparsifiers. J. Graph Algorithms Appl., 15(6):703–726, 2011.

[23] B. F. Welles, A. V. Devender, and N. S. Contractor. Is a friend a friend?: investigating the structure of friendship networks in virtual worlds. In Proceedings of the 28th International Conference on Human Factors in Computing Systems, CHI 2010, Extended Abstracts Volume, Atlanta, Georgia, USA, April 10-15, 2010, pages 4027–4032, 2010.

[24] R. Yuster. "Finding and counting cliques and independent sets in r-uniform hypergraphs." Information Processing Letters 99.4 (2006): 130-134.

[25] J. Zhang. (2010) A Survey on Streaming Algorithms for Massive Graphs. In: Aggarwal C., Wang H. (eds) Managing and Mining Graph Data. Advances in Database Systems, vol 40. Springer, Boston, MA