Pseudo-determinism for Graph Streaming Problems

Albert Gao, Fan Pu Zeng, Andrew Caosun, Puhua Cheng

ABSTRACT

Given a fixed input for a search problem, pseudo-deterministic algorithms produce the same answer over multiple independent runs, with high probability. For example, we can efficiently find a certificate for inequality of multivariate polynomials pseudo-deterministically, but it is not known how to do so deterministically. The same notion can be extended to the streaming model. The problem of finding a nonzero element from a turnstile stream is previously shown to require linear space for both deterministic and pseudo-deterministic algorithms. Another model of streaming problems is that of graphs, where edge insertions and deletions occur along a stream. Some natural problems include connectivity, bipartiteness, and colorability of a graph. While the randomized and deterministic graph streaming algorithms have been mostly well-studied, we investigate pseudo-deterministic space lower bounds and upper bounds for graph theoretic streaming problems.

School of Computer Science
Carnegie Mellon University
United States of America
November 22nd, 2022

1 Problem Statement

We begin with the definitions of pseudo-deterministic algorithms, the graph streaming model, and the turnstile streaming model.

Definition 1.1 (Pseudo-Determinism)**.** As in Goldwasser et al., 2019 [8], an algorithm A is *pseudo-deterministic* if for all inputs x, the algorithm A satisfies

$$\Pr_{r_1, r_2}[A(x, r_1) = A(x, r_2)] \geq 2/3,$$

where $A(x, r)$ denotes the output of A with input x and random bits r.

Though, this definition can be difficult to work with. Another way to think about pseudo-determinism can be seen in the following alternate definition:

Definition 1.2 (Alternate Definition for Pseudo-Determinism)**.** An algorithm A is *pseudo-deterministic* if there exists a function f such that for all inputs x, the algorithm A satisfies

$$\Pr_r[A(x, r) = f(x)] \geq 2/3.$$

That is, a pseudo-deterministic algorithm A simply computes some deterministic function f, except with probability $\frac{1}{3}$ it blows up and returns something else.

Note that given an algorithm A that satisfies the first definition of pseudo-determinism, we can create an algorithm that satisfies the second definition of pseudo-determinism by running A multiple times and taking the answer with a plurality. Next, we define the graph streaming model.

Definition 1.3 (Graph Streaming)**.** As in McGregor, 2014 [10], a graph stream is a sequence

$$S = \langle a_1, a_2, \cdots \rangle \text{ where } a_i = (e_i, \Delta_i),$$

where e_i denotes an undirected edge and $\Delta_i \in S$ for some $S \subseteq \mathbb{R}$. Let G be the graph defined by this stream; then the multiplicity f_e of an edge e in G is defined as $f_e := \sum_{i:e_i=e} \Delta_i$.

When $S = \{1\}$, we have an insert-only streaming model; when $S = \{-1, 1\}$, then we have an insert-delete streaming model. Finally, we define the turnstile streaming model.

Definition 1.4 (Turnstile Streaming)**.** In the turnstile streaming model, we have a stream that updates an underlying vector x initialized to 0^n. That is, we have a stream consisting of updates of the form $x_i \leftarrow x_i + \Delta_j$, where $\Delta_j \in \{-M, -M+1, \cdots, M-1, M\}$ for some $M \leq \text{poly}(n)$; and throughout the stream, each index of x is promised to be in $\{-M, -M+1, \cdots, M-1, M\}$. The goal is to estimate some function $f(x)$ using the final value of x.

In the following problems, we consider upper and lower bounds on pseudo-deterministic algorithm complexities under the insert-delete graph streaming model for various graph problems, such as finding spanning forests and other subgraphs, graph colorings, and counting triangles. We also consider pseudo-deterministic algorithms for estimating the l_2 norm under weak conditions in the turnstile streaming model.

1.1 Pseudo-deterministic Spanning Forests and Subgraphs

We investigate pseudo-determinism applied to spanning forests of a graph. In the insertion and deletion graph streaming model, it is known that there exists a randomized algorithm that outputs a spanning forest of the graph using $\widetilde{O}(n)$ space [1]. It is not difficult to show that any deterministic algorithm for the same task requires $\Omega(n^2)$ space. It is then natural to ask if adding pseudo-determinism can do better than deterministic algorithms. Borrowing a method from Goldwasser et al., 2019 [8], this turns out to be impossible via a reduction from other streaming problems that are hard for pseudo-deterministic algorithms. That is, any pseudo-deterministic algorithm also requires $\Omega(n^2)$ space. In a similar vein, we show that pseudo-deterministic algorithms cannot do better than deterministic algorithms in terms of space requirement for many other natural graph streaming problems.

In particular, consider any algorithm A in the graph streaming setting that when given an input graph G with at least one edge, returns a subgraph $A(G)$ of G with at least one edge. It can be shown (once again with Goldwasser et al., 2019 [8]) that if such an algorithm is pseudo-deterministic, then it must use at least $\Omega(n^2)$ space. This maps to a large variety of questions, including the question of spanning forests in the previous paragraph. Similar questions include "Return any edge in the graph," "Return the longest path," and "Return the largest connected component."

Another class of algorithms that we might consider is the class of algorithms that search for a subgraph of a particular shape, and return $\perp$ if one does not exist. That is, consider the following problem:

Definition 1.5 (IdenticalSubgraph)**.** For a fixed graph H, define IdenticalSubgraph to be the graph-streaming algorithm where the input is a graph G of size n, and the output is a subgraph of G that is isomorphic to H[1]; or is $\perp$ if no such subgraph of G exists.

More generally, we can define the following problem:

Definition 1.6 (ManyIdenticalSubgraph)**.** For a fixed set of subgraphs $\mathcal{H}$, define ManyIdentical-Subgraph to be the graph-streaming algorithm where the input is a graph G of size n, and the output is a subgraph of G that is isomorphic to some $H \in \mathcal{H}$; or is $\perp$ if no such subgraph of G exists.

One interesting property relating to this last problem is that of bipartiteness; when $\mathcal{H}$ is the set of all odd-length cycles, ManyIdenticalSubgraph corresponds exactly to outputting a certificate of non-bipartiteness under the graph streaming setting. And once again, it can be shown that these problems require $\Omega(n^2)$ space in the graph streaming setting.

1.2 Pseudo-deterministic Graph Colorings

Another interesting application of pseudo-deterministic algorithms is in the context of graph colorings. The number of k-colorings of a graph can be computed by the chromatic polynomial, which is polynomial in k, but still gives rise to a large number of possibilities.

Definition 1.7 (Graph Coloring)**.** A vertex graph coloring for a graph G is a function $C : V \to \mathbb{N}$ where $C(u) \neq C(v)$ for all $u \sim v \in G$.

We will only consider vertex graph colorings in this paper.

Definition 1.8 (k-coloring)**.** A coloring on G using at most k colors is called a k-coloring.

Definition 1.9 (Degree of a graph)**.** The degree Δ of a graph G is the maximum degree of any vertex in G.

Given such a graph G, we will show in Section 4.3 that under the more restrictive assumption that the algorithm is robust to non-existent edge deletions, any algorithm that finds a $(\Delta + 1)$ coloring of G in the semi-streaming context with high probability requires at least $\tilde{\Omega}(n^2)$ space.

1.3 The Usefulness of Pseudo-Determinism

Of course, one final question is whether or not *any* natural graph problem admits a pseudo-deterministic algorithm that uses asymptotically less space than its deterministic counterparts. For this question, it seems natural to look at algorithms that return quantities rather than vertices, edges, or subgraphs. For example, looking at the work of McGregor et al., 2016 [11], it seems plausible that pseudo-determinism could admit algorithms with lower space requirements. The same could be true of the counting versions of the IdenticalSubgraph and ManyIdenticalSubgraph problems.

2 Problem Motivation

We first discuss how pseudo-determinism can be useful in general, followed by in the specific context of graph streaming problems.

[1]This problem is similar to the graph isomorphism problem where the subgraph to search for is fixed.

There are many contexts in which a randomized algorithm that produces a unique answer can be useful. For instance, it is desirable for an algorithm for data restoration in a distributed setting to be pseudo-deterministic so that all parties recover the same output. Another example would be for generating system-wide parameters in the presence of distrust. For instance, the National Institute of Standards and Technology (NIST) may propose some value as a cryptographic parameter, but this may be distrusted by other parties as they may suspect it was chosen as there could be some backdoor for this choice of value. But if we could show that the parameter was generated by a pseudo-deterministic algorithm that others could replicate, then it is simply canonical in some sense and thus helps to remove the distrust.

Next, we consider how pseudo-deterministic algorithms are useful in a streaming context. It can help to reduce communication overhead in distributed algorithms who are all reading streams from the same data source, since with high probability the outputs that they computed are the same and they do not need to perform additional coordination to synchronize their values. It can also be useful for improving the stability of online learning algorithms in machine learning, since results are replicable. Furthermore, many large-scale problems are graphical in nature, like network graphs, protein folding, and transport optimization, which makes understanding pseudo-deterministic streaming algorithms for graph problems helpful as building blocks towards more specific solutions for each of the tasks.

3 Related Work

Originally the study of pseudo-determinism examined search problems that have fast randomized algorithms but no known deterministic algorithms such as finding a certificate of inequality of multivariate polynomials [7]. The application of pseudo-deterministic algorithms to streaming problems is explored in [8]. Finding a support element, finding a duplicate element, and finding l_2 approximation using a sketch in a stream are all shown to have no efficient pseudo-deterministic algorithms, whereas the problem of finding a nonzero row in a matrix is shown to have pseudo-deterministic complexity strictly between deterministic and randomized complexity. In the same paper, the authors leave open the question of whether there exists a sketching pseudo-deterministic algorithm for l_2 norm-estimation that uses $\text{poly}(\log n)$ space. Currently, there is only a $\tilde{O}(n)$ bound for deterministic complexity and $\tilde{\Omega}(\log n)$ bound for randomized complexity.

The study of graph streaming algorithms has a rich history. A sample of existing deterministic and randomized techniques for graph problems under the insertion-only model and the insertion-and-deletion model is listed in the survey [10]. In particular, there exists a deterministic algorithm for SpanningTree that achieves space linear in the number of edges [12], and there exists a randomized algorithm for SpanningTree that achieves space complexity $\widetilde{O}(n)$ [1]. As for IdenticalSubgraph and ManyIdenticalSubgraph, there has been extensive work done on the related subgraph isomorphism problem, but this problem has never been studied under the pseudo-deterministic nor graph streaming contexts. [2] has recently shown that it is possible to color graphs using $\tilde{O}(n)$ space with $O(\Delta^2)$ colors using two passes, and with $O(\Delta)$ colors in $O(\log \Delta)$ passes in the streaming context. They also showed the negative result that there is no deterministic single-pass semi-streaming algorithm for coloring graphs using at most $\exp(\Delta^{o(1)})$ colors, even if Δ is known beforehand. [3] also recently showed that it is possible to find with high probability a $(\Delta+1)$ coloring of a graph G with vertices of degree at most Δ in the semi-streaming context.

Finally, much prior work has been done on triangle counting in the graph streaming setting. Though most recent papers have focused on the insert-only graph streaming setting, [9] provides a seemingly optimal $\tilde{O}(\epsilon^{-2} m(\frac{1}{T^{2/3}} + \frac{\sqrt{\Delta_V}}{T} + \frac{\Delta_E}{T}))$ randomized algorithm for approximating the number of triangles up to a factor of $1 \pm \epsilon$ in the insert-delete graph streaming model. Here, in the final graph, m represents the number of edges, T represents the number of triangles, Δ_V represents the maximum number of triangles that share a vertex, and Δ_E represents the maximum number of edges that share an edge.

4 Lower Bounds

In this section, we present lower bounds for problems for pseudo-deterministic graph streaming.

4.1 Edge requires $\Omega(n^2)$ space

Proposition 4.1. *Let* Edge *be the graph streaming problem that returns an arbitrary edge from graph G (and $\perp$ if G has no edges). Any pseudo-deterministic algorithm requires $\Omega(n^2)$ space.*

Proof. We define a hard one-way communication problem OneWayEdges and reduce to Edge.

In an instance of OneWayEdges, we have two parties Alice and Bob, where Alice initially has a graph G on n vertices containing edge set E of size $\frac{3}{4}\binom{n}{2}$, and Bob has nothing. Alice sends a single message to Bob, and Bob desires to return a set $S \subseteq E$ of size $\frac{1}{10}\binom{n}{2}$.

To show our proposition, we proceed in two steps. First, we show that any pseudo-deterministic streaming algorithm for Edge with space s and failure probability $O(\frac{1}{n^3})$ induces a pseudo-deterministic algorithm for the communication problem OneWayEdges with the communication s and failure probability $O(\frac{1}{n})$ (which we denote OneWayEdges $\leq$ Edge). Then, we show that any pseudo-deterministic algorithm for OneWayEdges achieving $\frac{2}{3}$ success probability requires $\Omega(n^2)$ space.

Claim 1: OneWayEdges $\leq$ Edge.

Proof. Given an streaming algorithm A for Edge, Alice may run A on an initially empty graph G with a stream of edges for each $e \in E$. Alice then sends her local memory to Bob. Bob initializes a set $S = \varnothing$. From algorithm A and its current state, Bob obtains edge e and updates S to $S \cup \{e\}$. Then Bob continues the run of A with deletion of e, obtains e' from A (which is distinct from S), and update S to $S \cup \{e'\}$. Bob repeats this deletion and acquiring edge from A a total of $\frac{1}{10}\binom{n}{2}$ times and return the resulting S.

Note that the above algorithm has communication equal to space used by A, and by a union bound we know that this algorithm has failure probability $O(\frac{1}{n})$. ■

Claim 2: Every pseudo-deterministic OneWayEdges protocol which succeeds with probability at least $\frac{2}{3}$ requires $\Omega(n^2)$ bits of communication.

Proof. Given a bijection from possible edges on n vertices to $\{1, \cdots, \binom{n}{2}\}$ (which is easily defined), OneWayEdges is identical to the problem One-Way-Partial-Recovery with inputs from the set $[\binom{n}{2}]$, defined by Goldwasser et al, 2019 [8]. By Claim 3 from this same paper, every pseudo-deterministic OneWayEdges protocol which succeeds with probability at least $\frac{2}{3}$ requires $\Omega(\binom{n}{2}) = \Omega(n^2)$ bits of communication. ■

Combining the above gives us the proposition. ■

Corollary 4.2. *Any pseudo-deterministic algorithm A such that, if G has at least one edge, then $A(G)$ contains at least one edge in G, requires $\Omega(n^2)$ space.*

Proof. Given A, consider the algorithm A' that returns the smallest edge in lexicographical order (any ordering works) in $A(G)$. Then A' is a pseudo-deterministic algorithm for Edge with the same space requirements and error probability as A, and we obtain our lower bound from Proposition 1. ■

Note that Corollary 1 essentially captures the spanning forest and longest path problems, as those problems return an edge in G as long as G has an edge. It is interesting to note that if $A(G)$ contains edges *not* in G, then Corollary 1 no longer applies; exploring this further could be interesting.

The proof to show that IdenticalSubgraph and ManyIdenticalSubgraph need $\Omega(n^2)$ space requires a slightly different reduction, but the proof format is essentially the same as in Proposition 1.

4.2 HasTriangle requires $\Omega(n^2)$ space

We have established that finding an edge is requires quadratic space. It is therefore unsurprising that finding a triangle is requires quadratic space as well. Formally, we have the following proposition.

Proposition 4.3. *Let* HasTriangle *be the graph streaming problem that returns an arbitrary triangle (3-cycle) from graph G (and $\perp$ if G has no triangles). Any pseudo-deterministic algorithm requires $\Omega(n^2)$ space.*

Proof. Similar to the proof for EDGE, we establish a reduction ONEWAYEDGES $\leq$ EDGE. To achieve this, consider the problem ONEWAYEDGESPROMISE, which is exactly ONEWAYEDGES, except it is promised that vertex n is isolated.

Definition 4.4 (ONEWAYEDGESPROMISE)**.** ONEWAYEDGESPROMISE is the following communication problem: Alice initially has a graph $G = (V, E)$ on n vertices containing edge set E of size $\frac{3}{4}\binom{n}{2}$, and vertex n is isolated. Bob initially has nothing. Alice sends a single message to Bob, and Bob desires to return a set $S \subseteq E$ of size $\frac{1}{10}\binom{n}{2}$.

We shall proceed with two reductions.

Claim 1: ONEWAYEDGES $\leq$ ONEWAYEDGESPROMISE, i.e. given any pseudo-deterministic algorithm for ONEWAYEDGESPROMISE with space s and failure probability p, we can construct a pseudo-deterministic algorithm for ONEWAYEDGES with space s and failure probability p.

Proof. Suppose there is a pseudo-deterministic algorithm A for ONEWAYEDGESPROMISE. In order to solve ONEWAYEDGES, Alice includes an additional auxiliary vertex v_{n+1} that is isolated. Then running A works. ■

Claim 2: ONEWAYEDGESPROMISE $\leq$ HASTRIANGLE, i.e. given any pseudo-deterministic streaming algorithm for HASTRIANGLE with space s and failure probability $O(\frac{1}{n^3})$, we can construct a pseudo-deterministic algorithm for the communication problem ONEWAYEDGESPROMISE with communication s and failure probability $O(\frac{1}{n})$

Proof. Given a pseudo-deterministic streaming algorithm A for HASTRIANGLE, Alice may run A on an initially empty graph G with a stream of edges for each $e \in E$ and send her local memory M to Bob. It is promised that v_n is isolated. Bob first initializes a collection of edges found, $S = \emptyset$, and a list of edges that have been checked, $C = \emptyset$. Then, starting from the state received from Alice, Bob does the following using A:

- While a triangle can be found, include the edges of the triangle into S and C. Update the state by removing these edges and repeat until no triangles found.
- While there is an edge (u, v) amongst vertices in $V \setminus \{v_n\}$ that is not checked, check whether it exists by inserting (u, v_n) and (v, v_n), and querying for the existence of a triangle. If it exists, add (u, v) to S. Regardless of the outcome, add (u, v) to C and update the state by removing (u, v_n) and (v, v_n). Repeat until all possible edges are checked.

Using the above, Bob is able to recover all edges of A. Note that the above algorithm has communication equal to space used by A. By a union bound, the overall failure probability is in $O(\frac{1}{n})$. ■

The two reductions give us the desired result, as it has been established in the previous section that every pseudo-deterministic ONEWAYEDGES peotocol which succeeds with probability at least $\frac{2}{3}$ requires $\Omega(n^2)$ bits of communication. ■

4.3 MinColoring Requires $\Omega(n^2)$ space

Another property of graphs that we are often interested in are colorings, applied to areas such as data mining, image segmentation, and networking.

To this end, consider the MINCOLORING problem:

Definition 4.5 (MINCOLORING)**.** Given a graph G, return a coloring that only uses $\chi(G)$ colors.

Finding a coloring that uses the minimum number of colors is NP-hard, so we might expect that producing such a coloring with space instead of computational constraints would also be hard, even pseudo-deterministically.

We show lower bound for MINCOLORING under the assumption that the algorithm for MINCOLORING is robust to negative edges, which means that it treats edges with negative multiplicity as non-existent.

Proposition 4.6. *Any pseudo-deterministic algorithm that is robust to negative edges, that solves* MinColoring *with constant probability of success requires at least* $\Omega(n^2)$ *space.*

To prove Proposition 4.6, we rely on the following hard communication problem OneWayIndexEdge:

Definition 4.7 (OneWayIndexEdge)**.** Alice has edges S of a graph G on n vertices. Bob has a single edge e of G. Alice can send a single message to Bob, and Bob has to output whether Alice has edge e.

Recall the Index problem, which is the one-way communication where Alice has a bitstring x of length n and Bob has an index i, and Bob has to tell if x_i is 0 or 1. Then we can see that OneWayIndexEdge is equivalent to the Index problem, except now we have $\binom{n}{2}$ instead of n bits. Recall the following lower bound for Index:

Theorem 4.8. *The randomized one-way communication complexity of* Index *is* $\Omega(n)$

This immediately implies the following lower bound for OneWayIndexEdge:

Proposition 4.9. *Any randomized protocol that succeeds with constant probability on* OneWayIndexEdge *requires at least* $\Omega(n^2)$ *communication complexity.*

With this, we can prove 4.6.

Proof of Proposition 4.6. Suppose there exists some algorithm $\mathcal{A}$ that does this in $\tilde{o}(n^2)$ space. Then we show a reduction from OneWayIndexEdge to MinColoring.

Alice will stream all her edges S into $\mathcal{A}$, and end up with some state $M = \mathcal{A}(S)$, which requires $\tilde{o}(n^2)$ space.

She can then send her state M to Bob. Bob's goal now is to figure out if Alice has e. He will run $\mathcal{A}(S)$ by continuing from M, and stream updates to remove all $\binom{n}{2} - 1$ edges in the graph except for e.

Bob then asks the algorithm to output the coloring of the graph. If the coloring only uses a single color, output that Alice does not have e. Otherwise, if the coloring uses two colors, output that Alice has e. Otherwise if more than 2 colors is being used, then the randomized algorithm has run into the error case so output any answer at random.

This works since if Alice has e, then when all edges except e has been removed, it must be the case that the coloring still uses two colors, in particular when coloring the vertices of e. Otherwise, if Alice does not have e, then Bob would have removed all of her edges, and so only 1 color is sufficient to color all the vertices.

This therefore translates into a constant probability of success for OneWayIndexEdge using $o(n^2)$ space, a contradiction to Proposition 4.9. ∎

Note that we had to use the property that the algorithm is robust to negative edges, since Bob streams deletions of all edges except e, possibly resulting in negative edges. This is a strong requirement, as many algorithmic techniques for the streaming model rely on homomorphic sketches, but treating negative edges as 0 is a non-homomorphic property. To further illustrate this property, we consider what happens if we further strengthen the robustness requirements of the algorithms.

4.4 Lower Bounds for Algorithms with Stronger Robustness Requirements

For this section, consider robust graph-streaming algorithms that treat negative edges as non-existent.

Proposition 4.10. *Any randomized algorithm for* Edge *requires* $\Omega(n^2)$ *space.*

Note that the above proposition does not involve pseudo-randomness. This is in contrast with [6], which showed a support element (including negative) of can be sampled in polylogarithmic space using a random algorithm if within the stream, the count for each element never decreases below 0. The robustness which we require our algorithm to possess greatly increases the amount of space required.

To prove the proposition, we shall construct a reduction. Consider the following communication problem.

Definition 4.11 (OneWayFullRecovery)**.** Alice has a $\binom{n}{2}$-bit string, and Bob desires to recover the entire string.

Any randomized communication protocol for OneWayFullRecovery require $\Omega(n^2)$ bits of communication.

Proof of Proposition 4.10. We claim that OneWayFullRecovery $\leq$ Edge, i.e. given any streaming algorithm for Edge with space s and failure probability $O(n^{-3})$, we can construct a pseudo-deterministic algorithm for the communication problem OneWayFullRecovery with communication s and failure probability $O(\frac{1}{n})$

Consider a randomized algorithm A for Edge with error probability $O(n^{-3})$. View Alice's bit string as an indicator for the existence of each edge. Alice runs A on a stream consisting these edges and sends the resulting memory state M to Bob. For every possible edge $e \in V \times V$, Bob checks if $e \in E$ by doing the following: Starting from M, stream updates to remove all $\binom{n}{2} - 1$ possible edges except e. Check if an edge exists. If there is, then $e \in E$, else $e \notin E$.

Thus Bob can fully recover E, and therefore recover the original bit string. Union-bounding the error probabilities, the overall error probabilty is $O(\frac{1}{n})$. ■

In fact, this strategy generalizes a multitude of graph problems, not just Edge or MinColoring (in the previous subsection). The robustness of the required algorithm allows us to freely remove and add edges, hence we are able to create desired graph structures to test for the existence of a single edge.

5 Upper Bounds

It has been quite pessimistic to see so many results in lower bounds. As a great man once said, "I prefer algorithms to impossibility results". We now present several pseudo-deterministic algorithms. We are particularly interested in pseudo-deterministic algorithms that require much less space than their deterministic counterparts, which represents a large enough efficiency gain to be a good reason to use them for real-world applications.

5.1 FindIsolatedNode Can Be Done in $\tilde{O}(n)$ Space

As a warm-up, consider the FindIsolatedNode problem:

Definition 5.1 (FindIsolatedNode)**.** Let G be a graph with (possibly negative) weighted edges. An edge exists between nodes u and v if the edge $u \sim v$ has non-zero weight, i.e $e(u,v) \neq 0$. The graph stream will comprise edge weight updates in $\{-1,1\}$. Furthermore, the total weight of any edge will never exceed some $M = \text{poly}(n)$ at any point in the stream. The goal is to output any vertex that isolated (i.e has no edges), otherwise output NONE.

We illustrate 5.1 with an example:

Example 5.2. Suppose G is a graph on $n = 5$ vertices, and you stream the following weight updates:

$$(3 \sim 4, +1), (1 \sim 2, +1), (1 \sim 3, +1), (2 \sim 3, +1), (3 \sim 4, -1)$$

Then the weight updates for edge $(3 \sim 4)$ cancels out, so the final graph is just a triangle between vertices 1,2,3. So 4 or 5 are both isolated vertices.

Note that the requirement of allowing edge weights to go negative here is required, otherwise the problem becomes trivially doable in $\tilde{O}(n)$ space since one can simply store a counter for the running sum of the weights of all incident edges, and output any edge that has total sum 0.

Proposition 5.3. *There is a pseudo-deterministic algorithm to solve* FindIsolatedNode *using only* $\tilde{O}(n)$ *bits of space.*

To make space bounds meaningful, we cannot use infinite-precision real numbers. Instead, we assume a model where we can represent any real number up to a precision of $O(\min(1/M^4, 1/n^4))$, recalling that $M \in \text{poly}(n)$ is the maximum value of the weight that the edge will ever take at any point in the stream, which can still be done in $O(\log n)$ bits of space.

Proof of Proposition 5.3. We begin by sampling a uniformly random Gaussian vector $\mathbf{x}$ of length n, $\mathbf{x} \in \mathcal{N}(\mathbf{0}, \mathbf{I}_n)$. Initialize another vector $\boldsymbol{y} \in \mathbb{N}^n$ to the zero vector.

Whenever we stream an update $(u \sim v, w)$ at time t, update $\boldsymbol{y}$ by the following update rule:

$$\boldsymbol{y}_t \coloneqq \boldsymbol{y}_{t-1} + w \cdot (e_u e_v^\top + e_v e_u^\top)\boldsymbol{x},$$

where $\boldsymbol{y}_t, \boldsymbol{y}_{t-1}$ refers to the values of y at time t and $t-1$ respectively.

Once the end of the stream has been reached, output the first zero entry in y when rounded to a precision of $1/8n$, or NONE otherwise.

Now we analyze why the algorithm is correct. Let $\boldsymbol{A}$ represent the adjacency matrix of G at the end, so we have $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$. We show that the algorithm makes a one-sided error.

Suppose the ith row of $\boldsymbol{A}$ was indeed all zero at the end, i.e $\boldsymbol{A}_i = \mathbf{0}$. Then it is always the case that $\boldsymbol{y}_i = 0$.

However, when the ith row of $\boldsymbol{A}$ is not actually zero, it is possible for $\boldsymbol{y}_i = 0$, and cause us to output the wrong answer. We upper bound the probability of this occurring.

At the end, for each row i, we have $\boldsymbol{y}_i = \langle \boldsymbol{A}_i, \boldsymbol{x} \rangle$. But this is just the sum of n scaled Gaussians, each $\boldsymbol{x}_j$ by $\boldsymbol{A}_{i,j}$. Since the sum of independent Gaussians is another Gaussian, we have that $\boldsymbol{y}_i \sim \mathcal{N}\left(\mathbf{0}, \|\boldsymbol{A}_i\|_2^2\right)$. We incur $O(1/M^4)$ rounding error for each Gaussian that was generated, and so since we have n such Gaussians, the squared norm of each entry is at most M^2, and hence that of each row is at most M^3, which means that our error for each entry $\boldsymbol{y}_i$ is at most $O(1/M)$.

Then since we round to the nearest $1/8n$, the probability that we output 0 when $\boldsymbol{A}_i \neq 0$ can be upper bounded by the PDF of $\mathcal{N}\left(\mathbf{0}, \|\boldsymbol{A}_i\|_2^2\right)$ multiplied by the area of the possible error, i.e,

$$\begin{aligned}
\mathbf{Pr}[y_i = 0 \text{ when rounded }] &= 2 \cdot \frac{1}{8n} \cdot \frac{1}{\sqrt{2\pi}\|A_i\|_2} \exp\left(-\frac{1}{2}\left(\frac{0-0}{\|A_i\|_2}\right)^2\right) \\
&= \frac{1}{4\sqrt{2\pi}n\|A_i\|_2} \\
&\leq \frac{1}{4n},
\end{aligned}$$

recalling that $\|A_i\|_2 \geq 1$ by assumption.

Then by union bounding over all n coordinates of $\boldsymbol{y}$, we have that the probability that any of them fails is $1/4$, which gives us a success probability of at least $3/4$ for the entire algorithm.

Since our analysis of success probability requires all coordinates to not make an error, we can see that this algorithm is also pseudo-deterministic, since we always output the first zero coordinate in $\boldsymbol{y}$.

Furthermore, since we only keep track of n coordinates in $\boldsymbol{y}$ and $\boldsymbol{x}$ with a finite precision of at most $O(1/M^4)$, this only requires $O(n \log n)$ bits of storage.

This completes the proof of a $\tilde{O}(n)$-space algorithm for FindIsolatedNode. ∎

5.2 l_2 Norm Estimation

It is an open problem whether there exists a pseudo-deterministic algorithm for l_2-norm estimation that uses only $O(\text{poly}\log n)$ space.

We attempt to make progress towards answering this question by approaching the simpler problem of whether pseudo-deterministic l_2 norm estimation can be performed with polylog space when we impose weak conditions on the l_2 norms of the inputs. This motivates the Promise-l_p-NormEstimation problem:

Definition 5.4 (Promise-l_p-NormEstimation)**.** Consider the turnstile streaming model with insertions and deletions of entries of a vector $\mathbf{x}$. Suppose you are promised that the end, the l_p norm of x is one of the members of the set $S = \{n_1, n_2, \cdots, n_s\}$, and each member of the set is in $\text{poly}(n)$. Return an estimate of the l_p norm of $\mathbf{x}$ up to a multiplicative error of $(1+\varepsilon)$.

To construct and prove our low space algorithm for Promise-l_p-NormEstimation , we rely on the following lemma:

Lemma 5.5. *Given $S = \{y_1, y_2, \cdots, y_s\}$, there exists for some $u \in [1/2, 1]$ with the property that for all $y \in S$, $y \notin (1+\varepsilon/2)^i(1 \pm \varepsilon/s)$ for any $i \in \mathbb{Z}$. In fact, if $u \sim \mathcal{U}(1/2, 1)$, the probability that this holds is at least 3/4.*

Proof of Lemma 5.5. Suppose we are given such an $S = \{y_1, y_2, \cdots, y_s\}$. Fix a $y \in S$. Call $u \in [1/2, 1]$ bad if y falls into the range $u \cdot (1+\varepsilon/2)^i(1 \pm \epsilon/16s)$ for any i.

We claim that the values of u that are bad in $[1/2, 1]$ all belong to open intervals, and there are not too many of them, and that each of these intervals are small. As such, the probability that u is bad over the randomness of choosing u for y is small.

To see the first claim, consider the family of continuous functions $f_i : [1/2, 1] \to \mathbb{R}, f_i(u) = u \cdot (1+\varepsilon/2)^i$. The range can only include y if $y \in (1/2(1+\varepsilon/2)^i, (1+\varepsilon/2)^i)$. Notice that the end point of the range is a factor of 2 of the start point, so the total possible number of such i's can be upper bounded by how many times we can continue incrementing the power i while still staying within a factor of 2. Therefore, the total number of such i such that $y \in f_i([1/2, 1])$ is $\log_{1+\varepsilon/2} 2$, which we can upper bound by using the fact that $1 + k\varepsilon \leq (1+\varepsilon)^k$ as $1/(\varepsilon/2) = 2/\varepsilon$.

To see the second claim, since $y \in u \cdot (1+\varepsilon)^i(1 \pm \varepsilon/16s)$, equivalently we can see that $y \in u' \cdot (1+\varepsilon)^i$ where $u' \in (1 \pm \varepsilon/16s) \cdot u$. Therefore each interval has probability $2\varepsilon u/16s \leq \varepsilon/8s$, where the 2 comes from normalizing over the interval $[1/2, 1]$.

Thus overall the probability that u is bad for y is upper bounded by the upper bound of the product of the number of bad intervals and the probability of each of the intervals, i.e $2/\varepsilon \times \varepsilon/8s = 1/4s$.

Then we can show that there must exist some choice of u which is good by union bounding over all the s possible choices from S, which gives that all the bad u only has probability at most 1/4, meaning that at least 3/4 of the choices of u must be good. ∎

Theorem 5.6. *There is a pseudo-deterministic algorithm for* PROMISE-l_p-NORMESTIMATION *for $p = 2$ which only uses $O(s^2 \cdot \text{poly}\left(\frac{\log n}{\varepsilon}\right))$ space.*

Proof of Theorem 5.6. We present the following algorithm for solving PROMISE-l_p-NORMESTIMATION for $p = 2$ case.

Maintain a sketch of the streamed updates using $t = 64s^2$ independent CountSketch matrices, each with $O(1/\epsilon^2)$ rows and preserving l_2 norms up to a multiplicative factor of $(1 \pm \frac{\varepsilon}{2})$.

Construct $\mathbf{x}'$ by taking the median entries of each row of the t sketches. This gives us that with a probability of $1 - \text{poly}(n)$, the estimated norm has the property that

$$\left(1 - \frac{\varepsilon}{2\sqrt{t}}\right)\|\mathbf{x}\|_2 \leq \|\mathbf{x}'\|_2 \leq \left(1 + \frac{\varepsilon}{2\sqrt{t}}\right)\|\mathbf{x}\|_2 \tag{1}$$

$$\equiv \left(1 - \frac{\varepsilon}{16s}\right)\|\mathbf{x}\|_2 \leq \|\mathbf{x}'\|_2 \leq \left(1 + \frac{\varepsilon}{16s}\right)\|\mathbf{x}\|_2, \tag{2}$$

where the $1/\sqrt{t}$ error reduction comes from applying Chebyshev's inequality over the independent CountSketch sketches.

Sample $u \sim \mathcal{U}(1/2, 1)$ uniformly at random. Find the smallest i (possibly negative) with the property that $\|\mathbf{x}'\|_2 \leq u \cdot (1+\varepsilon/2)^i$, i.e rounding up to the nearest power of $(1+\varepsilon/2)$ scaled by u. Output $u \cdot (1+\varepsilon/2)^i$ as the answer.

We now analyze the correctness of this algorithm.

First see that we always output a $(1+\varepsilon)$ multiplicative approximation of $\|\mathbf{x}\|_2$, since we round to a power of $(1+\varepsilon/2)$ multiplied by a constant in $(0, 1]$. Therefore since the estimation error has factor at most $1 \pm \varepsilon/2$, then our overall error is not greater than $\varepsilon/2 + \varepsilon/2 = \varepsilon$.

Next, this is also pseudo-deterministic. This is because even though the estimated $\|\mathbf{x}'\|_2$ can vary as $\|\mathbf{x}'\|_2 \in \left(1 \pm \frac{\varepsilon}{16s}\right)y$, by Lemma 5.5, we have that with probability at least 3/4 over the draw of u, there is the good event that only one possible i that can be output for any $y \in S$. This means that conditioned on that good event happening, $\|\mathbf{x}'\|$ will never fall into the boundary case where it straddles the range $u \cdot (1+\varepsilon/2)^i(1 \pm \varepsilon/16s)$ and cause the algorithm to possibly output either $u \cdot (1+\varepsilon/2)^i$ or $u \cdot (1+\varepsilon/2)^{i+1}$.

By union bounding over both failure probabilities, we have that the algorithm always outputs the same answer with constant probability.

Finally, we analyze the space bounds. We need to store the result of $O(s^2)$ sketches, each of which has $O(1/\epsilon^2)$ rows. We need $O(\log n)$ bits to store the result for each of them. This gives an overall space usage of $O(s^2 \cdot \text{poly}\left(\frac{\log n}{\varepsilon}\right))$. ■

5.3 Counting Triangles

One other well-studied graph streaming problem is that of counting the number of triangles, T_3, in a graph, in an insertion-deletion stream. It is known that deciding if a graph is triangle-free requires $\Omega(n^2)$ space [5], so we restrict our attention to the case when the number of triangles is guaranteed to be at least some given lower threshold t. That is, we are promised that $t \le T_3$ at the end of a stream.

Theorem 5.7. *There is an 8-pseudo-deterministic algorithm A that provides a multiplicative $(1+\epsilon)$ estimate for the number of triangles T_3 in $O(\epsilon^{-2}((mn)/t)^2)$ space, as long as $T_3 \ge t$. That is, given a fixed graph stream, the algorithm outputs one of 8 values within $(1 \pm \epsilon)T_3$ with probability at least 9/10.*

Proof. We first reiterate ideas from [4]. We define $v \in \{0,1,2,3\}^{\binom{n}{3}}$ to be a vector whose entries represent the number of edges present in a 3-subset of $[n]$. Then T_3 is just the number of entries in v equaling 3. Now define frequency moments $F_n = \sum_i v_i^n$, where F_0 specially represents l_0 norm of v, which is the number of nonzero entries in v. It can be observed that

$$T_3 = F_0 - 1.5F_1 + 0.5F_2.$$

We can then approximate T_3 with estimates of each of $\{F_0, F_1, F_2\}$. It is known that for $n \in \{0,1,2\}$, an $O(\delta^{-2}\log n)$ space randomized algorithm exists to approximate the moments multiplicatively up to $1 \pm \delta$. It can then be shown that we can use the formula above to approximate T_3 up to $1 \pm \epsilon$ by setting $\delta = \epsilon/(8mn)$.

Claim: There exists a 2-pseudo-deterministic algorithm for estimating l_p norm up to $(1 \pm \epsilon)$ using $O(\epsilon^{-2}\log n)$ space, where $p \in \{0,1,2\}$.

Proof. This is in spirit the same as the arguments from the previous section. Let A be the randomized algorithm that achieves $(1 \pm \epsilon/5)$ l_p norm estimation using the same (asymptotic) space bound. Our 2-pseudo-deterministic algorithm may simply output the result r outputted by A, rounded up to the smallest integer power of $1+\epsilon/2$ larger than r. Since there can be at most one power of $(1+\epsilon/2)$ covered by a multiplicative $(1 \pm \epsilon/5)$ interval and overall our estimate is still up to $(1 \pm \epsilon)$ from l_p norm, we have our desired 2-psuedo-deterministic algorithm. ■

Using this claim, we may have a separate 2-pseudo-deterministic algorithm for each of l_0, l_1, l_2 norm estimation. Hence in total the derived triangle estimation algorithm outputs one of 8 results with high probability, making it 8-psuedo-deterministic. ■

6 Future Directions

Here is a list of problems we may seek to address next.

- Is there a low-space pseudo-deterministic algorithm for counting triangles? We showed that it is possible to do so if we allow a constant number of possible outputs, but it is not known if we can still do this when we allow only one unique output with high probability.
- Is there a general l_p norm estimation algorithm that is pseudo-deterministic? A weaker result would be to show a pseudo-deterministic algorithm for the promise version where the norm of the inputs can take a discrete set of values whose cardinality is polynomial in n. This would imply a pseudo-deterministic algorithm for counting triangles, but this is not implied by our conclusions from norm estimation (since, for example, linear in n set size would imply a $\Omega(n)$ storage space under the same method).
- What are the possible pseudo-deterministic algorithms for graph streaming problems if we have multiple passes? We may save significant space for many problems if we are allowed more than one pass over the inputs.

References

[1] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. “Analyzing graph structure via linear measurements”. In: *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*. SIAM. 2012, pp. 459–467.

[2] Sepehr Assadi, Andrew Chen, and Glenn Sun. “Deterministic Graph Coloring in the Streaming Model”. In: *CoRR* abs/2109.14891 (2021). arXiv: 2109.14891. URL: https://arxiv.org/abs/2109.14891.

[3] Sepehr Assadi, Yu Chen, and Sanjeev Khanna. “Sublinear Algorithms for $(\Delta + 1)$ Vertex Coloring”. In: *CoRR* abs/1807.08886 (2018). arXiv: 1807.08886. URL: http://arxiv.org/abs/1807.08886.

[4] Ziv Bar-Yossef, Ravi Kumar, and D Sivakumar. “Reductions in streaming algorithms, with an application to counting triangles in graphs”. In: *SODA*. Vol. 2. 2002, pp. 623–632.

[5] Vladimir Braverman, Rafail Ostrovsky, and Dan Vilenchik. “How hard is counting triangles in the streaming model?” In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2013, pp. 244–254.

[6] Gereon Frahling, Piotr Indyk, and Christian Sohler. “Sampling in dynamic data streams and applications”. In: vol. 18. 01n02. 2008, pp. 3–28. DOI: 10.1142/s0218195908002520.

[7] Eran Gat and Shafi Goldwasser. “Probabilistic Search Algorithms with Unique Answers and Their Cryptographic Applications.” In: *Electron. Colloquium Comput. Complex.* Vol. 18. 2011, p. 136.

[8] Shafi Goldwasser et al. “Pseudo-deterministic streaming”. In: *arXiv preprint arXiv:1911.11368* (2019).

[9] John Kallaugher and Eric Price. “Improved graph sampling for triangle counting”. In: *CoRR* abs/1610.02066 (2016). arXiv: 1610.02066. URL: http://arxiv.org/abs/1610.02066.

[10] Andrew McGregor. “Graph stream algorithms: a survey”. In: *ACM SIGMOD Record* 43.1 (2014), pp. 9–20.

[11] Andrew McGregor, Sofya Vorotnikova, and Hoa T. Vu. “Better Algorithms for Counting Triangles in Data Streams”. In: *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. PODS ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 401–411. ISBN: 9781450341912. DOI: 10.1145/2902251.2902283. URL: https://doi.org/10.1145/2902251.2902283.

[12] Mikkel Thorup. “Near-optimal fully-dynamic graph connectivity”. In: *Proceedings of the thirty-second annual ACM symposium on Theory of computing*. 2000, pp. 343–350.