Near-Optimal Two-Pass Streaming Algorithm for Sampling Random Walks over Directed Graphs

Lijie Chen
MIT, Cambridge, MA, USA

Gillat Kol
Princeton University, NJ, USA

Dmitry Paramonov
Princeton University, NJ, USA

Raghuvansh R. Saxena
Princeton University, NJ, USA

Zhao Song
Institute for Advanced Study, Princeton, NJ, US

Huacheng Yu
Princeton University, NJ, USA

Abstract

For a directed graph G with n vertices and a start vertex u_{start}, we wish to (approximately) sample an L-step random walk over G starting from u_{start} with minimum space using an algorithm that only makes few passes over the edges of the graph. This problem found many applications, for instance, in approximating the PageRank of a webpage. If only a single pass is allowed, the space complexity of this problem was shown to be $\widetilde{\Theta}(n \cdot L)$. Prior to our work, a better space complexity was only known with $\widetilde{O}(\sqrt{L})$ passes.

We essentially settle the space complexity of this random walk simulation problem for *two-pass* streaming algorithms, showing that it is $\widetilde{\Theta}(n \cdot \sqrt{L})$, by giving almost matching upper and lower bounds. Our lower bound argument extends to every constant number of passes p, and shows that any p-pass algorithm for this problem uses $\widetilde{\Omega}(n \cdot L^{1/p})$ space. In addition, we show a similar $\widetilde{\Theta}(n \cdot \sqrt{L})$ bound on the space complexity of any algorithm (with *any* number of passes) for the related problem of sampling an L-step random walk from *every* vertex in the graph.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases streaming algorithms, random walk sampling

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.52

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2102.11251` [11]

Funding *Lijie Chen*: Lijie Chen is supported by an IBM Fellowship.
Zhao Song: Zhao Song is supported in part by Schmidt Foundation, Simons Foundation, NSF, DARPA/SRC, Google and Amazon AWS.

Acknowledgements We would like to thank Rajesh Jayaram for discussions on ℓ_1 heavy hitters.

1 Introduction

1.1 Background and Motivation

Graph streaming algorithms. *Graph streaming algorithms* have been the focus of extensive study over the last two decades, mainly due to the important practical motivation in analyzing potentially *huge structured data* representing the relationships between a set of entities (*e.g.*, the link graph between webpages and the friendship graph in a social network). In the graph streaming setting, an algorithm gets access to a sequence of graph edges given in an arbitrary order and it can read them one-by-one in the order in which they appear in the sequence. The goal here is to design algorithms solving important graph problems that only make one or few passes through the edge sequence, while using as little memory as possible.

Much of the streaming literature was devoted to the study of *one-pass* algorithms and an $\Omega(n^2)$ space lower bound for such algorithms was shown for many fundamental graph problems. A partial list includes: maximum matching and minimum vertex cover [15, 19], s-t reachability and topological sorting [7, 15, 22], shortest path and diameter [15, 16], maximum and (global or s-t) minimum cut [39], maximal independent set [3, 13], and dominating set [4, 14].

Recently, the *multi-pass* streaming setting received quite a bit of attention. For some graph problems, allowing a few passes instead of a single pass can reduce the memory consumption of a streaming algorithm dramatically. In fact, even a single additional pass over the input can already greatly enhance the capability of the algorithms. For instance, minimum cut and s-t minimum cut in undirected graphs can be solved in two passes with only $\widetilde{O}(n)$ and $O(n^{5/3})$ space, respectively [33] (as mentioned above, any one-pass algorithm for these problems must use $\Omega(n^2)$ space). Additional multi-pass algorithms include an $O(1)$-pass algorithm for approximate matching [17, 19, 28, 29], an $O(\log \log n)$-pass algorithm for maximal independent set [3, 13, 18], and $O(\log n)$-pass algorithms for approximate dominating set [4, 8, 21] and weighted minimum cut [30].

Simulating random walks on graphs. Simulating random walks on graphs is a well-studied algorithmic problem with may applications in different areas of computer science, such as connectivity testing [32], clustering [1, 2, 10, 36], sampling [26], generating random spanning tree [35], and approximate counting [25]. Since most applications of random-walk simulation are concerned with huge networks that come from practice, it is of practical interest to design low-space graph streaming algorithms with few passes for this problem.

In an influential paper by Das Sarma, Gollapudi and Panigrahy [34], an $\widetilde{O}(\sqrt{L})$-pass and $\widetilde{O}(n)$ space algorithm for simulating L-step random walks on directed graphs was established. (Streaming algorithms with almost linear space complexity, like this one, are often referred to as *semi-streaming* algorithms). Using this algorithm together with some additional ideas, [34] obtained space-efficient algorithms for estimating *PageRank* on graph streams. Recall that the PageRank of a webpage corresponds to the probability that a person that randomly clicks on web links arrives at this particular page[1]. However, scanning the sequence of edges $\widetilde{O}(\sqrt{L})$ times may be time-inefficient in many realistic settings.

In the one-pass streaming setting, a folklore algorithm with $\widetilde{O}(n \cdot L)$ space complexity for simulating L-step random walks is known [34] (see Subsection 2.1 for a description of this algorithm), and it is proved to be optimal [27]. We mention that the work of [27]

[1] Given a web-graph $G = (V, E)$ representing the webpages and links between them, the PageRank of the vertices satisfy $\mathsf{PageRank}(u) = \sum_{(v,u)\in E} \mathsf{PageRank}(v)/d(v)$, simultaneously for all u, where $d(\cdot)$ denotes the out-degree, [6].

also considers random walks on *undirected graphs*, and shows that $\widetilde{\Theta}(n \cdot \sqrt{L})$ space is both necessary and sufficient for simulating L-step random walks on undirected graphs with n vertices in one pass.

Both of these known algorithms for general directed graphs have their advantages and disadvantage (either requiring many passes or more space). A natural question is whether one can interpolate between these two results and obtain an algorithm with pass complexity much smaller than $\sqrt{L}$, yet with a space complexity much smaller than $n \cdot L$. Prior to our work, it was not even known if an $o(\sqrt{L})$-pass streaming algorithm with $n \cdot L^{0.99}$ space is possible.

1.2 Our Results

We answer the above question in the affirmative by giving a two-pass streaming algorithm with $\widetilde{O}(n \cdot \sqrt{L})$ space for sampling a random walk of length L on a directed graph with n vertices. We complement this result by an almost matching $\widetilde{\Omega}(n \cdot \sqrt{L})$ lower bound on the space complexity of every two-pass streaming algorithm for this problem. In fact, our two-pass lower bound generalizes to an $\widetilde{\Omega}(n \cdot L^{1/p})$ lower bound on the space consumption of any p-pass algorithm, for a constant p.

1.2.1 Two-Pass Algorithm for Random Walk Sampling

For a directed graph $G = (V, E)$, a vertex $u_{\mathsf{start}} \in V$ and a non-negative integer L, we use $\mathsf{RW}_L^G(u_{\mathsf{start}})$ to denote the distribution of L-step random walks $(v_0, \ldots, v_L)$ in G starting from $v_0 = u_{\mathsf{start}}$ (see Subsection 3.2 for formal definitions). For a distribution $\mathcal{D}$ over a finite domain Ω, we say that a randomized algorithm *samples* from $\mathcal{D}$ if, over its internal randomness, it outputs an element $\omega \in \Omega$ distributed according to $\mathcal{D}$. We give a space-efficient streaming algorithm for (approximate) sampling from $\mathsf{RW}_L^G(u_{\mathsf{start}})$ with small error:

▶ **Theorem 1** (Two-pass algorithm). *There exists a streaming algorithm $\mathbb{A}_{\mathsf{two\text{-}pass}}$ that given an n-vertex* directed *graph $G = (V, E)$, a starting vertex $u_{\mathsf{start}} \in V$, a non-negative integer L indicating the number of steps to be taken, and an error parameter $\delta \in (0, 1/n)$, satisfies the following conditions:*

1. *$\mathbb{A}_{\mathsf{two\text{-}pass}}$ uses at most $\widetilde{O}(n \cdot \sqrt{L} \cdot \log \delta^{-1})$ space*[2] *and makes two passes over the input graph G.*
2. *$\mathbb{A}_{\mathsf{two\text{-}pass}}$ samples from some distribution $\mathcal{D}$ over V^{L+1} satisfying $\|\mathcal{D} - \mathsf{RW}_L^G(u_{\mathsf{start}})\|_{\mathrm{TV}} \leq \delta$.*

Our algorithm can also be generalized to the *turnstile* model, paying a $\operatorname{poly}\log n$ factor in the space usage. See Subsection 4.4.

Observe that our algorithm $\mathbb{A}_{\mathsf{two\text{-}pass}}$ allows for a considerable saving in space compared to the folklore single-pass algorithm ($\widetilde{O}(n \cdot \sqrt{L})$ vs $\widetilde{O}(n \cdot L)$) and considerable saving in the number of passes compared to [34] (2 vs $\widetilde{O}(\sqrt{L})$), at least if we allow some small error δ.

We mention that $\mathbb{A}_{\mathsf{two\text{-}pass}}$ can also be used to sample a random path from *every* vertex[3] of G with the same storage cost of $\widetilde{O}(n \cdot \sqrt{L} \cdot \log \delta^{-1})$ and two passes[4]. This is because $\mathbb{A}_{\mathsf{two\text{-}pass}}$ satisfies the useful property of *obliviousness to the starting vertex* u_{start}, meaning

[2] The $\widetilde{O}$ hides logarithmic factors in n. We may assume without loss of generality that $L \leq n^2$, as otherwise $n \cdot \sqrt{L} > n^2$ and that algorithm can store the entire input graph.

[3] Note, however, that the random walks from different vertices in the graph may be correlated.

[4] We count towards the space complexity only the space on the work tape used by the algorithm and do not count space on the output tape (otherwise an $\Omega(n \cdot L)$ lower bound is trivial).

that it scans the input graph before the start vertex is revealed. More formally, we say that an algorithm $\mathbb{A}$ is oblivious to the starting vertex if it first runs a *preprocessing* algorithm $\mathbb{P}$ and then a *sampling* algorithm $\mathbb{S}$; the algorithm $\mathbb{P}$ reads the input graph stream *without* knowing the starting vertex u_{start} (if $\mathbb{A}$ is a p-pass streaming algorithm, $\mathbb{P}$ makes p passes over the input graph stream), and outputs a string; $\mathbb{S}$ takes both the string outputted by $\mathbb{P}$ and a starting vertex u_{start} as an input, and outputs a walk on the input graph G.

1.2.2 Lower Bounds

We prove the following lower bound:

▶ **Theorem 2** (Multi-pass lower bound). *Fix a constant $\beta \in (0,1]$ and an integer $p \geq 1$. Let $n \geq 1$ be a sufficiently large integer and let $L = \lceil n^{\beta} \rceil$. Any randomized p-pass streaming algorithm that, given an n-vertex* directed *graph $G = (V, E)$ and a starting vertex $u_{\mathsf{start}} \in V$, samples from a distribution $\mathcal{D}$ such that $\|\mathcal{D} - \mathsf{RW}_L^G(u_{\mathsf{start}})\|_{\mathrm{TV}} \leq 1 - \frac{1}{\log^{10} n}$ requires $\widetilde{\Omega}(n \cdot L^{1/p})$ space.*

We remark that the theorem above shows that no low-space algorithms can achieve sampling error $1 - 1/\mathrm{poly}\log(n)$, which is quite strong as usual applications of simulating random walks would require at least a small constant sampling error. Plugging in $p = 2$ in Theorem 2, implies that our two-pass algorithm from Theorem 1 is essentially optimal. Also, with $p = 1$, the theorem reproduces the one-pass lower bound by [27]. In addition, Theorem 2 rules out the possibility of a semi-streaming algorithm with any constant number of passes.

Recall from Subsubsection 1.2.1, that our two-pass algorithm $\mathbb{A}_{\mathsf{two\text{-}pass}}$ utilizes $\widetilde{O}(n \cdot \sqrt{L})$ space and is oblivious to the starting vertex. Interestingly, we are able to show that *any* oblivious algorithm for random walk sampling (with any number of passes) requires $\widetilde{\Omega}(n \cdot \sqrt{L})$ space. Thus, any algorithm for random walk sampling with significantly less space than ours, has to be inherently different and have its storage depend on the starting vertex. Our lower bound for oblivious algorithms also implies that $\mathbb{A}_{\mathsf{two\text{-}pass}}$ gives an almost optimal algorithm for sampling a pass from every start vertex, even if any number of passes are allowed.

▶ **Theorem 3** (Lower bound for oblivious algorithms). *Let $n \geq 1$ be a sufficiently large integer and let L denote an integer satisfying that $L \in [\log^{40} n, n]$. Any randomized algorithm that is* oblivious to the start vertex *and given an n-vertex* directed *graph $G = (V, E)$ and a starting vertex $u_{\mathsf{start}} \in V$, samples from a distribution $\mathcal{D}$ such that $\|\mathcal{D} - \mathsf{RW}_L^G(u_{\mathsf{start}})\|_{\mathrm{TV}} \leq 1 - \frac{1}{\log^{10} n}$ requires $\widetilde{\Omega}(n \cdot \sqrt{L})$ space*[5].

1.3 Discussions and Open Problems

Better space complexity with more passes? Our results leave open a couple of interesting directions for future work. The most significant open question is to understand the streaming space complexity of sampling random walks with more than two passes. In particular, Theorem 2 implies that a three-pass streaming algorithm has space complexity at least $\widetilde{\Omega}(n \cdot L^{1/3})$. Can one get $\widetilde{O}(n \cdot L^{1/3})$ space with three passes, or at least $O(n \cdot L^{1/2-\varepsilon})$ space, for some constant $\varepsilon > 0$? Note that, as explained in Subsubsection 1.2.2, such an algorithm must utilize its knowledge of the starting vertex when it reads the graph stream.

[5] In fact, we show that Theorem 3 holds even if the preprocessing algorithm $\mathbb{P}$ and the sampling algorithm $\mathbb{S}$ are allowed to use an arbitrarily large amount of memory, as long as $\mathbb{P}$ passes a string of length at most (roughly) $n \cdot \sqrt{L}$ to $\mathbb{S}$.

Theorem 2 does not rule out semi-streaming $\widetilde{O}(n)$ space algorithms even when p is a moderately growing function of n and L. In [34], it is shown that such an $\widetilde{O}(n)$ space algorithm exists with $p = \widetilde{O}(\sqrt{L})$ passes. Does a semi-streaming algorithm with, say, $\operatorname{poly}\log(L)$ passes exist?

Undirected graphs? It would also be interesting to see what is the best two-pass streaming algorithm for simulating random walks on *undirected graphs*. Specifically, is it possible to combine our algorithm with the algorithm from [27] to obtain an improvement over the optimal $\widetilde{O}(n \cdot \sqrt{L})$ space complexity of a one-pass streaming algorithm for this problem?

Only outputting the end vertex? Finally, our lower bounds only apply to the case where the algorithms need to output an entire random path $(v_0, \ldots, v_L)$. If instead only the last vertex v_L in the random walk is required, can one design better two-pass algorithms or prove a non-trivial lower bound?

2 Techniques

2.1 The Two-Pass Algorithm

We next overview our two-pass algorithm from Theorem 1, that simulates random walks with only $\widetilde{O}(n \cdot \sqrt{L})$ space.

The folklore one-pass algorithm. Before discussing our algorithm, it would be instructive to review the folklore $\widetilde{O}(n \cdot L)$-space one-pass algorithm for simulating L-step random walks in a directed graph $G = (V, E)$ (for simplicity, we will always assume $L \leq n$ in the discussions). The algorithm is quite simple:

1. For every vertex $v \in V$, sample L of its outgoing neighbors *with replacement* and store them in a list L_v^{save} of length L (that is, for each $j \in [L]$, the j-th element of L_v^{save} is an *independent uniformly random* outgoing vertex of v). This can be done in a single pass over input graph stream using reservoir sampling [37].
2. Given a starting vertex $u_{\mathsf{start}} \in V$, our random walk starts from u_{start} and repeats the following for L steps: suppose we are currently at vertex v and it is the k-th time we visit this vertex, then we go from v to the k-th vertex in the list L_v^{save}.

It is not hard to see that the above algorithm works: whenever we visit a vertex $v \in V$, the next element in the list L_v^{save} will always be a uniformly random outgoing neighbor of v, *conditioned on* the walk we have produced so far; and we will never run out of the available neighbors of v as $|L_v^{\mathsf{save}}| = L$.

A naive attempt and the obstacles. Since we are aiming at only using $\widetilde{O}(n \cdot \sqrt{L})$ space, a naive attempt to improve the above algorithm is to just sample and store $\tau = O(\sqrt{L})$ outgoing neighbors instead of L neighbors, and simulate the walk starting from u_{start} in the same way. The issue here is that, during the simulation of an L-step walk, whenever one visits a vertex v more than τ times, one would run out of available vertices in the list L_v^{save}, and the algorithm can no longer produce a legit random walk. For a simple example, imagine we have a star-like graph where $n - 1$ vertices are connected to a center vertex via two-way edges. An L-step random walk starting at the center would require at least $\Omega(L)$ samples from the center's neighbors, and our naive algorithm completely breaks.

Our approach: heavy and light vertices. Observe, however, that in the above example of a star-like graph, we are only at risk of not storing enough random neighbors of the center node, as an L-step *random* walk would only visit the other non-center vertices a very small number of times. Thus, the algorithm may simply record all edges from the center with only $O(n)$ space. This observation inspires the following approach for a two-pass algorithm:

1. In the first pass, we identify all the vertices that are likely to be visited many times by a random walk (starting from *some* vertex). We call such vertices *heavy*, while all other vertices are called *light*.
2. In the second pass, we record *all* outgoing neighbors of all heavy vertices, as well as $O(\tau)$ random outgoing neighbors with replacement of each of the light vertices.

Observe that the obtained algorithm is indeed *oblivious to the starting vertex*: the two passes described above do not use the starting vertex. Still, given the set of outgoing neighbors stored by the second pass, we are able to sample a random walk from any start vertex.

First pass: how do we detect heavy vertices? The above approach requires that we detect, in a single pass, all vertices v that with a decent probability (say, $1/\text{poly}(n)$), are visited more than $O(\tau)$ times by an L-step random walk. To this end, we observe that if a random walk visits a vertex v more than τ times, this random walk must follow more than $\tau - 1$ self-circles around v in L steps. This, in turn, implies that a random walk that starts from v is likely to return to v in roughly $L/\tau = O(\sqrt{L})$ steps.

The above discussion suggests the following definition of heavy vertices: a vertex v is *heavy*, if a random walk starting from v is likely (say, with probability at least $1/3$) to revisit v in $O(\sqrt{L})$ steps. Indeed, this property is much easier to detect: we can run $O(\log n)$ independent copies of the folklore one-pass streaming algorithms to sample $O(\log n)$ $O(\sqrt{L})$-step random walks starting from v, and count how many of them return to v at some step.

Second pass: can we afford to store the neighbors? In Lemma 18, we show that for a light vertex v, an L-step random walk starting at any vertex visits v $O(\sqrt{L})$ times with high probability. Therefore, in the second pass, we can safely record only $O(\sqrt{L})$ outgoing neighbors for all light vertices. Still, we have to record all the outgoing neighbors for heavy vertices.

The crux of our analysis is a *structural result* about directed graphs, showing that the total outgoing degree of all heavy vertices is bounded by $O(n \cdot \sqrt{L})$, and therefore we can simply store all of their outgoing neighbors. This is proved in Lemma 10, which may also be of independent interest.

Intuition behind the structure lemma. Finally, we discuss the insights behind the above structure lemma for directed graphs. We will use $d_{\mathsf{out}}(v)$ to denote the number of outgoing neighbors of v. For concreteness, we now say a vertex v is heavy if a random walk starting from v revisits v in $\sqrt{L}$ steps with probability at least $1/3$.

Let $V_{\mathsf{heavy}} \subseteq V$ be the set of heavy vertices and let $v \in V_{\mathsf{heavy}}$. By a simple calculation, one can see that for at least a $1/6$ fraction of outgoing neighbors u of v, a random walk starting from u visits v in $\sqrt{L}$ steps with probability at least $1/6$. The key insight is to consider the number of pairs $(u, v) \in V^2$ such that a random walk starting from u visits v in $\sqrt{L}$ steps with probability at least $1/6$. We will use $\mathcal{S}$ to denote this set.

- By the previous discussions, we can see that for each heavy vertex v, it adds at least $1/6$ $d_{\mathsf{out}}(v)$ pairs to the set $\mathcal{S}$. Hence, we have

$$|\mathcal{S}| \geq \frac{1}{6} \cdot \sum_{v \in V_{\mathsf{heavy}}} d_{\mathsf{out}}(v). \tag{1}$$

- On the other hand, it is not hard to see that for each vertex v, there are at most $O(\sqrt{L})$ many pairs of the form $(v, u) \in \mathcal{S}$, since a $\sqrt{L}$-step walk can visit only $\sqrt{L}$ vertices. So we also have

$$|\mathcal{S}| \leq O(n\sqrt{L}). \tag{2}$$

Putting the above (Equation 1 and Equation 2) together, we get the desired bound

$$\sum_{v \in V_{\mathsf{heavy}}} d_{\mathsf{out}}(v) \leq O(n\sqrt{L}).$$

2.2 Lower Bound for p-Pass Algorithms

We now describe the ideas behind the proof of Theorem 2, our $\widetilde{\Omega}(n \cdot L^{1/p})$ space lower bound for p-pass randomized streaming algorithms for sampling random walks. We mention that many of the tools developed for proving space lower bounds are not directly applicable when one wishes to lower bound the space complexity of a *sampling* task and are more suitable for proving lower bounds on the space required to compute a function or a search problem[6].

From sampling to function computation. Our way around this is to first prove a reduction from streaming algorithms that sample a random walk from u_{start} to streaming algorithms that compute the $(p+1)$-neighborhood of the vertex u_{start}. This is done by considering a graph where a random walk returns to the vertex u_{start} every $p+2$ steps. If p is a constant, then a random walk of length L on such a graph can be seen as $L/(p+2) = O(L)$ copies of a random walk of length $p+2$. Observe that if the $(p+1)$-neighborhood of the vertex u_{start} has (almost) L vertices (and the probability of visiting each vertex is more or less uniform), then a random walk of length L is likely to visit all the vertices in the neighborhood and an algorithm that samples a random walk also outputs the entire neighborhood with high probability.

A lower bound for computing the $(p+1)$-neighborhood via pointer-chasing. Having reduced sampling a random walk to outputting the $(p+1)$-neighborhood, we now need to prove that a space efficient p-pass streaming algorithms cannot output the $(p+1)$-neighborhood of u_{start}, if this neighborhood has roughly L vertices. This is reminiscent of the "*pointer-chasing*" lower bounds found in the literature.

Pointer-chasing results are typically concerned with a graph with $p+1$ layers of vertices (p layers of edges) and show that given a vertex in the first layer, finding a vertex that is reachable from it in the last layer cannot be done with less than p passes, unless the memory is huge. Classical pointer-chasing lower bounds (*e.g.*, [31]), consider graphs where the out-degree of each vertex is 1, thus the start vertex reaches a unique vertex in the last layer. Unfortunately, this type of pointer-chasing instances are very *sparse* and a streaming algorithm can simply remember the entire graph in one pass using $\widetilde{O}(n)$ memory.

[6] One such tool that cannot be used directly for our purpose is the very useful *Yao's minimax principle* [38] that allows proving randomized communication lower bounds by proving the corresponding distributional (deterministic) communication lower bounds.

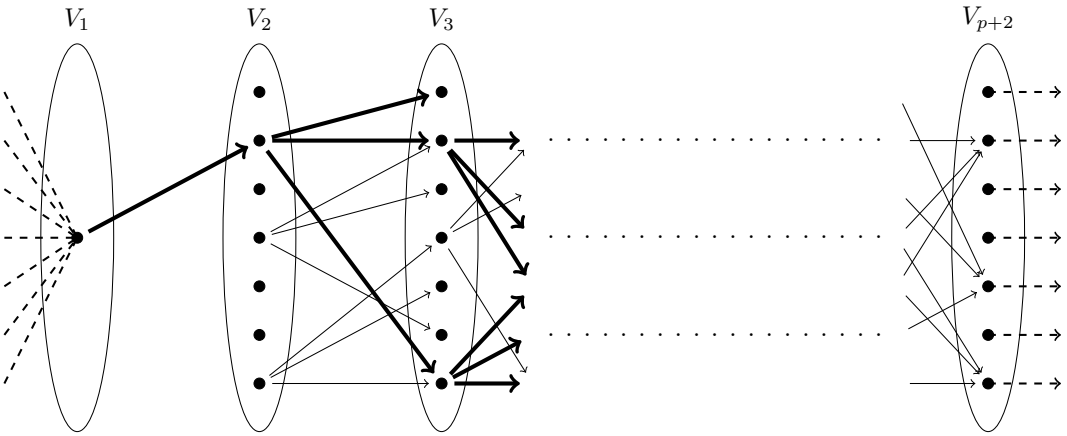

Figure 1 A depiction of our hard instance for p-pass streaming algorithms. Some edges omitted.

Since we wish to have roughly L vertices in a $(p+1)$-neighborhood of u_{start}, the out-degree of each vertex should be roughly $\Omega(L^{\frac{1}{p+1}})$ (assuming uniform degrees). Pointer-chasing lower bounds for this type of *dense* graphs were also proved (*e.g.*, [20] and [16]), showing that p-pass algorithms essentially need to store an entire layer of edges, which is $\Omega(n \cdot L^{\frac{1}{p+1}})$ in our case. However, this still does not give us the $\Omega(n \cdot L^{\frac{1}{p}})$ lower bound we aspire for (and which is tight, at least for two passes).

Towards a tight lower bound: combining dense and sparse. To get a better lower bound, we construct a hard instance that is a combination of the two above mentioned types of pointer-chasing instances, the dense and the sparse. Specifically, for a p-pass lower bound, we construct a layered graph with $p+2$ layers of vertices $V_1, \ldots, V_{p+2}$, where the first layer has only one vertex u_{start} and all the other layers are of equal size (see Figure 1). To ensure that vertex u_{start} is reached every $p+2$ steps, we connect all vertices in the last layer to u_{start}. Every vertex in layers $V_2, \ldots, V_{p+1}$ connects to a random set of roughly $L^{\frac{1}{p}}$ vertices in the next layer. Using Guruswami and Onak style arguments ([20]), it can be shown that when the edges are presented to the algorithm from right to left, finding a vertex in layer V_{p+2} that is reachable form a given vertex in V_2 with a $(p-1)$-pass algorithm requires $\Omega(n \cdot L^{\frac{1}{p}})$ space. We "squeeze out" an extra pass in the algorithm by connecting the start vertex u_{start} in V_1 to a single random vertex in V_2. Note that with this construction, it is indeed the case that a $(p+1)$-neighborhood of u_{start} consists of only roughly L vertices, but still, the out-degrees of vertices in $V_2, \ldots, V_{d+1}$ are roughly $L^{\frac{1}{p}}$ instead of only $L^{\frac{1}{p+1}}$.

2.3 Lower bounds for Oblivious Algorithms

Finally, we discuss the intuitions behind the proof of Theorem 3, showing that any algorithm that is oblivious to the starting vertex must use $\widetilde{\Omega}(n \cdot \sqrt{L})$ space space. Our proof is based on a reduction from a multi-output generalization of the well-studied INDEX problem for one-way communication protocols, denoted by $\mathsf{INDEX}_{m,\ell}$. In $\mathsf{INDEX}_{m,\ell}$, Alice gets ℓ strings $X_1, \ldots, X_\ell \in \{0,1\}^m$ and Bob gets an index $i \in [\ell]$. Alice sends a message to Bob and then Bob is required to output the string X_i. (Note that when $m = 1$ it becomes the original INDEX problem).

It is not hard to show that any one-way communication protocol solving $\mathsf{INDEX}_{m,\ell}$ with non-trivial probability (say, $1/\mathrm{poly}\log(m)$) requires Alice to send at least $\widetilde{\Omega}(m\ell)$ bits to Bob (see the full version of this paper [11] for details).

Our key observation here is that if there is a starting vertex oblivious algorithm $\mathbb{A} = (\mathbb{P}, \mathbb{S})$ with S space for approximate simulation of an $L = \widetilde{O}(m)$-step random walk on a graph with $n = O(\sqrt{m} \cdot \ell)$ vertices, then it implies a one-way communication protocol for $\mathsf{INDEX}_{m,\ell}$ with communication complexity S and a decent success probability. Recall the lower bound for $\mathsf{INDEX}_{m,\ell}$, we immediately have $S = \widetilde{\Omega}(m\ell) = \widetilde{\Omega}(n\sqrt{L})$.

In more detail, given an m-bit string X, we will build an $O(\sqrt{m})$-vertex graph $H(X)$ by encoding all bits of X as *existence/non-existence* of edges in H (this is possible since there are more than m potential edges in H). We also add some artificial edges to H to make sure it is strongly connected. Our construction will make sure that an $L = \widetilde{O}(m)$ steps random walk in H will reveal all edges in H with high probability, which in turn reveals all bits of X (see the proof of Theorem 3 in the full version of this paper for more details).

Now the reduction can be implemented as follows: given ℓ strings $X_1, \ldots, X_\ell \in \{0,1\}^m$, Alice constructs a graph $G = \bigsqcup_{i=1}^{\ell} H(X_i)$, as the joint union of ℓ graphs. Note that G has $n = O(\sqrt{m} \cdot \ell)$ vertices. Alice then runs the preprocessing algorithm $\mathbb{P}$ on G to obtain a string M, and sends it to Bob. Given an index $i \in [\ell]$, Bob simply runs $\mathbb{S}$ with M together with a suitable starting vertex inside the $H(X_i)$ component of G. By previous discussions, this reveals the string X_i with high probability and proves the correctness of this reduction. Hence, the space complexity of $\mathbb{A}$ must be $\widetilde{\Omega}(m\ell) = \widetilde{\Omega}(n\sqrt{L})$.

Organization of this paper

In Section 3 we introduce the necessary preliminaries for this paper. In Section 4 we present our nearly optimal two-pass streaming algorithm for simulating random walks and prove Theorem 1. See the full version of this paper [11] for the proof of Theorem 2 and Theorem 3.

3 Preliminaries

3.1 Notation

Let $n \in \mathbb{N}$. We use $[n]$ to denote the set $\{1, \ldots, n\}$. We often use sans-serif letters (*e.g.*, X) to denote random variables, and calligraphic font letters (*e.g.*, $\mathcal{X}$) to denote distributions. For two random variables X and Y, and for $Y \in \mathrm{supp}(\mathsf{Y})$, we use $(X|\mathsf{Y} = Y)$ to denote X conditioned on $\mathsf{Y} = Y$. For two lists a and b, we use $a \circ b$ to denote their concatenation.

For two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ on set $\mathcal{X}$ and $\mathcal{Y}$ respectively, we use $\mathcal{D}_1 \otimes \mathcal{D}_2$ to denote their product distribution over $\mathcal{X} \times \mathcal{Y}$, and $\|\mathcal{D}_1 - \mathcal{D}_2\|_{\mathrm{TV}}$ to denote the total variation distance between them.

3.2 Graphs

In this paper we will always consider directed graphs without multi-edges. A directed G is a pair (V, E), where V is the vertex set and $E \subseteq V \times V$ is the set of all edges.

For a vertex u in a graph $G = (V, E)$, we let $N^G_{\mathsf{out}}(u) \coloneqq \{v : (u, v) \in E\}$ and $N^G_{\mathsf{in}}(u) \coloneqq \{v : (v, u) \in E\}$. We also use $d^G_{\mathsf{out}}(u)$ and $d^G_{\mathsf{in}}(u)$ to denote its out and in degrees (*i.e.*, $|N^G_{\mathsf{out}}(u)|$ and $|N^G_{\mathsf{in}}(u)|$). For an edge $(u, v) \in E$, we say v is the *out-neighbor* of u and u is the *in-neighbor* of v.

Random walks on directed graphs. For a vertex u in a graph $G = (V, E)$ and an non-negative integer L, an L-step random walk $(v_0, v_1, \ldots, v_L)$ starting at u is generated as follows: set $v_0 = u$, for each $i \in [L]$, we draw v_i uniformly random from $N^G_{\mathsf{out}}(u)$. We say that $v_0 = u$ is the 0-th vertex on the walk, and v_i is the i-th vertex for each $i \in [L]$. We use $\mathsf{RW}^G_L(u)$ to denote the distribution of an L-step random walk starting from u in G.

We use $\mathsf{visit}^G_{[a,b]}(u,v)$ to denote the probability of a b-step random walk starting from u visits v between the a-th vertex and b-th vertex on the walk.

We often omit the superscript G when the graph G is clear from the context.

Starting vertex oblivious algorithms. Now we formally define a starting vertex oblivious streaming algorithm for simulating random walks.

▶ **Definition 4.** *We say a p-pass S-space streaming algorithm $\mathbb{A}$ for simulating random walks is starting vertex oblivious, if $\mathbb{A}$ can be decomposed into a preprocessing subroutine $\mathbb{P}$ and a sampling subroutine $\mathbb{S}$, such that:*

1. *(**Starting vertex oblivious preprocessing phase**) $\mathbb{P}$ makes p passes over the input graph stream, using at most S words of space. After that, $\mathbb{P}$ outputs at most S words, denoted as M.*
2. *(**Sampling phase**) $\mathbb{S}$ takes both the starting vertex u_{start} and M as input, and outputs a desired walk starting from u_{start}, using at most S words of space.*

3.3 Useful Concentration Bounds on Random Variables

The following standard concentration bounds will be useful for us.

▶ **Lemma 5** (Multiplicative Chernoff bound, [12])**.** *Suppose $X_1, \cdots, X_n$ are independent random variables taking values in $[0,1]$. Let X denote their sum and let $\mu = \mathbb{E}[X]$ denote the sum's expected value. Then,*

$$\Pr\left(X \geq (1+\delta)\mu\right) \leq \mathrm{e}^{-\frac{\delta^2\mu}{2+\delta}}, \qquad \forall 0 \leq \delta,$$

$$\Pr\left(X \leq (1-\delta)\mu\right) \leq \mathrm{e}^{-\frac{\delta^2\mu}{2}}, \qquad \forall 0 \leq \delta \leq 1.$$

In particular, we have that:

$$\Pr\left(X \geq (1+\delta)\mu\right) \leq \mathrm{e}^{-\frac{\delta\mu}{3}\cdot\min(\delta,1)}, \qquad \forall 0 \leq \delta,$$

$$\Pr\left(|X-\mu| \geq \delta\mu\right) \leq 2\cdot\mathrm{e}^{-\frac{\delta^2\mu}{3}}, \qquad \forall 0 \leq \delta \leq 1.$$

We also need the following Azuma-Hoeffding inequality.

▶ **Lemma 6** (Azuma-Hoeffding inequality, [5, 23])**.** *Let $Z_0, \ldots, Z_n$ be random variables satisfying (1) $\mathbb{E}[|Z_i|] < \infty$ for every $i \in \{0, \ldots, n\}$ and $\mathbb{E}[Z_i | Z_0, \ldots, Z_{i-1}] \leq Z_{i-1}$ for every $i \in [n]$ (i.e., $\{Z_i\}$ forms a supermartingale) and (2) for every $i \in [n]$, $|Z_i - Z_{i-1}| \leq 1$, then for all $\lambda > 0$, we have*

$$\Pr[Z_n - Z_0 \geq \lambda] \leq \exp(-\lambda^2/2n).$$

In particular, the following corollary will be useful for us.

▶ **Corollary 7** (Azuma-Hoeffding inequality for Boolean random variables, [5, 23])**.** *Let $X_1, \ldots, X_n$ be random variables satisfying $X_i \in \{0,1\}$ for each $i \in [n]$. Suppose that $\mathbb{E}[X_i | X_1, \ldots, X_{i-1}] \leq p_i$ for all i. Then for any $\lambda > 0$,*

$$\Pr\left[\sum_{i=1}^{n} X_i \geq \lambda + \sum_{i=1} p_i\right] \leq \exp(-\lambda^2/2n).$$

Proof. For $i \in \{0, \ldots, n\}$, let $Z_i = \sum_{j=1}^{i}(X_j - p_j)$. From the assumption one can see that all the Z_i form a supermartingale and $|Z_i - Z_{i-1}| \leq 1$, hence the corollary follows directly from Lemma 6. ◀

4 Two-Pass Streaming Algorithms for Simulating Directed Random Walk

In this section, we present our two-pass streaming algorithms for simulating random walks on directed graphs.

4.1 Heavy and Light Vertices

We first define the notion of heavy and light vertices.

▶ **Definition 8** (Heavy and light vertices). *Given a directed graph $G = (V, E)$ with n vertices and $\ell \in \mathbb{N}$.*

- *(**Heavy vertices**.) We say a vertex u is ℓ-heavy in G, if $\mathsf{visit}_{[1,\ell]}(u, u) \geq 1/3$ (i.e., if a random walk starting from u will revisit u in at most ℓ steps with probability at least $1/3$.)*
- *(**Light vertices**.) We say a vertex u is ℓ-light in G, if $\mathsf{visit}_{[1,\ell]}(u, u) \leq 2/3$ (i.e., if a random walk starting from u will revisit u in at most ℓ steps with probability at most $2/3$.)*

We also let $V^{\ell}_{\mathsf{heavy}}(G)$ and $V^{\ell}_{\mathsf{light}}(G)$ be the sets of ℓ-heavy and ℓ-light vertices in G. When G and ℓ are clear from the context, we simply refer to them as V_{heavy} and V_{light}.

▶ Remark 9. Note that if the revisiting probability is between $[1/3, 2/3]$, then the vertex is considered to be both heavy and light.

The following lemma is crucial for the analysis of our algorithm.

▶ **Lemma 10** (Upper bounds on the total out-degrees of heavy vertices). *Given a directed graph G with n vertices and $\ell \in \mathbb{N}$, it holds that*

$$\sum_{u \in V^{\ell}_{\mathsf{heavy}}(G)} d_{\mathsf{out}}(u) \leq O(n \cdot \ell).$$

Proof. We define a set $\mathcal{S}$ of pairs of vertices as follows:

$$\mathcal{S} \coloneqq \{(u, v) \in V^2 : \mathsf{visit}_{[0,\ell]}(u, v) \geq 1/6\}.$$

That is, a pair of vertices u and v belongs to $\mathcal{S}$ if and only if an ℓ-step random walk starting from u visits v with probability at least $1/6$.

For each fixed vertex u, we further define

$$\mathcal{S}_u \coloneqq \big\{v \in N_{\mathsf{out}}(u) \mid \mathsf{visit}_{[0,\ell]}(v, u) \geq 1/6\big\},$$

and

$$\mathcal{H}_u \coloneqq \big\{v \in V \mid \mathsf{visit}_{[0,\ell]}(u, v) \geq 1/6\big\}.$$

The following claim will be useful for the proof.

▷ Claim 11. The following two statements hold:

1. For every $u \in V$, it holds that $|\mathcal{H}_u| \leq O(\ell)$.
2. For every $u \in V_{\mathsf{heavy}}$, it holds that $|\mathcal{S}_u| \geq 1/6 \cdot d_{\mathsf{out}}(u)$.

Proof. Fixing $u \in V$, the first item follows from the simple fact that

$$\sum_{v \in V} \mathsf{visit}_{[0,\ell]}(u, v) \leq \ell + 1.$$

Now we move to the second item, and fix $u \in V_{\mathsf{heavy}}$. For the sake of contradiction, suppose that $|\mathcal{S}_u| < 1/6 \cdot d_{\mathsf{out}}(u)$. We have

$$\begin{aligned}\mathsf{visit}_{[1,\ell]}(u,u) &= \mathop{\mathbb{E}}_{v \in N_{\mathsf{out}}(u)}[\mathsf{visit}_{[0,\ell-1]}(v,u)] \\ &\leq \mathop{\mathbb{E}}_{v \in N_{\mathsf{out}}(u)}[\mathsf{visit}_{[0,\ell]}(v,u)] \\ &< \Pr_{v \in N_{\mathsf{out}}(u)}[v \in \mathcal{S}_u] \cdot 1 + \Pr_{v \in N_{\mathsf{out}}(u)}[v \notin \mathcal{S}_u] \cdot 1/6 < 1/6 + 1/6 < 1/3,\end{aligned}$$

a contradiction to the assumption that u is heavy. ◁

Finally, note that by definition of $\mathcal{H}_u$ and $\mathcal{S}_u$ we immediately have

$$|\mathcal{S}| = \sum_{u \in V} |\mathcal{H}_u| \geq \sum_{u \in V} |\mathcal{S}_u|.$$

By Claim 11, we have

$$\sum_{u \in V_{\mathsf{heavy}}} d_{\mathsf{out}}(u) \leq 6 \cdot \sum_{u \in V} |\mathcal{S}_u| \leq 6 \cdot \sum_{u \in V} |\mathcal{H}_u| \leq O\left(n \cdot \ell\right),$$

which completes the proof. ◀

4.2 A Simple One-Pass Algorithm for Simulating Random Walks

We first describe a simple one-pass algorithm for simulating random walks, which will be used as a sub-routine in our two-pass algorithm. Moreover, this one-pass algorithm is starting vertex oblivious, which will be crucial for us later.

Reservoir sampling in one pass. Before describing our one-pass subroutine, we need the following basic reservoir sampling algorithm.

▶ **Lemma 12** ([37]). *Given input access to a stream of n items such that each item can be described by $O(1)$ words, we can uniformly sample m of them without replacement using $O(m)$ words of space.*

Using m independent reservoir samplers each with capacity 1, one can also sample m items from the stream *with replacement* in a space-efficient way.

▶ **Corollary 13.** *Given input access to a stream of n items such that each item can be described by $O(1)$ words, we can uniformly sample m of them with replacement using $O(m)$ words of space.*

Description of the one-pass algorithm. Now we describe our one-pass algorithm for simulating random walks. Our algorithm $\mathbb{A}_{\mathsf{one\text{-}pass}}$ is starting vertex oblivious, and can be described by a preprocessing subroutine $\mathbb{P}_{\mathsf{one\text{-}pass}}$ and a sampling subroutine $\mathbb{S}_{\mathsf{one\text{-}pass}}$. Recall that as defined in Definition 4, $\mathbb{P}_{\mathsf{one\text{-}pass}}$ takes a single pass over the input graph streaming without knowing the starting vertex u_{start}, and $\mathbb{S}_{\mathsf{one\text{-}pass}}$ takes the output of $\mathbb{P}_{\mathsf{one\text{-}pass}}$ together with u_{start}, and outputs a desired sample fo the random walk.

Algorithm 1 Preprocessing phase of $\mathbb{A}_{\text{one-pass}}$: $\mathbb{P}_{\text{one-pass}}(G, \tau, V_{\text{full}})$.

Input: One pass streaming access to a directed graph $G = (V, E)$. A parameter $\tau \in \mathbb{N}$. A subset $V_{\text{full}} \subseteq V$, and we also let $V_{\text{samp}} = V \setminus V_{\text{full}}$.

1: For each vertex $v \in V_{\text{full}}$, we record all its out-neighbors in the list L_v^{save}. (That is, V_{full} stands for the set of vertices that we keep all its edges.)

2: For each vertex $v \in V_{\text{samp}}$, using Corollary 13, we sample τ of its out-neighbors uniformly at random with replacement in the list L_v^{save}. (That is, V_{samp} stands for the set of vertices that we sample some of its edges.)

3: For a big enough constant $c_2 > 1$, whenever the number of out-neighbors stored exceeds $c_2 \cdot \tau \cdot n$, the algorithm stops recording them. If this happens, we say the algorithm *operates incorrectly* and otherwise we say it *operates correctly*.

Output: A collection of lists $\vec{L}^{\text{save}} = \{L_v^{\text{save}}\}_{v \in V}$.

Algorithm 2 Sampling phase of $\mathbb{A}_{\text{one-pass}}$: $\mathbb{S}_{\text{one-pass}}(V, u_{\text{start}}, L, V_{\text{full}}, \vec{L}^{\text{save}} = \{L_v^{\text{save}}\}_{v \in V})$.

Input: A starting vertex u_{start}. The path length $L \in \mathbb{N}$. A subset $V_{\text{full}} \subseteq V$, and we also let $V_{\text{samp}} = V \setminus V_{\text{full}}$.

1: Let $v_0 = u_{\text{start}}$. For each $v \in V$, we set $k_v = 1$.
2: **for** $i \coloneqq 1 \to L$ **do**
3: **if** $v_{i-1} \in V_{\text{full}}$ **then**
4: v_i is set to be a uniformly random element from $L_{v_{i-1}}^{\text{save}}$
5: **else if** $k_{v_{i-1}} > |L_{v_{i-1}}^{\text{save}}|$ **then**
6: **return** failure
7: **else**
8: $v_i \leftarrow (L_{v_{i-1}}^{\text{save}})_{k_{v_{i-1}}}$.
9: $k_{v_{i-1}} \leftarrow k_{v_{i-1}} + 1$.
10: **end if**
11: **end for**

Output: The walk $(v_0, v_1, \ldots, v_L)$.

Analysis of the one-pass algorithm. Now we analyze the correctness of our one-pass algorithm. We first observe its space complexity can be easily bounded.

▶ **Observation 14** (Space complexity of $\mathbb{A}_{\text{one-pass}}$). *Given a directed graph $G = (V, E)$ with n vertices. For every $\tau \in \mathbb{N}$ and subset $V_{\text{full}} \subseteq V$, $\mathbb{P}_{\text{one-pass}}(G, \tau, V_{\text{full}})$ always takes at most $O(\tau \cdot n)$ words of space.*

Next we bound the statistical distance between its output distribution and the correct distribution of the random walk by the following lemma.

▶ **Lemma 15** (Correctness of $\mathbb{A}_{\text{one-pass}}$). *Given a directed graph $G = (V, E)$ with n vertices. For every integers $\tau, L \in \mathbb{N}$ and subset $V_{\text{full}} \subseteq V$ such that $\tau \cdot (n - |V_{\text{full}}|) + \sum_{v \in V_{\text{full}}} d_{\text{out}}(v) \leq c_2 \cdot \tau \cdot n$, let $\vec{\mathsf{L}}^{\text{save}}$ be random variable of the output of $\mathbb{P}_{\text{one-pass}}(G, \tau, V_{\text{full}})$. For every $u_{\text{start}} \in V$, the output distribution of $\mathbb{S}_{\text{one-pass}}(V, u_{\text{start}}, L, V_{\text{full}}, \vec{\mathsf{L}}^{\text{save}})$ has statistical distance β to $\mathsf{RW}_L^G(u_{\text{start}})$, where β is the probability that $\mathbb{S}_{\text{one-pass}}(V, u_{\text{start}}, L, V_{\text{full}}, \vec{\mathsf{L}}^{\text{save}})$ outputs failure.*

Proof. Conclude from $\tau \cdot (n - |V_{\mathsf{full}}|) + \sum_{v \in V_{\mathsf{full}}} d_{\mathsf{out}}(v) \leq c_2 \cdot \tau \cdot n$ that $\mathbb{P}_{\mathsf{one\text{-}pass}}(G, \tau, V_{\mathsf{full}})$ always operates correctly.

To bound the statistical distance between the distribution of $\mathbb{S}_{\mathsf{one\text{-}pass}}(V, u_{\mathsf{start}}, L, V_{\mathsf{full}}, \vec{\mathsf{L}}^{\mathsf{save}})$ and $\mathsf{RW}_L^G(u_{\mathsf{start}})$. We construct another random variable $(\vec{\mathsf{L}}^{\mathsf{save}})'$, in which for every vertex u, we sample another L out-neighbors of u uniformly at random with replacement, and add them to the end of the list L_v^{save} in $\vec{\mathsf{L}}^{\mathsf{save}}$.

Note that $\mathbb{S}_{\mathsf{one\text{-}pass}}(V, u_{\mathsf{start}}, L, V_{\mathsf{full}}, (\vec{\mathsf{L}}^{\mathsf{save}})')$ never outputs failure, and distributes exactly the same as $\mathsf{RW}_L^G(u_{\mathsf{start}})$. On the other hand, $\mathbb{S}_{\mathsf{one\text{-}pass}}(V, u_{\mathsf{start}}, L, V_{\mathsf{full}}, (\vec{\mathsf{L}}^{\mathsf{save}})')$ and $\mathbb{S}_{\mathsf{one\text{-}pass}}(V, u_{\mathsf{start}}, L, V_{\mathsf{full}}, \vec{\mathsf{L}}^{\mathsf{save}})$ are the same as long as $\mathbb{S}_{\mathsf{one\text{-}pass}}(V, u_{\mathsf{start}}, L, V_{\mathsf{full}}, \vec{\mathsf{L}}^{\mathsf{save}})$ does not output failure, which completes the proof. ◀

The following corollary follows immediately from the lemma above. (Note that this special case exactly corresponds to the folklore one-pass streaming algorithm for simulating random walks.)

▶ **Corollary 16.** *Given a directed graph $G = (V, E)$ with n vertices and an integer $L \in \mathbb{N}$. Let $\vec{\mathsf{L}}^{\mathsf{save}}$ be random variable of the output of $\mathbb{P}_{\mathsf{one\text{-}pass}}(G, L, \emptyset)$. For every $u_{\mathsf{start}} \in V$, the output distribution of $\mathbb{S}_{\mathsf{one\text{-}pass}}(V, u_{\mathsf{start}}, L, \emptyset, \vec{\mathsf{L}}^{\mathsf{save}})$ distributes identically as $\mathsf{RW}_L^G(u_{\mathsf{start}})$.*

4.3 Two-Pass Streaming Algorithm for Simulating Random Walks

Description of the two-pass algorithm. Now we are ready to describe our two pass algorithm $\mathbb{A}_{\mathsf{two\text{-}pass}}$, which is also starting vertex oblivious, and can be described by the following two sub-routines $\mathbb{P}_{\mathsf{two\text{-}pass}}$ and $\mathbb{S}_{\mathsf{two\text{-}pass}}$.

Algorithm 3 Preprocessing phase of $\mathbb{A}_{\mathsf{two\text{-}pass}}$: $\mathbb{P}_{\mathsf{two\text{-}pass}}(G, L, \delta)$.

Input: A directed graph $G = (V, E)$ with n vertices. An integer $L \in \mathbb{N}$. A failure parameter $\delta \in (0, 1/n)$. We also let $\ell = \sqrt{L}$, and $\gamma = c_1 \cdot \log \delta^{-1}$ where $c_1 \geq 1$ is a sufficiently large constant to be specified later.

1: **First pass: estimation of heavy and light vertices.**

1. Run γ independent instances of $\mathbb{P}_{\mathsf{one\text{-}pass}}(G, \ell, \emptyset)$ and let $(L^{\mathsf{save}})^{(1)}, \ldots, (L^{\mathsf{save}})^{(\gamma)}$ be the corresponding collections of lists.
2. For each vertex $u \in V$, by running $\mathbb{S}_{\mathsf{one\text{-}pass}}(V, u, \ell, \emptyset, (L^{\mathsf{save}})^{(j)})$ for each $j \in [\gamma]$, we take γ independent samples from RW_ℓ^G. Let w_u be the fraction of these random walks that revisit u in ℓ steps.
3. Let $\widetilde{V}_{\mathsf{heavy}}$ be the set of vertices with $w_u \geq 0.5$, and $\widetilde{V}_{\mathsf{light}}$ be the set of vertices with $w_u < 0.5$.

2: **Second Pass: heavy-light edge recording**

1. Let $V_{\mathsf{full}} = \widetilde{V}_{\mathsf{heavy}}$.
2. Run $\mathbb{P}_{\mathsf{one\text{-}pass}}(G, \gamma \cdot \ell, V_{\mathsf{full}})$ to obtain a collection of lists $\vec{L}^{\mathsf{save}}$.

Output: The set V_{full} and the collection of lists $\vec{L}^{\mathsf{save}}$.

Algorithm 4 Sampling phase of $\mathbb{A}_{\mathsf{two\text{-}pass}}$: $\mathbb{S}_{\mathsf{two\text{-}pass}}(V, u_{\mathsf{start}}, L, V_{\mathsf{full}}, \vec{L}^{\mathsf{save}} = \{L_v^{\mathsf{save}}\}_{v \in V})$.

Input: A starting vertex u_{start}. The path length $L \in \mathbb{N}$. A subset $V_{\mathsf{full}} \subseteq V$, and a collection of lists $\vec{L}^{\mathsf{save}}$.

Output: Simulate $\mathbb{S}_{\mathsf{one\text{-}pass}}(V, u_{\mathsf{start}}, L, V_{\mathsf{full}}, \vec{L}^{\mathsf{save}})$ and return its output.

Analysis of the algorithm. We first show that with high probability, $\widetilde{V}_{\mathsf{light}}$ and $\widetilde{V}_{\mathsf{heavy}}$ are subsets of V_{light} and V_{heavy} respectively.

▶ **Lemma 17.** *Given a directed graph $G = (V, E)$ with n vertices, $L \in \mathbb{N}$ and $\delta \in (0, 1/n)$, letting $\ell = \sqrt{L}$, with probability at least $1-\delta/2$ over the internal randomness of $\mathbb{P}_{\mathsf{two\text{-}pass}}(G, L, \delta)$, it holds that $\widetilde{V}_{\mathsf{light}} \subseteq V_{\mathsf{light}}$ and $\widetilde{V}_{\mathsf{heavy}} \subseteq V_{\mathsf{heavy}}$.*

Proof. Setting c_1 in Algorithm 3 to be a large enough constant and applying Corollary 16 and the Chernoff bound, with probability at least $1 - n \cdot \delta^3 \geq 1 - \delta/2$, $|w_u - \mathsf{visit}_{[1,\ell]}(u, u)| \leq 0.1$ for every $u \in V$. The lemma then follows from the definition of heavy and light vertices. ◀

Next, we show that with high probability, a random walk does not visit a light vertex too many times.

▶ **Lemma 18.** *Given a directed graph $G = (V, E)$ with n vertices, $L \in \mathbb{N}$ and $\delta \in (0, 1/n)$, letting $\ell = \sqrt{L}$ and $\gamma = c_1 \cdot \log \delta^{-1}$, where $c_1 > 1$ is the sufficiently large constant, for every vertex $u_{\mathsf{start}} \in V$ and vertex $v \in V^{\ell}_{\mathsf{light}}(G)$, an L-step random walk starting from u_{start} visits v more than $\gamma \cdot \ell$ times with probability at most $\delta/2n$.*

Proof. Suppose we have an infinite random walk W starting from u_{start} in G. Letting $\tau = \gamma\ell$, the goal here is to bound the probability that during the first L steps, W visits v more than τ times. We denote this as the bad event $\mathcal{E}_{\mathsf{bad}}$.

Let Z_i be the random variable representing the step at which W visits v for the i-th time (if W visits v less than i times in total, we let $\mathsf{Z}_i = \infty$). $\mathcal{E}_{\mathsf{bad}}$ is equivalent to that $\mathsf{Z}_{\tau+1} \leq L$.

$\mathsf{Z}_{\tau+1} \leq L$ further implies that for at least $(\tau - \ell)$ $i \in [\tau]$, $\mathsf{Z}_{i+1} - \mathsf{Z}_i \leq \ell$ and $\mathsf{Z}_i < \infty$. In the following we denote this event as $\mathcal{E}_1$ and bounds its probability instead.

For each $i \in [\tau]$, let Y_i be the random variable which takes value 1 if both $Z_i < \infty$ and $\mathsf{Z}_{i+1} - \mathsf{Z}_i \leq \ell$ hold, and 0 otherwise. Letting $\mathsf{Y}_{<i} = (\mathsf{Y}_1, \ldots, \mathsf{Y}_{i-1})$, the following claim is crucial for us.

▷ Claim 19. For every $i \in [\tau]$ and every possible assignments $Y_{<i} \in \{0, 1\}^{i-1}$, we have

$$\mathbb{E}[\mathsf{Y}_i | \mathsf{Y}_{<i} = Y_{<i}] \leq 2/3.$$

Proof. By the Markov property of the random walk, and noting that Y_i is always 0 when $\mathsf{Z}_i = \infty$, we have.

$$\begin{aligned}\mathbb{E}[\mathsf{Y}_i | \mathsf{Y}_{<i} = Y_{<i}] &= \sum_{j=0}^{\infty} \Pr[\mathsf{Z}_i = j | \mathsf{Y}_{<i} = Y_{<i}] \cdot \mathbb{E}[\mathsf{Y}_i | \mathsf{Y}_{<i} = Y_{<i}, \mathsf{Z}_i = j] \\ &= \sum_{j=0}^{\infty} \Pr[\mathsf{Z}_i = j | \mathsf{Y}_{<i} = Y_{<i}] \cdot \mathbb{E}[\mathsf{Y}_i | \mathsf{Z}_i = j].\end{aligned}$$

To further bound the quantity above, recall that the event $\mathsf{Z}_i = j$ means that the random walk W starting from u_{start} visits the light vertex v for the i-th time at W's j-th step, and we have

$$\mathbb{E}[\mathsf{Y}_i | \mathsf{Z}_i = j] = \Pr[\mathsf{Y}_i = 1 | \mathsf{Z}_i = j] = \Pr[\mathsf{Z}_{i+1} \leq j + \ell | \mathsf{Z}_i = j].$$

By the Markov property of the random walk W, $\Pr[\mathsf{Z}_{i+1} \leq j + \ell | \mathsf{Z}_i = j]$ equals the probability that a random walk starting from v revisits v in at most ℓ steps. By the definition of light vertices, we can bound that by $2/3$, which completes the proof. ◁

Then by the Azuma-Hoeffding inequality (Corollary 7),

$$\begin{aligned}\Pr_{\mathsf{W}}[\mathcal{E}_{\mathsf{bad}}] &\leq \Pr_{\mathsf{W}}[\mathcal{E}_1] \\ &= \Pr_{\mathsf{W}}\left[\sum_{i=1}^{\tau} \mathsf{Y}_i \geq (\tau - \ell)\right] \\ &\leq \exp(-\Omega(\tau - \ell - 2/3 \cdot \tau)) \leq \delta/2n,\end{aligned}$$

the last inequality follows from the fact that $\Omega(\tau - \ell - 2/3 \cdot \tau) = \Omega(\gamma)$, $\gamma = c_1 \cdot \log \delta^{-1}$ for a sufficiently large constant c_1, and $\delta \leq 1/n$. ◀

The correctness of the algorithm is finally completed by the following theorem.

▶ **Theorem 20** (Formal version of Theorem 1). *Given a directed graph $G = (V, E)$ with n vertices, $L \in \mathbb{N}$ and $\delta \in (0, 1/n)$. Let $\vec{\mathsf{L}}^{\mathsf{save}}$ and $\mathsf{V}_{\mathsf{full}}$ be the two random variables of the output of $\mathbb{P}_{\mathsf{two\text{-}pass}}(G, L, \delta)$. For every $u_{\mathsf{start}} \in V$, the following hold:*

- *The output distribution of $\mathbb{S}_{\mathsf{two\text{-}pass}}(V, u_{\mathsf{start}}, L, \mathsf{V}_{\mathsf{full}}, \vec{\mathsf{L}}^{\mathsf{save}})$ has statistical distance at most δ from $\mathsf{RW}_L^G(u_{\mathsf{start}})$.*
- *Both of $\mathbb{P}_{\mathsf{two\text{-}pass}}(G, L, \delta)$ and $\mathbb{S}_{\mathsf{two\text{-}pass}}(V, u_{\mathsf{start}}, L, \mathsf{V}_{\mathsf{full}}, \vec{\mathsf{L}}^{\mathsf{save}})$ use at most $O(n \cdot \sqrt{L} \cdot \log \delta^{-1})$ words of space.*

Proof. Note that we can safely assume $L \leq n^2$, since otherwise one can always use $O(n^2)$ words to store all the edges in the graph. In this case, we have that $L \leq n \cdot \sqrt{L}$ and the space for restoring the L-step output walk can be ignored.

Let $\widetilde{\mathsf{V}}_{\mathsf{heavy}} = \mathsf{V}_{\mathsf{full}}$ and $\widetilde{\mathsf{V}}_{\mathsf{light}} = V \setminus \widetilde{\mathsf{V}}_{\mathsf{heavy}}$. Let $\mathcal{E}_{\mathsf{good}}$ be the event that $\widetilde{\mathsf{V}}_{\mathsf{light}} \subseteq V_{\mathsf{light}}$ and $\widetilde{\mathsf{V}}_{\mathsf{heavy}} \subseteq V_{\mathsf{heavy}}$. By Lemma 17, we have that $\Pr[\mathcal{E}_{\mathsf{good}}] \geq 1 - \delta/2$.

Now we condition on the event $\mathcal{E}_{\mathsf{good}}$. In this case, it follows from Lemma 10 that $\mathbb{P}_{\mathsf{one\text{-}pass}}(G, \gamma \cdot \ell, \widetilde{\mathsf{V}}_{\mathsf{heavy}})$ operates correctly (by setting the constant c_2 in Algorithm 1 to be sufficiently large).

By Lemma 18 and a union bound, the probability of $\mathbb{S}_{\mathsf{two\text{-}pass}}(V, u_{\mathsf{start}}, L, \widetilde{\mathsf{V}}_{\mathsf{heavy}}, \vec{\mathsf{L}}^{\mathsf{save}})$ outputs failure is at most $\delta/2$. By Lemma 15, it follows that the statistical distance between the output distribution of $\mathbb{S}_{\mathsf{two\text{-}pass}}(V, u_{\mathsf{start}}, L, \widetilde{\mathsf{V}}_{\mathsf{heavy}}, \vec{\mathsf{L}}^{\mathsf{save}})$ and $\mathsf{RW}_L^G(u_{\mathsf{start}})$ is at most $\delta/2$.

The theorem follows by combing the above with the fact that $\Pr[\mathcal{E}_{\mathsf{good}}] \geq 1 - \delta/2$. ◀

4.4 Two-pass Streaming in the Turnstile Model

Similar to the algorithm in [27], our algorithms can also be easily adapted to work for the *turnstile graph streaming model*, where both insertions and deletions of edges are allowed. Note that our two-pass algorithm $\mathbb{A}_{\mathsf{two\text{-}pass}}$ only accesses the input graph stream via the one-pass preprocessing subroutine $\mathbb{P}_{\mathsf{one\text{-}pass}}$. Hence, it suffices to implement $\mathbb{P}_{\mathsf{one\text{-}pass}}$ in the turnstile model as well. There are two distinct tasks in $\mathbb{P}_{\mathsf{one\text{-}pass}}$: (1) for light vertices, we need to sample their outgoing neighbors with replacement and (2) for heavy vertices, we need to record all their outgoing neighbors.

Uniformly sampling via ℓ_1 sampler. For light vertices, uniformly sampling some out-neighbors from each vertex without replacement can be implemented via the following ℓ_1 sampler in the turnstile model.

▶ **Lemma 21** (ℓ_1 sampler in the turnstile model [24]). *Let $n \in \mathbb{N}$, failure probability $\delta \in (0, 1/2)$ and $f \in \mathbb{R}^n$ be a vector defined by a streaming of updates to its coordinates of the form $f_i \leftarrow f_i + \Delta$, where $\Delta \in \{-1, 1\}$. There is a randomized algorithm which reads the stream, and with probability at most δ it outputs FAIL, otherwise it outputs an index $i \in [n]$ such that:*

$$\Pr(i = j) = \frac{|f_j|}{\|f\|_1} + O(n^{-c}), \qquad \forall j \in [n]$$

where $c \geq 1$ is some arbitrarily large constant.

The space complexity of this algorithm is bounded by $O(\log^2(n) \cdot \log(1/\delta))$ bits in the random oracle model, and $O(\log^2(n) \cdot (\log\log n)^2 \cdot \log(1/\delta))$ bits otherwise.

▶ Remark 22. To get error in the statistical distance also to be at most δ, one can simply set n to be larger than $1/\delta$. And in that case the space complexity can be bounded by $O(\log^4(n/\delta))$.

Recording all outgoing neighbors via ℓ_1 heavy hitter. For heavy vertices, recording all their outgoing neighbors can be implemented using the following ℓ_1 heavy hitter in the turnstile model. (Recall that we assumed our graphs is a simple graph without multiple edges.)

▶ **Lemma 23** (ℓ_1 heavy hitter in the turnstile model [9]). *Let $n, k \in \mathbb{N}$, $\delta \in (0, 0.1)$ and $f \in \mathbb{R}^n$ be a vector defined by a streaming of updates to its coordinates of the form $f_i \leftarrow f_i + \Delta$, where $\Delta \in \{-1, 1\}$. There is an algorithm which reads the stream and returns a subset $L \subset [n]$ such that $i \in L$ for every $i \in [n]$ such that $|f_i| \geq \|f\|_1/k$ and $i \notin L$ for every $i \in [n]$ such that $|f_i| \leq \|f\|_1/2k$. The failure probability is at most δ, and the space complexity is at most $O(k \cdot \log(n) \cdot \log(n/\delta))$.*

Algorithm in the turnstile model. Modifying $\mathbb{P}_{\text{one-pass}}$ with Lemma 21 and Lemma 23, we can generalize our two-pass streaming algorithm to work in two-pass turnstile model.[7]

▶ Remark 24 (Two-pass algorithm in the turnstile model). There exists a streaming algorithm $\mathbb{A}_{\text{turnstile}}$ that given an n-vertex *directed* graph $G = (V, E)$ via a stream of both edge insertions and edge deletions, a starting vertex $u_{\text{start}} \in V$, a non-negative integer L indicating the number of steps to be taken, and an error parameter $\delta \in (0, 1/n)$, satisfies the following conditions:

1. $\mathbb{A}_{\text{turnstile}}$ uses at most $\widetilde{O}(n \cdot \sqrt{L} \cdot \log \delta^{-1})$ space and makes two passes over the input graph G.
2. $\mathbb{A}_{\text{turnstile}}$ samples from some distribution $\mathcal{D}$ over V^{L+1} satisfying $\|\mathcal{D} - \mathsf{RW}_L^G(u_{\text{start}})\|_{\text{TV}} \leq \delta$.

References

1 Reid Andersen, Fan Chung, and Kevin Lang. Using pagerank to locally partition a graph. *Internet Mathematics*, 4(1):35–64, 2007.

2 Reid Andersen and Yuval Peres. Finding sparse cuts locally using evolving sets. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 235–244, 2009.

[7] In more details, for each light vertex u, we run τ independent copies of the ℓ_1 sampler to obtain τ samples from its outgoing neighbors with replacement. We also let $k = c_2 \cdot \tau \cdot n$ and use the ℓ_1 heavy hitter to record all outgoing neighbors for all heavy vertices in $\widetilde{O}(n \cdot \sqrt{L} \cdot \log(1/\delta))$ space.

3 Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear algorithms for $(\Delta + 1)$ vertex coloring. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 767–786. SIAM, 2019.

4 Sepehr Assadi, Sanjeev Khanna, and Yang Li. Tight bounds for single-pass streaming complexity of the set cover problem. In *48th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 698–711. Association for Computing Machinery, 2016.

5 Kazuoki Azuma. Weighted sums of certain dependent random variables. *Tohoku Mathematical Journal, Second Series*, 19(3):357–367, 1967.

6 Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

7 Amit Chakrabarti, Prantar Ghosh, Andrew McGregor, and Sofya Vorotnikova. Vertex ordering problems in directed graph streams. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1786–1802. SIAM, 2020.

8 Amit Chakrabarti and Anthony Wirth. Incidence geometries and the pass complexity of semi-streaming set cover. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 1365–1373. SIAM, 2016.

9 Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 693–703. Springer, 2002.

10 Moses Charikar, Liadan O'Callaghan, and Rina Panigrahy. Better streaming algorithms for clustering problems. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 30–39, 2003.

11 Lijie Chen, Gillat Kol, Dmitry Paramonov, Raghuvansh Saxena, Zhao Song, and Huacheng Yu. Near-optimal two-pass streaming algorithm for sampling random walks over directed graphs. *CoRR*, abs/2102.11251, 2021. `arXiv:2102.11251`.

12 Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, pages 493–507, 1952.

13 Graham Cormode, Jacques Dark, and Christian Konrad. Independent sets in vertex-arrival streams. In *46th International Colloquium on Automata, Languages, and Programming (ICALP)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

14 Yuval Emek and Adi Rosén. Semi-streaming set cover. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 453–464. Springer, 2014.

15 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 531–543. Springer, 2004.

16 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the data-stream model. *SIAM Journal on Computing*, 38(5):1709–1727, 2009.

17 Buddhima Gamlath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. Weighted matchings via unweighted augmentations. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 491–500, 2019.

18 Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrović, and Ronitt Rubinfeld. Improved massively parallel computation algorithms for mis, matching, and vertex cover. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 129–138, 2018.

19 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms (SODA)*, pages 468–485. SIAM, 2012.

20 Venkatesan Guruswami and Krzysztof Onak. Superlinear lower bounds for multipass graph processing. *Algorithmica*, 76(3):654–683, 2016.

21 Sariel Har-Peled, Piotr Indyk, Sepideh Mahabadi, and Ali Vakilian. Towards tight bounds for the streaming set cover problem. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*, pages 371–383, 2016.

22 Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. *External memory algorithms*, 50:107–118, 1998.

23 Wassily Hoeffding. Probability inequalities for sums of bounded random variables. In *The Collected Works of Wassily Hoeffding*, pages 409–426. Springer, 1994.

24 Rajesh Jayaram and David P. Woodruff. Perfect lp sampling in a data stream. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 544–555. IEEE Computer Society, 2018.

25 Mark Jerrum and Alistair Sinclair. Approximating the permanent. *SIAM journal on computing*, 18(6):1149–1178, 1989.

26 Mark R Jerrum, Leslie G Valiant, and Vijay V Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical computer science*, 43:169–188, 1986.

27 Ce Jin. Simulating random walks on graphs in the streaming model. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPIcs*, pages 46:1–46:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ITCS.2019.46.

28 Michael Kapralov. Better bounds for matchings in the streaming model. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 1679–1697. SIAM, 2013.

29 Andrew McGregor. Finding graph matchings in data streams. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 170–181. Springer, 2005.

30 Sagnik Mukhopadhyay and Danupon Nanongkai. Weighted min-cut: sequential, cut-query, and streaming algorithms. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 496–509, 2020.

31 Noam Nisan and Avi Wigderson. Rounds in communication complexity revisited. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 419–429. ACM, 1991.

32 Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):1–24, 2008.

33 Aviad Rubinstein, Tselil Schramm, and Seth Matthew Weinberg. Computing exact minimum cuts without knowing the graph. In *9th Innovations in Theoretical Computer Science (ITCS)*, page 39. Schloss Dagstuhl-Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 2018.

34 Atish Das Sarma, Sreenivas Gollapudi, and Rina Panigrahy. Estimating pagerank on graph streams. *J. ACM*, 58(3):13:1–13:19, 2011. doi:10.1145/1970392.1970397.

35 Aaron Schild. An almost-linear time algorithm for uniform random spanning tree generation. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 214–227, 2018.

36 Daniel A Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM Journal on computing*, 42(1):1–26, 2013.

37 Jeffrey Scott Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985. doi:10.1145/3147.3165.

38 Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 222–227. IEEE Computer Society, 1977.

39 Mariano Zelke. Intractability of min-and max-cut in streaming graphs. *Information Processing Letters*, 111(3):145–150, 2011.