Densest Subgraph in Dynamic Graph Streams

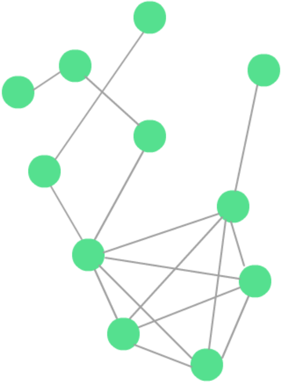
Andrew McGregor, David Tench, Sofya Vorotnikova, Hoa Vu

University of Massachusetts, Amherst

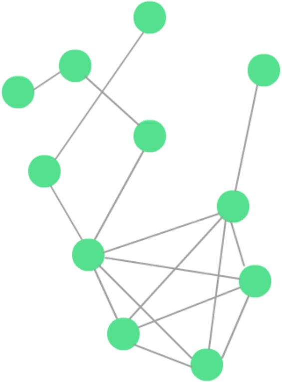
dtench@cs.umass.edu

MFCS 2015

The Densest Subgraph Problem

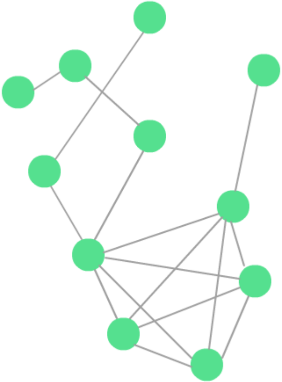


The Densest Subgraph Problem



Given a graph G = (V,E), find the node-induced subgraph that maximizes the ratio of edges to nodes.

The Densest Subgraph Problem



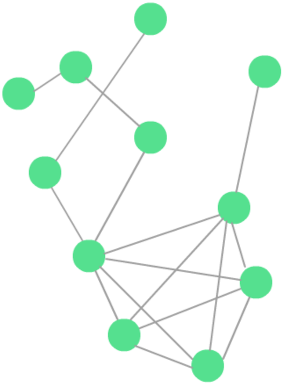
Given a graph G = (V,E), find the node-induced subgraph that maximizes the ratio of edges to nodes.

$$d^* = \max_{U \subseteq V} d_U$$

where

$$d_U = \frac{\text{\# of edges in subgraph induced by U}}{|U|}.$$

The Densest Subgraph Problem



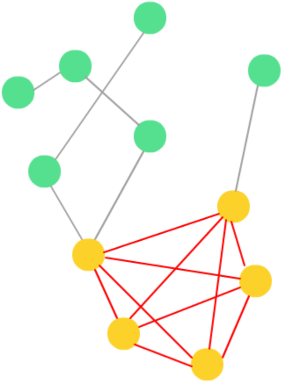
Given a graph G = (V,E), find the node-induced subgraph that maximizes the ratio of edges to nodes.

$$d^* = \max_{U \subseteq V} d_U$$

where

$$d_U = \frac{\text{\# of edges in subgraph induced by U}}{|U|}.$$

The Densest Subgraph Problem



Given a graph G = (V,E), find the node-induced subgraph that maximizes the ratio of edges to nodes.

$$d^* = \max_{U \subseteq V} d_U$$

where
$$d_U = \frac{\text{\# of edges in subgraph induced by U}}{|U|}.$$

The Streaming Domain

The Streaming Domain

Imagine that our input graph is very, very large. We have enough space to store the node list but not much more.

The Streaming Domain

Imagine that our input graph is very, very large. We have enough space to store the node list but not much more.

- We only have n polylog(n) space

The Streaming Domain

Imagine that our input graph is very, very large. We have enough space to store the node list but not much more.

- We only have n polylog(n) space
- G might have as many as $\Theta(n^2)$ edges

The Streaming Domain

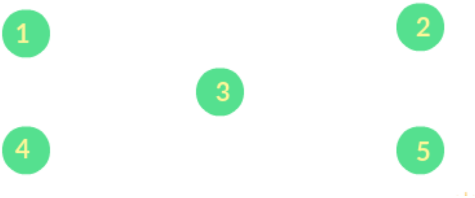
Imagine that our input graph is very, very large. We have enough space to store the node list but not much more.

- We only have n polylog(n) space
- G might have as many as $\Theta(n^2)$ edges
- G is defined by a *stream* of edge insertions and deletions

The Streaming Domain

Imagine that our input graph is very, very large. We have enough space to store the node list but not much more.

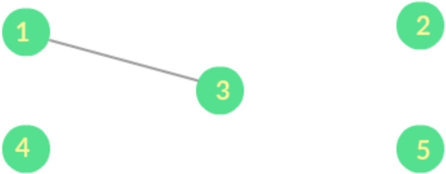
- We only have n polylog(n) space
- G might have as many as $\Theta(n^2)$ edges
- G is defined by a *stream* of edge insertions and deletions
  Stream:



The Streaming Domain

Imagine that our input graph is very, very large. We have enough space to store the node list but not much more.

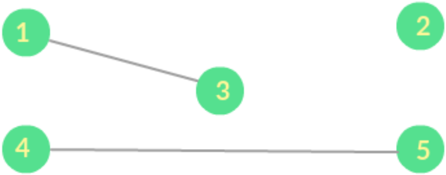
- We only have n polylog(n) space
- G might have as many as $\Theta(n^2)$ edges
- G is defined by a *stream* of edge insertions and deletions
  Stream: (insert,1,3),



The Streaming Domain

Imagine that our input graph is very, very large. We have enough space to store the node list but not much more.

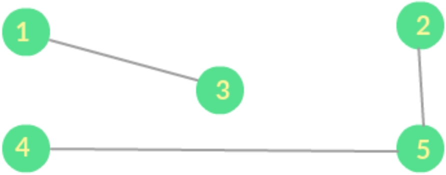
- We only have n polylog(n) space
- G might have as many as $\Theta(n^2)$ edges
- G is defined by a *stream* of edge insertions and deletions
  Stream: (insert,1,3), (insert,4,5),



The Streaming Domain

Imagine that our input graph is very, very large. We have enough space to store the node list but not much more.

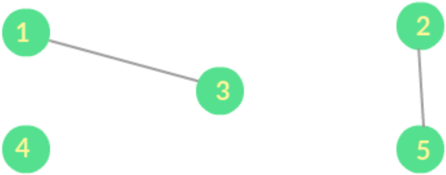
- We only have n polylog(n) space
- G might have as many as $\Theta(n^2)$ edges
- G is defined by a *stream* of edge insertions and deletions
  Stream: (insert,1,3), (insert,4,5), (insert,2,5),



The Streaming Domain

Imagine that our input graph is very, very large. We have enough space to store the node list but not much more.

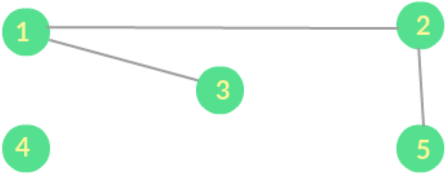
- We only have n polylog(n) space
- G might have as many as $\Theta(n^2)$ edges
- G is defined by a *stream* of edge insertions and deletions
  Stream: (insert,1,3), (insert,4,5), (insert,2,5), (delete,4,5)



The Streaming Domain

Imagine that our input graph is very, very large. We have enough space to store the node list but not much more.

- We only have n polylog(n) space
- G might have as many as $\Theta(n^2)$ edges
- G is defined by a *stream* of edge insertions and deletions
  Stream: (insert,1,3), (insert,4,5), (insert,2,5), (delete,4,5)
  (insert,1,2)



Densest Subgraph in Streaming

Our task is to approximate the maximum density subgraph of the graph defined by the input stream.

Densest Subgraph in Streaming

Our task is to approximate the maximum density subgraph of the graph defined by the input stream.

- Can't store entire stream

Densest Subgraph in Streaming

Our task is to approximate the maximum density subgraph of the graph defined by the input stream.

- Can't store entire stream
- Must discard some information

Densest Subgraph in Streaming

Our task is to approximate the maximum density subgraph of the graph defined by the input stream.

- Can't store entire stream
- Must discard some information
- Prior work:
  - Bahmani et al. (PVLDB 2012) have a $(2 + \epsilon)$-approximation that requires log(n) passes over the stream

Densest Subgraph in Streaming

Our task is to approximate the maximum density subgraph of the graph defined by the input stream.

- Can't store entire stream
- Must discard some information
- Prior work:
  - Bahmani et al. (PVLDB 2012) have a $(2+\epsilon)$-approximation that requires log(n) passes over the stream
  - Bhattycharya et al. (STOC 2015) only require 1 pass

Densest Subgraph in Streaming

Our result:

Densest Subgraph in Streaming

Our result:

- $(1+\epsilon)$-approximation to the densest subgraph problem

Densest Subgraph in Streaming

Our result:

- $(1+\epsilon)$-approximation to the densest subgraph problem
- n polylog(n) space

Densest Subgraph in Streaming

Our result:

- $(1 + \epsilon)$-approximation to the densest subgraph problem
- n polylog(n) space
- polylog(n) time per update and poly(n) post-processing time

Densest Subgraph in Streaming

Our result:

- $(1 + \epsilon)$-approximation to the densest subgraph problem
- n polylog(n) space
- polylog(n) time per update and poly(n) post-processing time
- Requires a single pass over the input stream

Densest Subgraph in Streaming

Our approach:

Edge sampling technique

Approximately preserves max density

Remaining graph has n polylog(n) edges

This can be done in streaming

ℓ_0-sampling allows emulation of edge sampling

Naive implementation is slow, but improvable

Edge Sampling Preserves Max Density

Edge Sampling Preserves Max Density

Sample each edge in G = (V,E) with probability

$$p \approx \epsilon^{-2} \log(n) \frac{n}{m}$$

where $n = |V|$ and $m = |E|$. Call the resulting graph G'.

Edge Sampling Preserves Max Density

Sample each edge in G = (V,E) with probability

$$p \approx \epsilon^{-2} \log(n) \frac{n}{m}$$

where $n = |V|$ and $m = |E|$. Call the resulting graph G'.

Expected # of edges in G':

$$mp = O(\epsilon^{-2} n \log(n))$$

Edge Sampling Preserves Max Density

Sample each edge in G = (V,E) with probability

$$p \approx \epsilon^{-2} \log(n) \frac{n}{m}$$

where $n = |V|$ and $m = |E|$. Call the resulting graph G'.

Expected # of edges in G':

$$mp = O(\epsilon^{-2} n \log(n))$$

Sampling Theorem

G' can be used to approximate d^* (max density of G) up to a factor $(1 + \epsilon)$.

Proof of Sampling Theorem: Preliminaries

Proof of Sampling Theorem: Preliminaries

For any $U \subseteq V$:

$$d_U = \frac{\text{\# of edges in subgraph of G induced by U}}{|U|}$$

$$\tilde{d}_U = \frac{1}{p}\frac{\text{\# of edges in subgraph of G' induced by U}}{|U|}$$

Proof of Sampling Theorem: Preliminaries

For any $U \subseteq V$:

$$d_U = \frac{\text{\# of edges in subgraph of G induced by U}}{|U|}$$

$$\tilde{d}_U = \frac{1}{p} \frac{\text{\# of edges in subgraph of G' induced by U}}{|U|}$$

We want to show that

$$(1-\epsilon)d^* \leq \max_U \tilde{d}_U \leq (1+\epsilon)d^*$$

Proof of Sampling Theorem: Preliminaries

Proof of Sampling Theorem: Preliminaries

Pick some $U \subseteq V$ of size k. By Chernoff, with probability $1 - n^{-9k}$,

<u>Low Density Case</u>

$$\text{if } d_U \leq \frac{d^*}{60} \text{ then } \tilde{d}_U \leq \frac{d^*}{10}$$

<u>High Density Case</u>

$$\text{if } d_U > \frac{d^*}{60} \text{ then } \tilde{d}_U \approx (1 \pm \epsilon) d_U$$

Proof of Sampling Theorem: Preliminaries

Pick some $U \subseteq V$ of size k. By Chernoff, with probability $1 - n^{-9k}$,

Low Density Case

$$\text{if } d_U \leq \frac{d^*}{60} \text{ then } \tilde{d}_U \leq \frac{d^*}{10}$$

High Density Case

$$\text{if } d_U > \frac{d^*}{60} \text{ then } \tilde{d}_U \approx (1 \pm \epsilon) d_U$$

By union bound over all U of size k, and then by all k, the above holds for all $U \subseteq V$ whp.

Proof of Sampling Theorem: Lower Bound

Proof of Sampling Theorem: Lower Bound

Let $U^* = arg\max_U d_U$. Then, since $d_{U^*} = d^* > \frac{d^*}{60}$,

$$\tilde{d_{U^*}} \geq (1-\epsilon)d^*.$$

Proof of Sampling Theorem: Lower Bound

Let $U^* = arg\max_U d_U$. Then, since $d_{U^*} = d^* > \frac{d^*}{60}$,

$$\tilde{d_{U^*}} \geq (1 - \epsilon)d^*.$$

Thus

$$\max_U \tilde{d_U} \geq d_{U^*} \geq (1 - \epsilon)d^*.$$

Proof of Sampling Theorem: Upper Bound

$$\max_{U} \tilde{d}_U$$

Proof of Sampling Theorem: Upper Bound

$$\max_U \tilde{d}_U = max\{\max_{U:d_U \leq \frac{d^*}{60}} \tilde{d}_U, \max_{U:d_U > \frac{d^*}{60}} \tilde{d}_U\}$$

Proof of Sampling Theorem: Upper Bound

$$\max_U \tilde{d}_U = max\{\max_{U:d_U \leq \frac{d^*}{60}} \tilde{d}_U, \max_{U:d_U > \frac{d^*}{60}} \tilde{d}_U\}$$
$$\leq max\{\frac{d^*}{10}, (1+\epsilon)d_U\}$$

Proof of Sampling Theorem: Upper Bound

$$
\begin{aligned}
\max_U \tilde{d}_U &= max\{\max_{U:d_U \le \frac{d^*}{60}} \tilde{d}_U, \max_{U:d_U > \frac{d^*}{60}} \tilde{d}_U\} \\
&\le max\{\frac{d^*}{10}, (1+\epsilon)d_U\} \\
&= max\{\frac{d^*}{10}, (1+\epsilon)d^*\} = (1+\epsilon)d^*
\end{aligned}
$$

Proof of Sampling Theorem: Upper Bound

$$\begin{aligned}
\max_U \tilde{d}_U &= max\{\max_{U:d_U \leq \frac{d^*}{60}} \tilde{d}_U, \max_{U:d_U > \frac{d^*}{60}} \tilde{d}_U\} \\
&\leq max\{\frac{d^*}{10}, (1+\epsilon)d_U\} \\
&= max\{\frac{d^*}{10}, (1+\epsilon)d^*\} = (1+\epsilon)d^*
\end{aligned}$$

Thus, putting together the upper and lower bounds,

$$(1-\epsilon)d^* \leq \max_U \tilde{d}_U \leq (1+\epsilon)d^*.$$

Densest Subgraph in Streaming

Edge sampling technique

Approximately preserves max density

Remaining graph has n polylog(n) edges

This can be done in streaming

ℓ_0-sampling allows emulation of edge sampling

Naive implementation is slow, but improvable

Implementation in the Streaming Setting

Implementation in the Streaming Setting

We can solve the problem if we can sample the edges of the stream with probability $p \approx \epsilon^{-2} log(n) \frac{n}{m}$. However, there are two challenges:

Implementation in the Streaming Setting

We can solve the problem if we can sample the edges of the stream with probability $p \approx \epsilon^{-2} log(n) \frac{n}{m}$. However, there are two challenges:

- Edges we sample during the stream may be deleted later

Implementation in the Streaming Setting

We can solve the problem if we can sample the edges of the stream with probability $p \approx \epsilon^{-2} log(n) \frac{n}{m}$. However, there are two challenges:

- Edges we sample during the stream may be deleted later
- p depends on m, inaccessible until end of stream

Implementation in the Streaming Setting

Implementation in the Streaming Setting

- Edges we sample during the stream may be deleted later

Implementation in the Streaming Setting

- Edges we sample during the stream may be deleted later

Luckily, we can use ℓ_0-sampling to handle this. Using polylog(n) space and update time, we can return a random edge from G at the end of the stream.

Implementation in the Streaming Setting

- Edges we sample during the stream may be deleted later

Luckily, we can use ℓ_0-sampling to handle this. Using polylog(n) space and update time, we can return a random edge from G at the end of the stream.

- p depends on m, inaccessible until end of stream

Implementation in the Streaming Setting

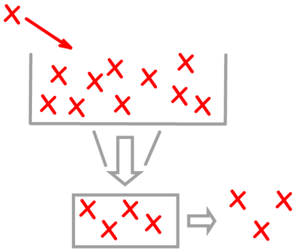
- Edges we sample during the stream may be deleted later

Luckily, we can use ℓ_0-sampling to handle this. Using polylog(n) space and update time, we can return a random edge from G at the end of the stream.

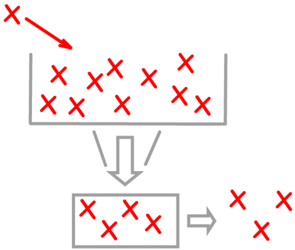
- p depends on m, inaccessible until end of stream

We sample $r \gg mp$ edges, and when the stream is over randomly choose X $\sim$ Bin(m,p) those edges without replacement.

Implementation in the Streaming Setting

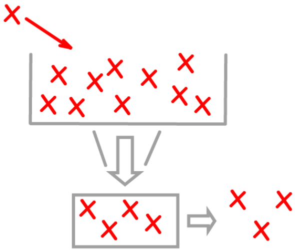


Implementation in the Streaming Setting



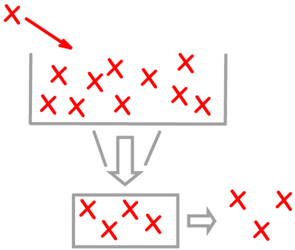
- Edge updates appear in stream

Implementation in the Streaming Setting



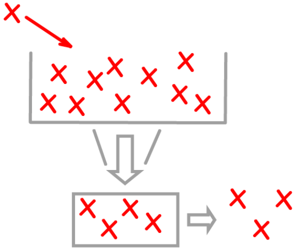
- Edge updates appear in stream
- ℓ_0-samplers project this information into a smaller space

Implementation in the Streaming Setting



- Edge updates appear in stream
- ℓ_0-samplers project this information into a smaller space
- After the stream, we query the samplers to get the right number of edges

Implementation in the Streaming Setting



- Edge updates appear in stream
- ℓ_0-samplers project this information into a smaller space
- After the stream, we query the samplers to get the right number of edges

Unfortunately, this takes $\Omega(n)$ time per update!

Why is the update time slow?

Why is the update time slow?

- ℓ_0-sampling maintains a projection of the edge set defined by the input stream

Why is the update time slow?

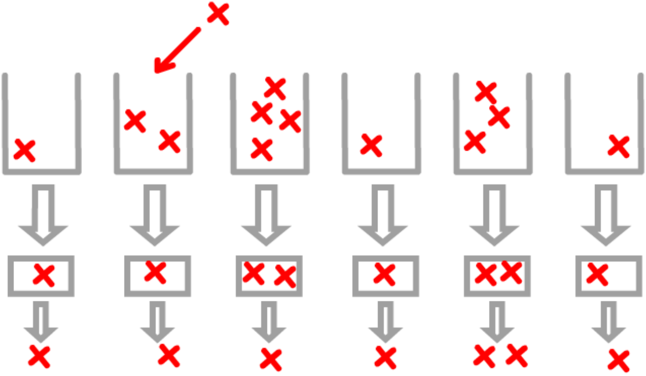
- ℓ_0-sampling maintains a projection of the edge set defined by the input stream
- Each time a new edge arrives, we must update every ℓ_0-sampler

Why is the update time slow?

- ℓ_0-sampling maintains a projection of the edge set defined by the input stream
- Each time a new edge arrives, we must update every ℓ_0-sampler
- We have more than n ℓ_0-samplers, and updating each takes $O(\mathrm{polylog}\, n)$ time

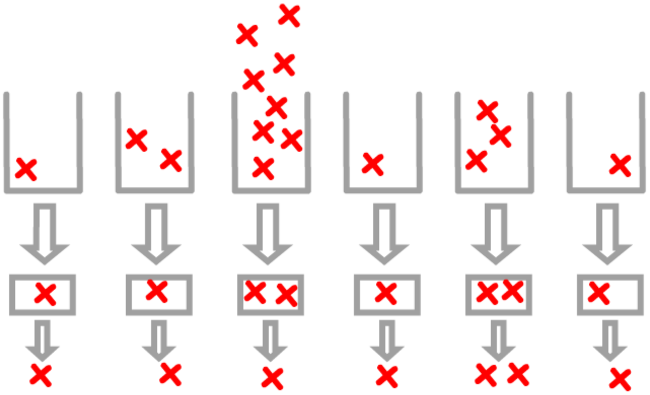
The Solution

Using a hash function, randomly partition the edge set into $\Theta(n)$ buckets. Maintain only log(n) ℓ_0-samplers for the edges in each group. When a new edge arrives in the stream, you only need to update the ℓ_0-samplers for its group!

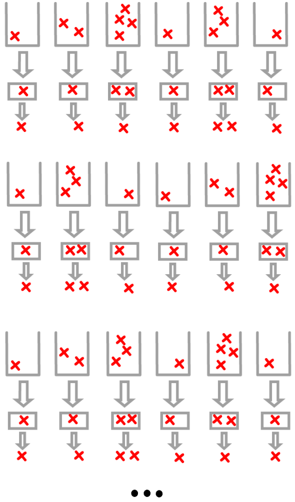


Overflowing Buckets

Problem: Some buckets might get too full. If that happens, we can't sample those edges properly.

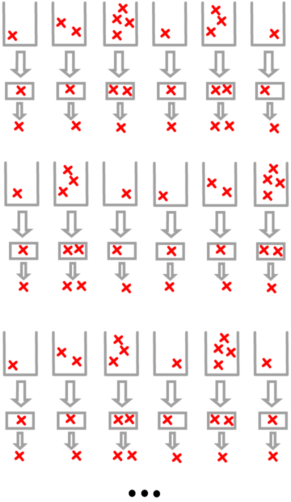


Solution: More Buckets



So we repeat the process in parallel with different partitions log(n) times. With high probability, each edge will end up in some sufficiently small group, so it can be sampled properly.

Solution: More Buckets



So we repeat the process in parallel with different partitions log(n) times. With high probability, each edge will end up in some sufficiently small group, so it can be sampled properly.

For each edge update: $\log(n)$ partitions each with $\log(n)$ ℓ_0-samplers each with polylog(n) update time yields polylog(n) update time.

Densest Subgraph in Streaming

Edge sampling technique

Approximately preserves max density

Remaining graph has n polylog(n) edges

This can be done in streaming

ℓ_0-sampling allows emulation of edge sampling

Naive implementation is slow, but improvable

Grazie!