Simulating Random Walks on Graphs in the Streaming Model

Ce Jin
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
jinc16@mails.tsinghua.edu.cn

Abstract

We study the problem of approximately simulating a t-step random walk on a graph where the input edges come from a single-pass stream. The straightforward algorithm using reservoir sampling needs $O(nt)$ words of memory. We show that this space complexity is near-optimal for directed graphs. For undirected graphs, we prove an $\Omega(n\sqrt{t})$-bit space lower bound, and give a near-optimal algorithm using $O(n\sqrt{t})$ words of space with $2^{-\Omega(\sqrt{t})}$ simulation error (defined as the ℓ_1-distance between the output distribution of the simulation algorithm and the distribution of perfect random walks). We also discuss extending the algorithms to the turnstile model, where both insertion and deletion of edges can appear in the input stream.

2012 ACM Subject Classification Theory of computation → Streaming models

Keywords and phrases streaming models, random walks, sampling

Digital Object Identifier 10.4230/LIPIcs.ITCS.2019.46

Acknowledgements I would like to thank Professor Jelani Nelson for introducing this problem to me, advising this project, and giving many helpful comments on my writeup.

1 Introduction

Graphs of massive size are used for modeling complex systems that emerge in many different fields of study. Challenges arise when computing with massive graphs under memory constraints. In recent years, graph streaming has become an important model for computation on massive graphs. Many space-efficient streaming algorithms have been designed for solving classical graph problems, including connectivity [2], bipartiteness [2], minimum spanning tree [2], matching [8, 12, 1], spectral sparsifiers [14, 13], etc. We will define the streaming model in Section 1.1.

Random walks on graphs are stochastic processes that have many applications, such as connectivity testing [17], clustering [18, 3, 4, 5], sampling [11] and approximate counting [10]. Since random walks are a powerful tool in algorithm design, it is interesting to study them in the streaming setting. A natural problem is to find the space complexity of simulating random walks in graph streams. Das Sarma et al. [7] gave a multi-pass streaming algorithm that simulates a t-step random walk on a directed graph using $O(\sqrt{t})$ passes and only $O(n)$ space. By further extending this algorithm and combining with other ideas, they obtained space-efficient algorithms for estimating PageRank on graph streams. However, their techniques crucially rely on reading multiple passes of the input stream.

In this paper, we study the problem of simulating random walks in the *one-pass* streaming model. We show space lower bounds for both directed and undirected versions of the problem, and present algorithms that nearly match with the lower bounds. We summarize our results in Section 1.3.

10th Innovations in Theoretical Computer Science (ITCS 2019).
Editor: Avrim Blum; Article No. 46; pp. 46:1–46:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1.1 One-pass streaming model

Let $G = (V, E)$ be a graph with n vertices. In the insertion-only model, the input graph G is defined by a stream of edges $(e_1, \dots, e_m)$ seen in arbitrary order, where each edge e_i is specified by its two endpoints $u_i, v_i \in V$. An algorithm must process the edges of G in the order that they appear in the input stream. The edges can be directed or undirected, depending on the problem setting. Sometimes we allow multiple edges in the graph, where the multiplicity of an edge equals its number of occurrences in the input stream.

In the turnstile model, we allow both insertion and deletion of edges. The input is a stream of updates $((e_1, \Delta_1), (e_2, \Delta_2), \dots)$, where e_i encodes an edge and $\Delta_i \in \{1, -1\}$. The multiplicity of edge e is $f(e) = \sum_{e_i = e} \Delta_i$. We assume $f(e) \geq 0$ always holds for every edge e.

1.2 Random walks

Let $f(u, v)$ denote the multiplicity of edge (u, v). The degree of u is defined by $d(u) = \sum_{v \in V} f(u, v)$. A t-step random walk starting from a vertex $s \in V$ is a random sequence of vertices $v_0, v_1, \dots, v_t$ where $v_0 = s$ and v_i is a vertex uniformly randomly chosen from the vertices that v_{i-1} connects to, i.e., $\mathbb{P}[v_i = v | v_{i-1} = u] = f(u, v)/d(u)$. Let $\mathcal{RW}_{s,t} : V^{t+1} \to [0, 1]$ denote the distribution of t-step random walks starting from s, defined by[1]

$$\mathcal{RW}_{s,t}(v_0, \dots, v_t) = \mathbf{1}[v_0 = s] \prod_{i=0}^{t-1} \frac{f(v_i, v_{i+1})}{d(v_i)}. \tag{1}$$

For two distributions P, Q, we denote by $|P - Q|_1$ their ℓ_1 distance. We say that a randomized algorithm can simulate a t-step random walk starting from v_0 within error ε, if the distribution $\mathbb{P}_w$ of its output $w \in V^{t+1}$ satisfies $|\mathbb{P}_w - \mathcal{RW}_{v_0,t}|_1 \leq \varepsilon$. We say the random walk simulation is perfect if $\varepsilon = 0$.

We study the problem of simulating a t-step random walk within error ε in the streaming model using small space. We assume the length t is specified at the beginning. Then the algorithm reads the input stream. When a query with parameter v_0 comes, the algorithm should simulate and output a t-step random walk starting from vertex v_0.

It is without loss of generality to assume that the input graph has no self-loops. If we can simulate a random walk on the graph with self-loops removed, we can then turn it into a random walk of the original graph by simply inserting self-loops after u with probability $d_{\text{self}}(u)/d(u)$. The values $d_{\text{self}}(u), d(u)$ can be easily maintained by a streaming algorithm using $O(n)$ words.

The random walk is not well-defined when it starts from a vertex u with $d(u) = 0$. For undirected graphs, this can only happen at the beginning of the random walk, and we simply let our algorithm return FAIL if $d(v_0) = 0$. For directed graphs, one way to fix this is to continue the random walk from v_0, by adding an edge (u, v_0) for every vertex u with $d(u) = 0$. We will not deal with $d(u) = 0$ in the following discussion.

1.3 Our results

We will use $\log x = \log_2 x$ throughout this paper.

The following two theorems give space lower bounds on directed and undirected versions of the problem. Note that the lower bounds hold even for simple graphs[2].

[1] For a statement p, define $\mathbf{1}[p] = 1$ if p is true, and $\mathbf{1}[p] = 0$ if p is false.

[2] A *simple* graph is a graph with no multiple edges.

▶ **Theorem 1.** *For $t \leq n/2$, simulating a t-step random walk on a simple directed graph in the insertion-only model within error $\varepsilon = \frac{1}{3}$ requires $\Omega(nt \log(n/t))$ bits of memory.*

▶ **Theorem 2.** *For $t = O(n^2)$, simulating a t-step random walk on a simple undirected graph in the insertion-only model within error $\varepsilon = \frac{1}{3}$ requires $\Omega(n\sqrt{t})$ bits of memory.*

Theorem 3 and Theorem 4 give near optimal space upper bounds for the problem in the insertion-only streaming model.

▶ **Theorem 3.** *We can simulate a t-step random walk on a directed graph in the insertion-only model perfectly using $O(nt)$ words*[3] *of memory. For simple directed graphs, the memory can be reduced to $O(nt \log(n/t))$ bits, assuming $t \leq n/2$.*

▶ **Theorem 4.** *We can simulate a t-step random walk on an undirected graph in the insertion-only model within error ε using $O\left(n\sqrt{t} \cdot \frac{q}{\log q}\right)$ words of memory, where $q = 2 + \frac{\log(1/\varepsilon)}{\sqrt{t}}$. In particular, the algorithm uses $O(n\sqrt{t})$ words of memory when $\varepsilon = 2^{-\Theta(\sqrt{t})}$.*

Our algorithms also extend to the turnstile model.

▶ **Theorem 5.** *We can simulate a t-step random walk on a directed graph in the turnstile model within error ε using $O(n(t + \log \frac{1}{\varepsilon}) \log^2 \max\{n, 1/\varepsilon\})$ bits of memory.*

▶ **Theorem 6.** *We can simulate a t-step random walk on an undirected graph in the turnstile model within error ε using $O(n(\sqrt{t} + \log \frac{1}{\varepsilon}) \log^2 \max\{n, 1/\varepsilon\})$ bits of memory.*

2 Directed graphs in the insertion-only model

The simplest algorithm uses $O(n^2)$ words of space (or only $O(n^2)$ *bits*, if we assume the graph is simple) to store the adjacency matrix of the graph. When $t \ll n$, a better solution is to use reservoir sampling.

▶ **Lemma 7** (Reservoir sampling)**.** *Given a stream of n items as input, we can uniformly sample m of them without replacement using $O(m)$ words of memory.*

We can also sample m items from the stream *with* replacement in $O(m)$ words of memory using m independent reservoir samplers each with capacity 1.

▶ **Theorem 8.** *We can simulate a t-step random walk on a directed graph in the insertion-only model perfectly using $O(nt)$ words of memory.*

Proof. For each vertex $u \in V$, we sample t edges $e_{u,1}, \ldots, e_{u,t}$ outgoing from u with replacement. Then we perform a random walk using these edges. When u is visited for the i-th time ($i \leq t$), we go along edge $e_{u,i}$. ◀

By treating an undirected edge as two opposite directed edges, we can achieve the same space complexity in undirected graphs.

Now we show a space lower bound for the problem. We will use a standard result from communication complexity.

▶ **Definition 9.** In the Index problem, Alice has an n-bit vector $X \in \{0,1\}^n$ and Bob has an index $i \in [n]$. Alice sends a message to Bob, and then Bob should output the bit X_i.

[3] A *word* has $\Theta(\log \max\{n, m\})$ bits.

▶ **Lemma 10** ([15]). *For any constant $1/2 < c \le 1$, solving the* Index *problem with success probability c requires sending $\Omega(n)$ bits.*

▶ **Theorem 11.** *For $t \le n/2$, simulating a t-step random walk on a simple directed graph in the insertion-only model within error $\varepsilon = \frac{1}{3}$ requires $\Omega(nt\log(n/t))$ bits of memory.*

Proof. We prove by showing a reduction from the Index problem. Before the protocol starts, Alice and Bob agree on a family $\mathcal{F}$ of t-subsets of $[n]$ [4] such that the condition $|S \cap S'| < t/2$ is satisfied for every $S, S' \in \mathcal{F}, S \neq S'$. For two independent uniform random t-subsets $S, S' \subseteq [n]$, let $p = \mathbb{P}[|S \cap S'| \ge t/2] \le \binom{t}{t/2}(\frac{t}{n})^{t/2} < (\frac{4t}{n})^{t/2}$. By union bound over all pairs of subsets, a randomly generated family $\mathcal{F}$ satisfies the condition with probability at least $1 - \binom{|\mathcal{F}|}{2}p$, which is positive when $|\mathcal{F}| = \lceil \sqrt{1/p} \rceil \ge (\frac{n}{4t})^{t/4}$. So we can choose such family $\mathcal{F}$ with $\log|\mathcal{F}| = \Omega(t\log(n/t))$.

Assume $|\mathcal{F}|$ is a power of two. Alice encodes $n\log|\mathcal{F}|$ bits as follows. Let G be a directed graph with vertex set $\{v_0, v_1, \dots, v_{2n}\}$. For each vertex $u \in \{v_{n+1}, v_{n+2}, \dots, v_{2n}\}$, Alice chooses a set $S_u \in \mathcal{F}$, and inserts an edge (u, v_i) for every $i \in S_u$.

Suppose Bob wants to query S_u. He adds an edge (v, u) for every $v \in \{v_0, v_1, v_2, \dots, v_n\}$, and then simulates a random walk starting from v_0. The random walk visits u every two steps, and it next visits v_i for some random $i \in S_u$. At least $t/2$ different elements from S_u can be seen in $2t$ samples with probability at least $1 - \binom{t}{t/2}(\frac{1}{2})^{2t} \ge 1 - 2^{-t}$, so S_u can be uniquely determined by an $O(t)$-step random walk (simulated within error ε) with probability $1 - 2^{-t} - \frac{\varepsilon}{2} > \frac{1}{2}$. By Lemma 10, the space usage for simulating the $O(t)$-step random walk is at least $\Omega(n\log|\mathcal{F}|) = \Omega(nt\log(n/t))$ bits. The theorem is proved by scaling down n and t by a constant factor. ◀

For simple graphs, we can achieve an upper bound of $O(nt\log(n/t))$ bits.

▶ **Theorem 12.** *For $t \le n/2$, we can simulate a t-step random walk on a simple directed graph in the insertion-only model perfectly using $O(nt\log(n/t))$ bits of memory.*

Proof. For every $u \in V$, we run a reservoir sampler with capacity t, which samples (at most) t edges from u's outgoing edges *without* replacement. After reading the entire input stream, we begin simulating the random walk. When u is visited during the simulation, in the next step we choose at random an outgoing edge used before with probability $d_{\text{used}}(u)/d(u)$, or an unused edge from the reservoir sampler with probability $1 - d_{\text{used}}(u)/d(u)$, where $d_{\text{used}}(u)$ is the number of edges in u's sampler that are previously used in the simulation. We maintain a t-bit vector to keep track of these used samples.

The number of different possible states of a sampler is at most $\sum_{0\le i\le t}\binom{n}{i} \le (t+1)(\frac{en}{t})^t$, so it can be encoded using $\left\lceil \log\left((t+1)(\frac{en}{t})^t\right)\right\rceil = O(t\log(n/t))$ bits. The total space is $O(nt\log(n/t))$ bits. ◀

3 Undirected graphs in the insertion-only model

3.1 A space lower bound

▶ **Theorem 13.** *For $t = O(n^2)$, simulating a t-step random walk on a simple undirected graph in the insertion-only model within error $\varepsilon = \frac{1}{3}$ requires $\Omega(n\sqrt{t})$ bits of memory.*

[4] Define $[n] = \{1, 2, \dots, n\}$. A t-subset is a subset of size t.

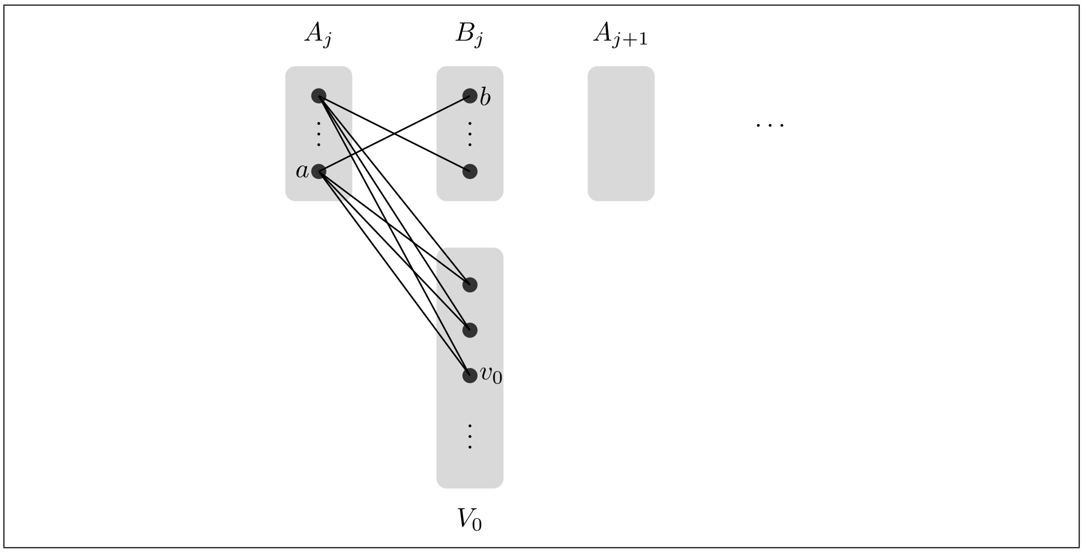

Figure 1 Proof of Theorem 13.

Proof. Again we show a reduction from the INDEX problem.

Alice encodes $\Omega(n\sqrt{t})$ bits as follows. Let G be an undirected graph with vertex set $V_0 \cup V_1 \cup \cdots \cup V_{n/\sqrt{t}}$, where each V_j has size $2\sqrt{t}$, and the starting vertex $v_0 \in V_0$. For each $j \geq 1$, V_j is divided into two subsets A_j, B_j with size $\sqrt{t}$ each, and Alice encodes $|A_j| \times |B_j| = t$ bits by inserting a subset of edges from $\{(u,v) : u \in A_j, v \in B_j\}$. In total she encodes $t \cdot n/\sqrt{t} = n\sqrt{t}$ bits.

Suppose Bob wants to query some bit, i.e., he wants to see whether a and b are connected by an edge. Assume $(a,b) \in A_j \times B_j$. He adds an edge (u,v) for every $u \in A_j$ and every $v \in V_0$ (see Figure 1). A perfect random walk starting from $v_0 \in V_0$ will be inside the bipartite subgraph $(A_j, B_j \cup V_0)$. Suppose the current vertex of the perfect random walk is $v_i \in A_j$. If a, b are connected by an edge, then

$$\begin{aligned}
&\mathbb{P}[(v_{i+2}, v_{i+3}) = (a,b) \mid v_i] \\
\geq\ &\mathbb{P}[v_{i+1} \in V_0 \mid v_i]\, \mathbb{P}[v_{i+2} = a \mid v_{i+1} \in V_0]\, \mathbb{P}[v_{i+3} = b \mid v_{i+2} = a] \\
\geq\ &\frac{|V_0|}{|V_0| + |B_j|} \cdot \frac{1}{|A_j|} \cdot \frac{1}{|V_0| + |B_j|} \\
\geq\ &\frac{2}{9t},
\end{aligned}$$

so in every four steps the edge (a,b) is passed with probability $\Omega(\frac{1}{t})$. Then a $O(t)$-step perfect random walk will pass the edge (a,b) with probability 0.9. Hence Bob can know whether the edge (a,b) exists by looking at the random walk (simulated within error ε) with success probability $0.9 - \frac{\varepsilon}{2} > 1/2$. By Lemma 10, the space usage for simulating the $O(t)$-step random walk is at least $\Omega(n\sqrt{t})$ bits. The theorem is proved by scaling down n and t by a constant factor. ◀

3.2 An algorithm for simple graphs

Now we describe our algorithm for undirected graphs in the insertion-only model. As a warm-up, we consider simple graphs in this section. We will deal with multi-edges in Section 3.3.

Intuition

We start by informally explaining the intuition of our algorithm for simple undirected graphs.

We maintain a subset of $O(n\sqrt{t})$ edges from the input graph, and use them to simulate the random walk after reading the entire input stream.

For a vertex u with degree smaller than $\sqrt{t}$, we can afford to store all its neighboring edges in memory. For u with degree greater than $\sqrt{t}$, we can only sample and store $O(\sqrt{t})$ of its neighboring edges. During the simulation, at every step we first toss a coin to decide whether the next vertex has small degree or large degree. In the latter case, we have to pick a sampled neighboring edge and walk along it. If all sampled neighboring edges have already been used, our algorithm fails. Using the large degree and the fact that edges are undirected, we can show that the failure probability is low.

Description of the algorithm

We divide the vertices into two types according to their degrees: the set of *big* vertices $B = \{u \in V : d(u) \geq C + 1\}$, and the set of *small* vertices $S = \{u \in V : d(u) \leq C\}$, where parameter C is an positive integer to be determined later.

We use *arc* (u, v) to refer to an edge when we want to specify the direction $u \to v$. So an undirected edge (u, v) corresponds to two different[5] arcs, arc (u, v) and arc (v, u).

We say an arc (u, v) is *important* if $v \in S$, or *unimportant* if $v \in B$. Denote the set of important arcs by E_1, and the set of unimportant arcs by E_0. The total number of important arcs equals $\sum_{s \in S} d(s) \leq |S|C$, so it is possible to store E_1 in $O(nC)$ words of space.

The set E_0 of unimportant arcs can be huge, so we only store a subset of E_0. For every vertex u, we sample with replacement C unimportant arcs outgoing from u, denoted by $a_{u,1}, \ldots, a_{u,C}$.

To maintain the set E_1 of important arcs and the samples of unimportant arcs after every edge insertion, we need to handle the events when some small vertex becomes big. This procedure is straightforward, as described by ProcessInput in Figure 2. Since $|E_1|$ never exceeds nC, and each of the n samplers uses $O(C)$ words of space, the overall space complexity is $O(nC)$ words.

We begin simulating the random walk after ProcessInput finishes. When the current vertex of the random walk is v, with probability $d_1(v)/d(v)$ the next step will be along an important arc, where $d_1(v)$ denotes the number of important arcs outgoing from v. In this case we simply choose a uniform random vertex from $\{u : (v, u) \in E_1\}$ as the next vertex. However, if the next step is along an unimportant arc, we need to choose an unused sample $a_{v,j}$ and go along this arc. If at this time all C samples $a_{v,j}$ are already used, then our algorithm fails (and is allowed to return an arbitrary walk). The pseudocode of this simulating procedure is given in Figure 3.

In a walk $w = (v_0, \ldots, v_t)$, we say vertex u *fails* if $|\{i : v_i = u \text{ and } (v_i, v_{i+1}) \in E_0\}| > C$. If no vertex fails in w, then our algorithm will successfully return w with probability $\mathcal{RW}_{v_0,t}(w)$. Otherwise our algorithm will fail after some vertex runs out of the sampled unimportant arcs. To ensure the output distribution is ε-close to $\mathcal{RW}_{v_0,t}$ in ℓ_1 distance, it suffices to make our algorithm fail with probability at most $\varepsilon/2$, by choosing a large enough capacity C.

[5] We have assumed no self-loops exist, so $u \neq v$.

```
procedure InsertArc($u, v$)
    $d(v) \leftarrow d(v) + 1$
    if $d(v) = C + 1$ then                              ▷ $v$ changes from small to big
        for $x \in V$ such that $(x, v) \in E_1$ do     ▷ arc $(x, v)$ becomes unimportant
            $E_1 \leftarrow E_1 \backslash \{(x, v)\}$
            Feed arc $(x, v)$ into $x$'s sampler
        end for
    end if
    if $d(v) \leq C$ then                                               ▷ $v \in S$
        $E_1 \leftarrow E_1 \cup \{(u, v)\}$
    else                                                                ▷ $v \in B$
        Feed arc $(u, v)$ into $u$'s sampler
    end if
end procedure
procedure ProcessInput
    $E_1 \leftarrow \emptyset$                                ▷ Set of important arcs
    for $u \in V$ do
        $d(u) \leftarrow 0$
        Initialize $u$'s sampler (initially empty) which maintains $a_{u,1}, \ldots, a_{u,C}$
    end for
    for undirected edge $(u, v)$ in the input stream do
        InsertArc($u, v$)
        InsertArc($v, u$)
    end for
end procedure
```

Figure 2 Pseudocode for processing the input stream (for simple undirected graphs).

```
procedure SimulateRandomWalk($v_0, t$)
    for $v \in V$ do
        $c(v) \leftarrow 0$                                  ▷ counter of used samples
    end for
    for $i = 0, \ldots, t - 1$ do
        $N_1 \leftarrow \{u : (v_i, u) \in E_1\}$
        $x \leftarrow$ uniformly random integer from $\{1, 2, \ldots, d(v_i)\}$
        if $x \leq |N_1|$ then
            $v_{i+1} \leftarrow$ uniformly random vertex from $N_1$
        else
            $j \leftarrow c(v_i) + 1$
            $c(v_i) \leftarrow j$
            if $j > C$ then return Fail
            else
                $v_{i+1} \leftarrow u$, where $(v_i, u) = a_{v_i,j}$
            end if
        end if
    end forreturn $(v_0, \ldots, v_t)$
end procedure
```

Figure 3 Pseudocode for simulating a t-step random walk starting from v_0.

To bound the probability $\mathbb{P}[\text{at least one vertex fails} \mid v_0 = s]$[6], we will bound the individual failure probability of every vertex, and then use union bound.

▶ **Lemma 14.** *Suppose for every $u \in V$, $\mathbb{P}[u \text{ fails} \mid v_0 = u] \leq \delta$. Then for any starting vertex $s \in V$, $\mathbb{P}[\text{at least one vertex fails} \mid v_0 = s] \leq t\delta$.*

Proof. Fix a starting vertex s. For any particular $u \in V$,

$$\begin{aligned}
&\mathbb{P}[u \text{ fails} \mid v_0 = s] \\
&= \mathbb{P}[u \text{ fails, and } \exists i \leq t-1, v_i = u \mid v_0 = s] \\
&= \mathbb{P}[\exists i \leq t-1, v_i = u \mid v_0 = s]\, \mathbb{P}[u \text{ fails} \mid v_0 = s, \text{ and } \exists i \leq t-1, v_i = u] \\
&\leq \mathbb{P}[\exists i \leq t-1, v_i = u \mid v_0 = s]\, \mathbb{P}[u \text{ fails} \mid v_0 = u] \\
&\leq \mathbb{P}[\exists i \leq t-1, v_i = u \mid v_0 = s] \cdot \delta.
\end{aligned}$$

By union bound,

$$\begin{aligned}
&\mathbb{P}[\text{at least one vertex fails} \mid v_0 = s] \\
&\leq \sum_{u \in V} \mathbb{P}[u \text{ fails} \mid v_0 = s] \\
&\leq \sum_{u \in V} \mathbb{P}[\exists i \leq t-1, v_i = u \mid v_0 = s] \cdot \delta \\
&= \mathbb{E}[\text{number of distinct vertices visited in } \{v_0, \ldots, v_{t-1}\} \mid v_0 = s] \cdot \delta \\
&\leq t\delta.
\end{aligned}$$

◀

▶ **Lemma 15.** *We can choose integer parameter $C = O\left(\sqrt{t} \cdot \frac{q}{\log q}\right)$, where $q = 2 + \frac{\log(1/\delta)}{\sqrt{t}}$, so that $\mathbb{P}[u \text{ fails} \mid v_0 = u] \leq \delta$ holds for every $u \in V$.*

Proof. Let $d_0(u) = |\{v : (u, v) \in E_0\}|$.

For any $u \in V$,

$$\begin{aligned}
&\mathbb{P}[u \text{ fails} \mid v_0 = u] \\
&\leq \mathbb{P}[u \text{ fails} \mid v_0 = u, (v_0, v_1) \in E_0].
\end{aligned}$$

We rewrite this probability as the sum of probabilities of possible random walks in which u fails. Recall that u fails if and only if $|\{i : v_i = u, (v_i, v_{i+1}) \in E_0\}| \geq C + 1$. In the summation over possible random walks, we only keep the shortest prefix $(v_0, \ldots, v_k)$ in which u fails, i.e., the last step (v_{k-1}, v_k) is the $(C+1)$-st time walking along an unimportant arc outgoing from u. We have

$$\begin{aligned}
&\mathbb{P}[u \text{ fails} \mid v_0 = u, (v_0, v_1) \in E_0] \\
&= \sum_{k \leq t} \sum_{\text{walk}(v_0, \ldots, v_k)} \mathbf{1}\Big[v_0 = v_{k-1} = u,\ (v_0, v_1), (v_{k-1}, v_k) \in E_0, \\
&\qquad\qquad |\{i : v_i = u, (v_i, v_{i+1}) \in E_0\}| = C + 1\Big] \frac{1}{d_0(u)} \prod_{i=1}^{k-1} \frac{1}{d(v_i)} \\
&= \sum_{k \leq t} \sum_{\text{walk}(v_0, \ldots, v_{k-1})} \mathbf{1}\Big[v_0 = v_{k-1} = u,\ (v_0, v_1) \in E_0, \\
&\qquad\qquad |\{i : v_i = u, (v_i, v_{i+1}) \in E_0\}| = C\Big] \prod_{i=1}^{k-1} \frac{1}{d(v_i)}.
\end{aligned} \tag{2}$$

[6] If not specified, assume the probability space is over all t-step random walks $(v_0, \ldots, v_t)$ starting from v_0.

Let $v'_i = v_{k-1-i}$. Since the graph is undirected, the vertex sequence $(v'_0, \ldots, v'_{k-1})$ (the reversal of walk $(v_0, \ldots, v_{k-1})$) is also a walk starting from and ending at u. So the summation (2) equals

$$\sum_{k \le t} \sum_{\text{walk}(v'_0,\ldots,v'_{k-1})} \mathbf{1}\Big[v'_0 = v'_{k-1} = u,\ (v'_{k-1}, v'_{k-2}) \in E_0, \\ |\{i : v'_i = u, (v'_i, v'_{i-1}) \in E_0\}| = C\Big] \prod_{i=0}^{k-2} \frac{1}{d(v'_i)}$$

$$= \mathop{\mathbb{P}}_{\text{random walk } (v'_0,\ldots,v'_{t-1})}\Big[|\{i : v'_i = u, (v'_i, v'_{i-1}) \in E_0\}| \ge C \,\Big|\, v'_0 = u\Big].$$

Recall that $(v'_i, v'_{i-1}) \in E_0$ if and only if $v'_{i-1} \in B$. For any $1 \le i \le t-1$ and any fixed prefix $v'_0, \ldots, v'_{i-1}$,

$$\begin{aligned} &\mathbb{P}\big[v'_i = u, (v'_i, v'_{i-1}) \in E_0 \,\big|\, v'_0, \ldots, v'_{i-1}\big] \\ &\le \mathbf{1}[v'_{i-1} \in B] \cdot \frac{1}{d(v'_{i-1})} \\ &< \frac{1}{C}. \end{aligned} \tag{3}$$

Hence the probability that $|\{1 \le i \le t-1 : v'_i = u, (v'_i, v'_{i-1}) \in E_0\}| \ge C$ is at most

$$\begin{aligned} &\binom{t-1}{C}\left(\frac{1}{C}\right)^C \\ &\le \left(\frac{e(t-1)}{C}\right)^C \left(\frac{1}{C}\right)^C \\ &< \left(\frac{et}{C^2}\right)^C. \end{aligned}$$

We set $C = \lceil 4\sqrt{t}\, q/\log q \rceil$, where $q = 2 + \log(1/\delta)/\sqrt{t} > 2$. Notice that $q/\log^2 q > 1/4$. Then

$$C \log\left(\frac{C^2}{et}\right) \ge \frac{4\sqrt{t}q}{\log q} \log\left(\frac{16q^2}{e\log^2 q}\right) > \frac{4\sqrt{t}q}{\log q}\log(4q/e) > 4\sqrt{t}q > \log(1/\delta),$$

so

$$\left(\frac{et}{C^2}\right)^C < \delta.$$

Hence we have made $\mathbb{P}[u \text{ fails} \mid v_0 = u] < \delta$ by choosing $C = O(\sqrt{t}q/\log q)$. ◀

▶ **Theorem 16.** *We can simulate a t-step random walk on a simple undirected graph in the insertion-only model within error ε using $O\left(n\sqrt{t} \cdot \frac{q}{\log q}\right)$ words of memory, where $q = 2 + \frac{\log(1/\varepsilon)}{\sqrt{t}}$.*

Proof. The theorem follows from Lemma 14 and Lemma 15 by setting $\delta = \frac{\varepsilon}{2t}$. ◀

3.3 On graphs with multiple edges

When the undirected graph contains multiple edges, condition (3) in the proof of Lemma 15 may not hold, so we need to slightly modify our algorithm.

We still maintain the multiset E_1 of important arcs. Whether an arc is important will be determined by our algorithm. (This is different from the previous algorithm, where important arcs were simply defined as (u, v) with $d(v) \leq C$.) We will ensure that condition (3) still holds, i.e., for any $u \in V$ and any fixed prefix of the random walk $v_0, \ldots, v_{i-1}$,

$$\mathbb{P}\big[(v_i, v_{i-1}) \notin E_1, \text{ and } v_i = u \,\big|\, v_0, \ldots, v_{i-1}\big] < 1/C. \tag{4}$$

Note that there can be both important arcs and unimportant arcs from u to v. Let $f(u, v)$ denote the number of undirected edges between u, v. Then there are $f(u, v)$ arcs (u, v). Suppose $f_1(u, v)$ of these arcs are important, and $f_0(u, v) = f(u, v) - f_1(u, v)$ of them are unimportant. Then we can rewrite condition (4) as

$$\frac{f_0(u, v_{i-1})}{d(v_{i-1})} < 1/C, \tag{5}$$

for every $u, v_{i-1} \in V$.

Similarly as before, we need to store the multiset E_1 using only $O(nC)$ words of space. And we need to sample with replacement C unimportant arcs $a_{u,1}, \ldots, a_{u,C}$ outgoing from u, for every $u \in V$. Finally we use the procedure SimulateRandomWalk in Figure 3 to simulate a random walk.

The multiset E_1 is determined as follows: For every vertex $v \in V$, we run Misra-Gries algorithm [16] on the sequence of all v's neighbors. We will obtain a list L_v of at most C vertices, such that for every vertex $u \notin L_v$, $\frac{f(u,v)}{d(v)} < \frac{1}{C}$. Moreover, we will get a frequency estimate $A_v(u) > 0$ for every $u \in L_v$, such that $0 \leq f(u, v) - A_v(u) < \frac{d(v)}{C}$. Assuming $A_v(u) = 0$ for $u \notin L_v$, we can satisfy condition (5) for all $u \in V$ by setting $f_1(u, v) = A_v(u)$. Hence we have determined all the important arcs, and they can be stored in $O(\sum_v |L_v|) = O(nC)$ words. To sample from the unimportant arcs, we simply insert the arcs discarded by Misra-Gries algorithm into the samplers. The pseudocode is given in Figure 4.

▶ **Lemma 17.** *After* ProcessInput *(in Figure 4) finishes,* $|L_v| \leq C$*. For every* $u \in L_v$*,* $0 \leq f(u, v) - A_v(u) \leq \frac{d(v)}{C+1}$*. For every* $u \notin L_v$*,* $f(u, v) \leq \frac{d(v)}{C+1}$*.*

Proof. Every time the **for** loop in procedure InsertArc finishes, the newly added vertex u must have been removed from L_v, so $|L_v| \leq C$ still holds. Let $W = \{w_1, \cdots, w_{C+1}\}$ be the set of vertices in L_v before this **for** loop begins. Then for every $u \in V$, $f(u, v) - A_v(u)$ equals the number of times u is contained in W (assuming $A_v(u) = 0$ for $u \notin L_v$), which is at most $\frac{1}{C+1} \sum_W |W| \leq \frac{d(v)}{C+1}$. ◀

▶ **Corollary 18.** *Procedure* ProcessInput *in Figure 4 computes the multiset* E_1 *of important edges and stores it using* $O(nC)$ *words. It also samples with replacement* C *unimportant arcs* $a_{u,1}, \ldots, a_{u,C}$ *outgoing from* u*, for every* $u \in V$*. Moreover,*

$$\frac{f_0(u, v)}{d(v)} < \frac{1}{C}$$

holds for every $u, v \in V$*.*

Now we analyze the failure probability of SimulateRandomWalk (in Figure 3), similar to Lemma 15.

▶ **Lemma 19.** *We can choose integer parameter* $C = O\left(\sqrt{t} \cdot \frac{q}{\log q}\right)$*, where* $q = 2 + \frac{\log(1/\delta)}{\sqrt{t}}$*, so that* $\mathbb{P}[u \text{ fails} \mid v_0 = u] \leq \delta$ *holds for every* $u \in V$*.*

```
procedure InsertArc($(u, v)$)
    $d(v) \leftarrow d(v) + 1$
    if $u \in L_v$ then
        $A_v(u) \leftarrow A_v(u) + 1$
    else
        Insert $u$ into $L_v$
        $A_v(u) \leftarrow 1$
        if $|L_v| \geq C + 1$ then
            for $w \in L_v$ do
                Feed arc $(w, v)$ into $w$'s sampler
                $A_v(w) \leftarrow A_v(w) - 1$
                if $A_v(w) = 0$ then
                    Remove $w$ from $L_v$
                end if
            end for
        end if
    end if
end procedure
procedure ProcessInput
    for $u \in V$ do
        $d(u) \leftarrow 0$
        Initialize $u$'s sampler (initially empty) which maintains $a_{u,1}, \dots, a_{u,C}$
        Initialize empty list $L_u$
    end for
    for undirected edge $(u, v)$ in the input stream do
        InsertArc($(u, v)$)
        InsertArc($(v, u)$)
    end for
    $E_1 \leftarrow \bigcup_{v \in V} \bigcup_{u \in L_v} \{A_v(u) \text{ copies of arc } (u, v)\}$        ▷ Multiset of important arcs
end procedure
```

Figure 4 Pseudocode for processing the input stream (for undirected graphs with possibly multiple edges).

Proof. Let $d_0(u) = \sum_{v \in V} f_0(u, v)$. As before, we rewrite this probability as a sum over possible random walks. Here we distinguish between important and unimportant arcs. Denote $s_i = \mathbf{1}[\text{step } (v_{i-1}, v_i) \text{ is along an important arc}]$. Then for any $u \in V$,

$$\begin{aligned} &\mathbb{P}[u \text{ fails} \mid v_0 = u] \\ &\le \mathbb{P}[u \text{ fails} \mid v_0 = u, \text{ arc } (v_0, v_1) \text{ is unimportant}] \\ &= \frac{d(u)}{d_0(u)} \sum_{k \le t} \sum_{(v_0, \dots, v_k)} \sum_{s_1, \dots, s_k} \mathbf{1}\big[v_0 = v_{k-1} = u,\ s_1 = s_k = 0, \\ &\qquad\qquad |\{i : v_i = u, s_{i+1} = 0\}| = C + 1\big] \prod_{i=0}^{k-1} \frac{f_{s_{i+1}}(v_i, v_{i+1})}{d(v_i)} \\ &= \sum_{k \le t} \sum_{(v_0, \dots, v_{k-1})} \sum_{s_1, \dots, s_{k-1}} \mathbf{1}\big[v_0 = v_{k-1} = u,\ s_1 = 0, \\ &\qquad\qquad |\{i : v_i = u, s_{i+1} = 0\}| = C\big] \prod_{i=0}^{k-2} \frac{f_{s_{i+1}}(v_i, v_{i+1})}{d(v_i)}. \end{aligned}$$

Let $v'_i = v_{k-1-i}, s'_i = s_{k-i}$. Then this sum equals

$$\sum_{k\le t}\sum_{(v'_0,\ldots,v'_{k-1})}\sum_{s'_1,\ldots,s'_{k-1}} \mathbf{1}\Big[v'_0 = v'_{k-1} = u,\ s'_{k-1} = 0,\quad |\{i : s'_i = 0, v'_i = u\}| = C\Big] \prod_{i=1}^{k-1} \frac{f_{s'_i}(v'_i, v'_{i-1})}{d(v'_{i-1})}$$

$$= \mathop{\mathbb{P}}_{\text{random walk } (v'_0,\ldots,v'_{t-1})}\Big[|\{i : v'_i = u, \text{ arc } (v'_i, v'_{i-1}) \text{ is unimportant}\}| \ge C \,\Big|\, v'_0 = u\Big].$$

Notice that for any i and any fixed prefix $v'_0, \ldots, v'_{i-1}$,

$$\mathbb{P}\Big[v'_i = u, \text{ arc } (v'_i, v'_{i-1}) \text{ is unimportant} \,\Big|\, v'_0, v'_1, \ldots, v'_{i-1}\Big] = \frac{f_0(u, v'_{i-1})}{d(v'_{i-1})} < \frac{1}{C}$$

by Corollary 18. The rest of the proof is the same as in Lemma 15. ◀

▶ **Theorem 20.** *We can simulate a random walk on an undirected graph with possibly multiple edges in the insertion-only model within error ε using $O\left(n\sqrt{t}\cdot\frac{q}{\log q}\right)$ words of memory, where $q = 2 + \frac{\log(1/\varepsilon)}{\sqrt{t}}$.*

Proof. The theorem follows from Lemma 14 and Lemma 19 by setting $\delta = \frac{\varepsilon}{2t}$. ◀

4 Turnstile model

In this section we consider the turnstile model where both insertion and deletion of edges can appear.

▶ **Lemma 21** (ℓ_1 sampler in the turnstile model, [9]). *Let $f \in \mathbb{R}^n$ be a vector defined by a stream of updates to its coordinates of the form $f_i \leftarrow f_i + \Delta$, where Δ can either be positive or negative. There is an algorithm which reads the stream and returns an index $i \in [n]$ such that for every $j \in [n]$,*

$$\mathbb{P}[i = j] = \frac{|f_j|}{\|f\|_1} + O(n^{-c}), \tag{6}$$

where $c \ge 1$ is some arbitrary large constant. It is allowed to output Fail *with probability δ, and in this case it will not output any index. The space complexity of this algorithm is $O(\log^2 n \log(1/\delta))$ bits.*

▶ Remark. For $\varepsilon \ll 1/n$, the $O(n^{-c})$ error term in (6) can be reduced to $O(\varepsilon^c)$ by running the ℓ_1 sampler on $f \in \mathbb{R}^{\lceil 1/\varepsilon \rceil}$, using $O(\log^2(1/\varepsilon)\log(1/\delta))$ bits of space.

We will use the ℓ_1 sampler for sampling neighbors (with possibly multiple edges) in the turnstile model. The error term $O(n^{-c})$ (or $O(\varepsilon^c)$) in (6) can be ignored in the following discussion, by choosing sufficiently large constant c and scaling down ε by a constant.

4.1 Directed graphs

▶ **Theorem 22.** *We can simulate a t-step random walk on a directed graph in the turnstile model within error ε using $O(n(t + \log\frac{1}{\varepsilon})\log^2\max\{n, 1/\varepsilon\})$ bits of memory.*

Proof. For every $u \in V$, we run $C' = 2t + 16\log(2t/\varepsilon)$ independent ℓ_1 samplers each having failure probability $\delta = 1/2$. We use them to sample the outgoing edges of u (as in the algorithm of Theorem 8). By Chernoff bound, the probability that less than t samplers succeed is at most $\varepsilon/(2t)$.

We say a vertex u fails if u has less than t successful samplers, and $u \in \{v_0, v_1, \ldots, v_{t-1}\}$ (where $v_0, v_1, \ldots, v_t$ is the random walk). Then $\mathbb{P}[u \text{ fails}] \le \frac{\varepsilon}{2t}\mathbb{P}[u \in \{v_0, \ldots, v_{t-1}\}]$. By union bound, $\mathbb{P}[\text{at least one vertex fails}] \le \frac{\varepsilon}{2t}\sum_{u\in V}\mathbb{P}[u \in \{v_0, \ldots, v_{t-1}\}] \le \frac{\varepsilon}{2}$. Hence, with probability $1 - \frac{\varepsilon}{2}$, every vertex u visited (except the last one) has at least t outgoing edges sampled, so our simulation can succeed. The space usage is $O(nC'\log^2\max\{n, 1/\varepsilon\}\log(1/\delta)) = O(n(t + \log\frac{1}{\varepsilon})\log^2\max\{n, 1/\varepsilon\})$ bits. ◀

4.2 Undirected graphs

We slightly modify the ProcessInput procedure of our previous algorithm in Section 3.3. We will use the ℓ_1 heavy hitter algorithm in the turnstile model.

▶ **Lemma 23** (ℓ_1 heavy hitter, [6]). *Let $f \in \mathbb{R}^n$ be a vector defined by a stream of updates to its coordinates of the form $f_i \leftarrow f_i + \Delta$, where Δ can either be positive or negative. There is a randomized algorithm which reads the stream and returns a subset $L \subseteq [n]$ such that $i \in L$ for every $|f_i| \ge \frac{\|f\|_1}{k}$, and $i \notin L$ for every $|f_i| \le \frac{\|f\|_1}{2k}$. Moreover it returns a frequency estimate $\tilde{f}_i$ for every $i \in L$, which satisfies $0 \le f_i - \tilde{f}_i \le \frac{\|f\|_1}{2k}$. The failure probability of this algorithm is $O(n^{-c})$. The space complexity is $O(k\log^2 n)$ bits.*

▶ Remark. For $\varepsilon \ll 1/n$, the $O(n^{-c})$ failure probability of this ℓ_1 heavy hitter algorithm can be reduced to $O(\varepsilon^c)$ by running the algorithm on $f \in \mathbb{R}^{\lceil 1/\varepsilon\rceil}$, using $O(k\log^2(1/\varepsilon))$ bits of space. In the following discussion, this failure probability can be ignored by making the constant c sufficiently large.

▶ **Theorem 24.** *We can simulate a t-step random walk on an undirected graph in the turnstile model within error ε using $O(n(\sqrt{t} + \log\frac{1}{\varepsilon})\log^2\max\{n, 1/\varepsilon\})$ bits of memory.*

Proof. Similar to the previous insertion-only algorithm (in Figure 4), we perform two *arc updates* $((u,v),\Delta)$, $((v,u),\Delta)$ when we read an *edge update* $((u,v),\Delta)$ from the stream.

For every $u \in V$, we run $C' = 2C + 16\log(2t/\varepsilon)$ independent ℓ_1 samplers each having failure probability $\delta = 1/2$, where C is the same constant as in the proof of Lemma 19 and Theorem 20. By Chernoff bound, the probability that less than C samplers succeed is at most $\varepsilon/(2t)$. For every arc update $((u,v),\Delta)$, we send update $(v,\ \Delta)$ to u's ℓ_1 sampler.

In addition, for every $v \in V$, we run ℓ_1 heavy hitter algorithm with $k = C$. For every arc update $((u,v),\Delta)$, we send update (u,Δ) to v's heavy hitter algorithm. In the end, we will get a frequency estimate $A_v(u)$ for every $u \in V$, such that $f(u,v) - \frac{d(v)}{C} \le A_v(u) \le f(u,v)$. We then insert $A_v(u)$ copies of arc (u,v) into E_1 (the multiset of important arcs), and send update $(v, -A_v(u))$ to u's ℓ_1 sampler. Then we use the ℓ_1 samplers to sample unimportant arcs for every u.

As before, we use the procedure SimulateRandomWalk (in Figure 3) to simulate the random walk. The analysis of the failure probability of the ℓ_1 samplers is the same as in Theorem 22. The analysis of the failure probability of procedure SimulateRandomWalk is the same as in Lemma 19. The space usage of the algorithm is $O(nC'\log^2\max\{n, 1/\varepsilon\}\log\delta) = O(n(\sqrt{t} + \log\frac{1}{\varepsilon})\log^2\max\{n, 1/\varepsilon\})$ bits. ◀

5 Conclusion

We end our paper by discussing some related questions for future research.

- The output distribution of our insertion-only algorithm for undirected graphs is ε-close to the random walk distribution. What if the output is required to be perfectly random, i.e., $\varepsilon = 0$?
- For insertion-only simple undirected graphs, we proved an $\Omega(n\sqrt{t})$-bit space lower bound. Our algorithm uses $O(n\sqrt{t}\log n)$ bits (for not too small ε). Can we close the gap between the lower bound and the upper bound, as in the case of directed graphs?
- In the undirected version, suppose the starting vertex v_0 is drawn from a distribution (for example, the stationary distribution of the graph) rather than being specified. Is it possible to obtain a better algorithm in this new setting? Notice that our proof of the $\Omega(n\sqrt{t})$ lower bound does not work here, since it requires v_0 to be specified.
- We required the algorithm to output all vertices on the random walk. If only the last vertex is required, can we get a better algorithm or prove non-trivial lower bounds?

References

1 Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. *Information and Computation*, 222:59–79, 2013. doi:10.1016/j.ic.2012.10.006.

2 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 459–467, 2012. doi:10.1137/1.9781611973099.40.

3 Reid Andersen, Fan Chung, and Kevin Lang. Using pagerank to locally partition a graph. *Internet Mathematics*, 4(1):35–64, 2007. doi:10.1080/15427951.2007.10129139.

4 Reid Andersen and Yuval Peres. Finding sparse cuts locally using evolving sets. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC)*, pages 235–244, 2009. doi:10.1145/1536414.1536449.

5 Moses Charikar, Liadan O'Callaghan, and Rina Panigrahy. Better streaming algorithms for clustering problems. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, pages 30–39, 2003. doi:10.1145/780542.780548.

6 Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005. doi:10.1016/j.jalgor.2003.12.001.

7 Atish Das Sarma, Sreenivas Gollapudi, and Rina Panigrahy. Estimating pagerank on graph streams. *Journal of the ACM (JACM)*, 58(3):13, 2011. doi:10.1145/1970392.1970397.

8 Leah Epstein, Asaf Levin, Julián Mestre, and Danny Segev. Improved Approximation Guarantees for Weighted Matching in the Semi-streaming Model. *SIAM Journal on Discrete Mathematics*, 25(3):1251–1265, 2011. doi:10.1137/100801901.

9 Rajesh Jayaram and David P. Woodruff. Perfect Lp Sampling in a Data Stream. In *Proceedings of the 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 544–555, 2018. doi:10.1109/FOCS.2018.00058.

10 Mark Jerrum and Alistair Sinclair. Approximating the permanent. *SIAM Journal on Computing*, 18(6):1149–1178, 1989. doi:10.1137/0218077.

11 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. doi:10.1016/0304-3975(86)90174-X.

12 Michael Kapralov. Better bounds for matchings in the streaming model. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1679–1697, 2013. doi:10.1137/1.9781611973105.121.

13 Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. *SIAM Journal on Computing*, 46(1):456–477, 2017. doi:10.1137/141002281.

14 Jonathan A. Kelner and Alex Levin. Spectral sparsification in the semi-streaming setting. *Theory of Computing Systems*, 53(2):243–262, 2013. doi:10.1007/s00224-012-9396-1.

15 Peter Bro Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On Data Structures and Asymmetric Communication Complexity. *Journal of Computer and System Sciences*, 57(1):37–49, 1998. doi:10.1006/jcss.1998.1577.

16 J. Misra and David Gries. Finding repeated elements. *Science of Computer Programming*, 2(2):143–152, 1982. doi:10.1016/0167-6423(82)90012-0.

17 Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):17, 2008. doi:10.1145/1391289.1391291.

18 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. doi:10.1137/080744888.