Simulating Random Walks on Graphs in the Streaming Model

Ce Jin

Tsinghua University

ITCS 2019

Problem Definition

Insertion-only graph streaming model

Let G be the (directed or undirected) input graph with n vertices.
The edges of G come as an input stream $(e_1, e_2, \ldots, e_m)$.
A streaming algorithm must read the edges one by one in this order.

Problem Definition

Insertion-only graph streaming model

Let G be the (directed or undirected) input graph with n vertices.
The edges of G come as an input stream $(e_1, e_2, \ldots, e_m)$.
A streaming algorithm must read the edges one by one in this order.

Random walk on graph

A sequence of vertices $(v_0, v_1, \ldots, v_t)$ starting from v_0.
For $i = 1, 2, \ldots, t$, (v_{i-1}, v_i) is a uniform random edge drawn from the edges adjacent to v_{i-1}.

Problem Definition

Insertion-only graph streaming model

Let G be the (directed or undirected) input graph with n vertices.
The edges of G come as an input stream $(e_1, e_2, \ldots, e_m)$.
A streaming algorithm must read the edges one by one in this order.

Random walk on graph

A sequence of vertices $(v_0, v_1, \ldots, v_t)$ starting from v_0.
For $i = 1, 2, \ldots, t$, (v_{i-1}, v_i) is a uniform random edge drawn from the edges adjacent to v_{i-1}.

Our problem: Simulating a t-step random walk

A starting vertex v_0 is given at the end of the input stream.
The streaming algorithm outputs a random **sequence** $(v_0, v_1, \ldots, v_t)$.
The ℓ_1 distance between the output distribution and the distribution of t-step random walks is less than ε.

A simple algorithm

Reservoir Sampling

Given a stream of elements as input, one can uniformly sample m elements from them using $O(m)$ space.

A simple algorithm

Reservoir Sampling

Given a stream of elements as input, one can uniformly sample m elements from them using $O(m)$ space.

For every vertex u, store t independent samples $v_{u,1}, v_{u,2}, \ldots, v_{u,t}$ of u's neighbors.

A simple algorithm

Reservoir Sampling

Given a stream of elements as input, one can uniformly sample m elements from them using $O(m)$ space.

For every vertex u, store t independent samples $v_{u,1}, v_{u,2}, \ldots, v_{u,t}$ of u's neighbors.

Perform a t-step random walk using these samples. After visiting u for the i-th time, go to $v_{u,i}$ in the next step.

A simple algorithm

Reservoir Sampling

Given a stream of elements as input, one can uniformly sample m elements from them using $O(m)$ space.

For every vertex u, store t independent samples $v_{u,1}, v_{u,2}, \ldots, v_{u,t}$ of u's neighbors.

Perform a t-step random walk using these samples. After visiting u for the i-th time, go to $v_{u,i}$ in the next step.

$O(nt)$ words of space. Perfect simulation ($\varepsilon = 0$)

A simple algorithm

Reservoir Sampling

Given a stream of elements as input, one can uniformly sample m elements from them using $O(m)$ space.

For every vertex u, store t independent samples $v_{u,1}, v_{u,2}, \ldots, v_{u,t}$ of u's neighbors.

Perform a t-step random walk using these samples. After visiting u for the i-th time, go to $v_{u,i}$ in the next step.

$O(nt)$ words of space. Perfect simulation ($\varepsilon = 0$)

Main questions

Can we do better (when small error $\varepsilon > 0$ is allowed)?

Can we prove space lower bounds?

Related work

In the multi-pass streaming model: Algorithm using $O(n)$ space and $O(\sqrt{t})$ passes. [Das Sarma, Gollapudi, Panigrahy, 2011]

Related work

In the multi-pass streaming model: Algorithm using $O(n)$ space and $O(\sqrt{t})$ passes. [Das Sarma, Gollapudi, Panigrahy, 2011]
Applications to estimating the page-rank vector, mixing time and conductance of graphs.

Related work

In the multi-pass streaming model: Algorithm using $O(n)$ space and $O(\sqrt{t})$ passes. [Das Sarma, Gollapudi, Panigrahy, 2011]
Applications to estimating the page-rank vector, mixing time and conductance of graphs.

Our study: What can we do in the single-pass streaming model?

Results

- On a directed graph, simulating a t-step random walk with error $\varepsilon \le 1/3$ requires $\Omega(nt\log(n/t))$ bits of memory. (for $t \le n/2$)

Results

- On a directed graph, simulating a t-step random walk with error $\varepsilon \leq 1/3$ requires $\Omega(nt \log(n/t))$ bits of memory. (for $t \leq n/2$)
- On an undirected graph, simulating a t-step random walk with error $\varepsilon \leq 1/3$ requires $\Omega(n\sqrt{t})$ bits of memory. (for $t = O(n^2)$)

Results

- On a directed graph, simulating a t-step random walk with error $\varepsilon \leq 1/3$ requires $\Omega(nt\log(n/t))$ bits of memory. (for $t \leq n/2$)
- On an undirected graph, simulating a t-step random walk with error $\varepsilon \leq 1/3$ requires $\Omega(n\sqrt{t})$ bits of memory. (for $t = O(n^2)$)
- On an undirected graph, we can simulate a t-step random walk using $O(n\sqrt{t})$ words of memory, with error $\varepsilon \leq 2^{-\Omega(\sqrt{t})}$.

Results

- On a directed graph, simulating a t-step random walk with error $\varepsilon \le 1/3$ requires $\Omega(nt\log(n/t))$ bits of memory. (for $t \le n/2$)
- On an undirected graph, simulating a t-step random walk with error $\varepsilon \le 1/3$ requires $\Omega(n\sqrt{t})$ bits of memory. (for $t = O(n^2)$)
- On an undirected graph, we can simulate a t-step random walk using $O(n\sqrt{t})$ words of memory, with error $\varepsilon \le 2^{-\Omega(\sqrt{t})}$.
  - For smaller ε, we use $O(n(\sqrt{t} + \frac{\log \varepsilon^{-1}}{\log\log \varepsilon^{-1}}))$ words of memory.

Results

- On a directed graph, simulating a t-step random walk with error $\varepsilon \le 1/3$ requires $\Omega(nt \log(n/t))$ bits of memory. (for $t \le n/2$)
- On an undirected graph, simulating a t-step random walk with error $\varepsilon \le 1/3$ requires $\Omega(n\sqrt{t})$ bits of memory. (for $t = O(n^2)$)
- On an undirected graph, we can simulate a t-step random walk using $O(n\sqrt{t})$ words of memory, with error $\varepsilon \le 2^{-\Omega(\sqrt{t})}$.
  - For smaller ε, we use $O(n(\sqrt{t} + \frac{\log \varepsilon^{-1}}{\log\log \varepsilon^{-1}}))$ words of memory.

Results

- On a directed graph, simulating a t-step random walk with error $\varepsilon \le 1/3$ requires $\Omega(nt\log(n/t))$ bits of memory. (for $t \le n/2$)
- On an undirected graph, simulating a t-step random walk with error $\varepsilon \le 1/3$ requires $\Omega(n\sqrt{t})$ bits of memory. (for $t = O(n^2)$)
- On an undirected graph, we can simulate a t-step random walk using $O(n\sqrt{t})$ words of memory, with error $\varepsilon \le 2^{-\Omega(\sqrt{t})}$.
  - For smaller ε, we use $O(n(\sqrt{t} + \frac{\log \varepsilon^{-1}}{\log\log \varepsilon^{-1}}))$ words of memory.

Nearly matching space lower bounds & upper bounds for both directed/undirected settings!

Results

- On a directed graph, simulating a t-step random walk with error $\varepsilon \le 1/3$ requires $\Omega(nt\log(n/t))$ bits of memory. (for $t \le n/2$)
- On an undirected graph, simulating a t-step random walk with error $\varepsilon \le 1/3$ requires $\Omega(n\sqrt{t})$ bits of memory. (for $t = O(n^2)$)
- On an undirected graph, we can simulate a t-step random walk using $O(n\sqrt{t})$ words of memory, with error $\varepsilon \le 2^{-\Omega(\sqrt{t})}$.
  - For smaller ε, we use $O(n(\sqrt{t} + \frac{\log \varepsilon^{-1}}{\log\log \varepsilon^{-1}}))$ words of memory.

Nearly matching space lower bounds & upper bounds for both directed/undirected settings!

Space lower bound for undirected graphs

We show a reduction from the INDEX problem.

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.

INDEX lower bound [Miltersen, Nisan, Safra, Wigderson, 1998]

For any constant $1/2 < c \leq 1$, solving the INDEX problem with success probability c requires sending $\Omega(n)$ bits.

Space lower bound for undirected graphs

We show a reduction from the INDEX problem.

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.

INDEX lower bound [Miltersen, Nisan, Safra, Wigderson, 1998]

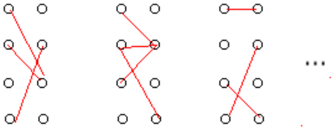
For any constant $1/2 < c \leq 1$, solving the INDEX problem with success probability c requires sending $\Omega(n)$ bits.

INDEX protocol

Alice creates a graph consisting of $\frac{n}{\sqrt{t}}$ disjoint groups. Each group is a bipartite graph with $\sqrt{t}$ vertices on each side.

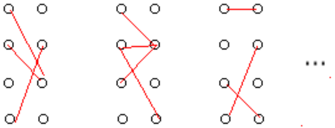
INDEX protocol

Alice creates a graph consisting of $\frac{n}{\sqrt{t}}$ disjoint groups. Each group is a bipartite graph with $\sqrt{t}$ vertices on each side.
By inserting edges into every groups, she can encode $\frac{n}{\sqrt{t}} \times \sqrt{t} \times \sqrt{t} = n\sqrt{t}$ bits of information.



INDEX protocol

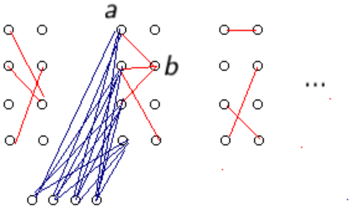
Alice creates a graph consisting of $\frac{n}{\sqrt{t}}$ disjoint groups. Each group is a bipartite graph with $\sqrt{t}$ vertices on each side.
By inserting edges into every groups, she can encode $\frac{n}{\sqrt{t}} \times \sqrt{t} \times \sqrt{t} = n\sqrt{t}$ bits of information.



Bob wants to see whether edge (a, b) exists.

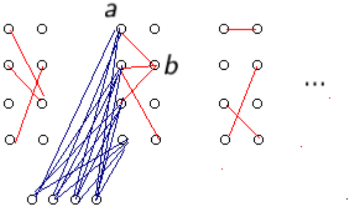
INDEX protocol

Alice creates a graph consisting of $\frac{n}{\sqrt{t}}$ disjoint groups. Each group is a bipartite graph with $\sqrt{t}$ vertices on each side.
By inserting edges into every groups, she can encode $\frac{n}{\sqrt{t}} \times \sqrt{t} \times \sqrt{t} = n\sqrt{t}$ bits of information.



Bob wants to see whether edge (a, b) exists.
Alice sends the memory of the streaming algorithm to Bob. Bob adds $\sqrt{t}$ vertices and connect each of them to every vertex in a's side.

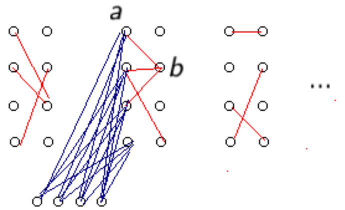
Space lower bound for undirected graphs



Bob wants to see whether edge (a, b) exists.
He adds $\sqrt{t}$ vertices and connects each of them to every vertex in a's side.

- Starting from a Bob's vertex, go to a w.p. $\frac{1}{\sqrt{t}}$, then go to b w.p. $\Theta(\frac{1}{\sqrt{t}})$

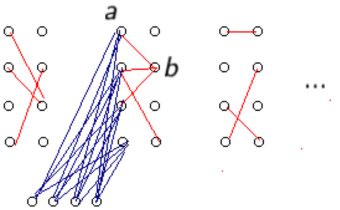
Space lower bound for undirected graphs



Bob wants to see whether edge (a, b) exists.
He adds $\sqrt{t}$ vertices and connects each of them to every vertex in a's side.

- Starting from a Bob's vertex, go to a w.p. $\frac{1}{\sqrt{t}}$, then go to b w.p. $\Theta(\frac{1}{\sqrt{t}})$
- visit a Bob's vertex every $O(1)$ steps with good probability

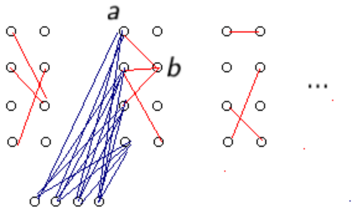
Space lower bound for undirected graphs



Bob wants to see whether edge (a, b) exists.
He adds $\sqrt{t}$ vertices and connects each of them to every vertex in a's side.

- Starting from a Bob's vertex, go to a w.p. $\frac{1}{\sqrt{t}}$, then go to b w.p. $\Theta(\frac{1}{\sqrt{t}})$
- visit a Bob's vertex every $O(1)$ steps with good probability

Space lower bound for undirected graphs

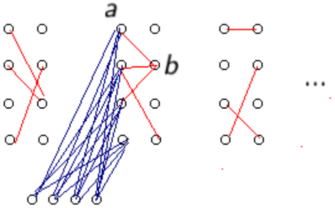


Bob wants to see whether edge (a, b) exists.
He adds $\sqrt{t}$ vertices and connects each of them to every vertex in a's side.

- Starting from a Bob's vertex, go to a w.p. $\frac{1}{\sqrt{t}}$, then go to b w.p. $\Theta(\frac{1}{\sqrt{t}})$
- visit a Bob's vertex every $O(1)$ steps with good probability

Edge (a, b) is likely to be visited (if exists) after $O(t)$ steps. So bob can tell whether (a, b) exists by simulating the generated $O(t)$-step random walk.

Space lower bound for undirected graphs



Bob wants to see whether edge (a, b) exists.
He adds $\sqrt{t}$ vertices and connects each of them to every vertex in a's side.

- Starting from a Bob's vertex, go to a w.p. $\frac{1}{\sqrt{t}}$, then go to b w.p. $\Theta(\frac{1}{\sqrt{t}})$
- visit a Bob's vertex every $O(1)$ steps with good probability

Edge (a, b) is likely to be visited (if exists) after $O(t)$ steps. So bob can tell whether (a, b) exists by simulating the generated $O(t)$-step random walk.
By INDEX lower bound, we need $\Omega(n\sqrt{t})$ bits of space.

Algorithm for undirected graphs

(For now we assume there are no multiple edges or self-loops)

- Small vertices: degree $\leq C$
- Big vertices: degree $> C$

for some parameter $C \approx \sqrt{t}$.

Algorithm for undirected graphs

(For now we assume there are no multiple edges or self-loops)

- Small vertices: degree $\leq C$
- Big vertices: degree $> C$

for some parameter $C \approx \sqrt{t}$.
For every small vertex u: store all neighbors of u.

Algorithm for undirected graphs

(For now we assume there are no multiple edges or self-loops)

- Small vertices: degree $\leq C$
- Big vertices: degree $> C$

for some parameter $C \approx \sqrt{t}$.
For every small vertex u: store all neighbors of u.
For every big vertex u: store C independent samples of u's big neighbors.

Algorithm for undirected graphs

(For now we assume there are no multiple edges or self-loops)

- Small vertices: degree $\leq C$
- Big vertices: degree $> C$

for some parameter $C \approx \sqrt{t}$.
For every small vertex u: store all neighbors of u.
For every big vertex u: store C independent samples of u's big neighbors.
Total space: $O(n\sqrt{t})$ words.

Algorithm for undirected graphs using $O(n\sqrt{t})$ space

For every small vertex u: store all neighbors of u.
For every big vertex u: store C independent samples of u's big neighbors.

How to simulate a random walk? (Suppose we are now at vertex u)

Algorithm for undirected graphs using $O(n\sqrt{t})$ space

For every small vertex u: store all neighbors of u.
For every big vertex u: store C independent samples of u's big neighbors.

How to simulate a random walk? (Suppose we are now at vertex u)

- If u is small: simply pick a random neighbor v as the next vertex

Algorithm for undirected graphs using $O(n\sqrt{t})$ space

For every small vertex u: store all neighbors of u.

For every big vertex u: store C independent samples of u's big neighbors.

How to simulate a random walk? (Suppose we are now at vertex u)

- If u is small: simply pick a random neighbor v as the next vertex
- If u is big: flip a biased coin to decide if the next vertex will be big/small

Algorithm for undirected graphs using $O(n\sqrt{t})$ space

For every small vertex u: store all neighbors of u.

For every big vertex u: store C independent samples of u's big neighbors.

How to simulate a random walk? (Suppose we are now at vertex u)

- If u is small: simply pick a random neighbor v as the next vertex
- If u is big: flip a biased coin to decide if the next vertex will be big/small
  - next vertex is small: pick a random small neighbor (we know all of them!)

Algorithm for undirected graphs using $O(n\sqrt{t})$ space

For every small vertex u: store all neighbors of u.

For every big vertex u: store C independent samples of u's big neighbors.

How to simulate a random walk? (Suppose we are now at vertex u)

- If u is small: simply pick a random neighbor v as the next vertex
- If u is big: flip a biased coin to decide if the next vertex will be big/small
  - next vertex is small: pick a random small neighbor (we know all of them!)
  - next vertex is big: have to consume a sample of u's big neighbor. (FAIL if all have been used)

Algorithm for undirected graphs using $O(n\sqrt{t})$ space

For every small vertex u: store all neighbors of u.

For every big vertex u: store C independent samples of u's big neighbors.

How to simulate a random walk? (Suppose we are now at vertex u)

- If u is small: simply pick a random neighbor v as the next vertex
- If u is big: flip a biased coin to decide if the next vertex will be big/small
  - next vertex is small: pick a random small neighbor (we know all of them!)
  - next vertex is big: have to consume a sample of u's big neighbor. (FAIL if all have been used)

Algorithm for undirected graphs using $O(n\sqrt{t})$ space

For every small vertex u: store all neighbors of u.

For every big vertex u: store C independent samples of u's big neighbors.

How to simulate a random walk? (Suppose we are now at vertex u)

- If u is small: simply pick a random neighbor v as the next vertex
- If u is big: flip a biased coin to decide if the next vertex will be big/small
  - next vertex is small: pick a random small neighbor (we know all of them!)
  - next vertex is big: have to consume a sample of u's big neighbor. (FAIL if all have been used)

If we can make Pr[FAIL]$\leq \varepsilon$, then our output distribution will be (2ε)-close.

Analysis of failure probability

We say a vertex u fails, if the failure happens when we are at u.

Analysis of failure probability

We say a vertex u fails, if the failure happens when we are at u.
$\Pr[\text{FAIL}] \leq \sum_u \Pr[u \text{ fails}]$

Analysis of failure probability

We say a vertex u fails, if the failure happens when we are at u.
$\Pr[\text{FAIL}] \le \sum_u \Pr[u \text{ fails}]$
u fails only if the number of "$u \to \text{Big}$" steps exceeds C.

Analysis of failure probability

We say a vertex u fails, if the failure happens when we are at u.
$\Pr[\text{FAIL}] \le \sum_u \Pr[u \text{ fails}]$
u fails only if the number of "$u \to$ Big" steps exceeds C.

$$\Pr[u \text{ fails}] \le \sum_{\text{walk } w} \Pr[w] \cdot 1[w \text{ has more than } C \text{ "}u \to \text{Big" steps}]$$

$$\text{where } \Pr[(v_0, v_1, \ldots, v_t)] = \frac{1}{d(v_0)d(v_1)\ldots d(v_{t-1})}$$

$$= \Pr[(v_t, \ldots, v_0)] \frac{d(v_t)}{d(v_0)}$$

Analysis of failure probability

We say a vertex u fails, if the failure happens when we are at u.
$\Pr[\text{FAIL}] \le \sum_u \Pr[u \text{ fails}]$
u fails only if the number of "$u \to$ Big" steps exceeds C.

$$\Pr[u \text{ fails}] \le \sum_{\text{walk } w} \Pr[w] \cdot 1[w \text{ has more than } C \text{ "}u \to \text{Big" steps}]$$

$$\text{where } \Pr[(v_0, v_1, \ldots, v_t)] = \frac{1}{d(v_0)d(v_1)\ldots d(v_{t-1})}$$

$$= \Pr[(v_t, \ldots, v_0)] \frac{d(v_t)}{d(v_0)}$$

The reversed walk $(v_t, \ldots, v_0)$ is still a walk (since the graph is undirected).

Analysis of failure probability

We say a vertex u fails, if the failure happens when we are at u.
$\Pr[\text{FAIL}] \le \sum_u \Pr[u \text{ fails}]$
u fails only if the number of "$u \to$ Big" steps exceeds C.

$$\Pr[u \text{ fails}] \le \sum_{\text{walk } w} \Pr[w] \cdot \mathbb{1}[w \text{ has more than } C \text{ "} u \to \text{Big" steps}]$$

where
$$\Pr[(v_0, v_1, \ldots, v_t)] = \frac{1}{d(v_0)d(v_1)\ldots d(v_{t-1})}$$
$$= \Pr[(v_t, \ldots, v_0)]\frac{d(v_t)}{d(v_0)}$$

The reversed walk $(v_t, \ldots, v_0)$ is still a walk (since the graph is undirected).
"$u \to$ Big" steps becomes "Big $\to u$" steps!

Algorithm for undirected graphs

Hence

$$\Pr[u \text{ fails}] \le n \sum_{\text{walk } w} \Pr[w] \cdot 1[w \text{ has more than } C \text{ ``Big} \to u\text{'' steps}]$$

Algorithm for undirected graphs

Hence

$$\Pr[u \text{ fails}] \leq n \sum_{\text{walk } w} \Pr[w] \cdot 1[w \text{ has more than } C \text{ ``Big} \to u\text{'' steps}]$$

Big vertex has degree $> C$. $\Pr\left[v_i \to v_{i+1} \text{ is a ``Big} \to u\text{'' step} \mid v_i\right] < 1/C$
In t steps, "Big $\to u$" happens $< t/C$ times in expectation (and it has concentration!)
Choosing $C = O(\sqrt{t} \log n\varepsilon^{-1})$ makes $\Pr[\text{FAIL}] \leq \sum_u \Pr[u \text{ fails}] \ll \varepsilon$.

(we can improve the log-factor using more careful analysis..)

Dealing with multiple edges

When there are multiple edges,

$$\Pr\left[v_i \to v_{i+1} \text{ is a ``Big} \to u\text{'' step} \mid v_i\right] < 1/C$$

might not hold. (example: most of v's adjacent edges are connecting u)

Fix: For each u, we use heavy-hitter algorithms to find those neighbors v such that $\Pr[v_i = v \mid v_{i-1} = u] > 1/C$.

Thank you!