# Streaming Graph Mining

Mauro Sozio

Telecom ParisTech

November 15, 2018

# Large-Scale Dynamic Data

Examples:

- The Web, with approximately 5 billion Web Pages;
- Models of the Human brain, approx. $10^{10}$ neurons;
- Twitter, highly dynamic, 6000 tweets per second;
- Facebook: 1.86 billion monthly active users.

To process the sheer amount of data, several models of computations have emerged...

# Models of Computations

- *streaming algorithms* The input is defined by a stream of data (e.g. edges/nodes in a graph). Algorithms in this model must process the input stream in the order it arrives (sequential access) while using only limited amount of memory. Semi-streaming algorithms are allowed to make multiple passes over the data. See [4, 5, 6].
- *dynamic algorithms* The main goal is to support query and update operations (e.g. remove an edge or update its weight) as quickly as possible, in particular, much faster than recomputing from scratch. Amortized analysis is used to analyze an algorithm: total worst case time / number of update operations. The input might or might not fit into memory. See [8, 7].
- *MapReduce Model* The input is partitioned into different machines, with each machine processing its chunk in parallel. The results are then aggregated. This can be iterated multiple times, typically constant or logarithmic in the size of the input. See [9, 10].

# Streaming Algorithm for Densest Subgraph

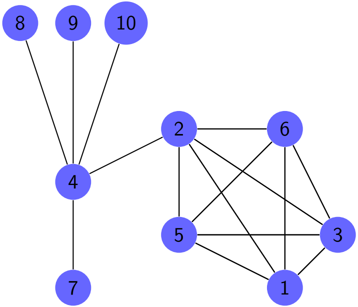
**Require**: an undirected graph $G$, a value $\epsilon > 0$

$H = G$;
while ($G$ contains at least one edge)

- rem. all nodes $v$ (and their edges) with $\delta_G(v) \leq 2(1+\epsilon)\rho(G)$ from $G$.
- if $\rho(G) > \rho(H)$ then $H \leftarrow G$;

return $H$;

# Faster algorithm: Example

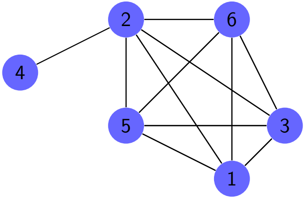



$\epsilon = 0.1$

**Iteration 1:**

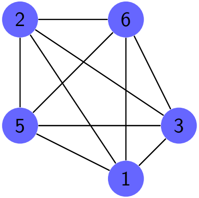
$\rho(G) = \frac{16}{10}$, remove nodes with degree $\leq 2 * (1.1) * 1.6 = 3.52$.

# Faster algorithm: Example




$\epsilon = 0.1$
**Iteration 2:**
$\rho(G) = \frac{11}{6}$, remove nodes with degree $\leq 2 * (1.1) * \frac{11}{6} = 3.45$.

# Faster algorithm: Example




$\epsilon = 0.1$

**Iteration 3:**

$\rho(G) = \frac{10}{5}$, remove nodes with degree $\leq 2 * (1.1) * 2 = 4.4$.

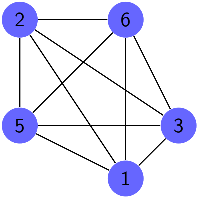
# Faster algorithm: Example

$\epsilon = 0.1$

**Iteration 4:**

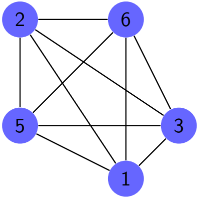
Empty Graph.

# Faster algorithm: Example




$\epsilon = 0.1$
$2(1+\epsilon)-$**Approx. Densest Subgraph!**

# Faster algorithm: Example




$\epsilon = 0.1$
$2(1+\epsilon)-$**Approx. Densest Subgraph!**
What if $\epsilon$ is large? (say $\epsilon = 0.5$)

# Approx. guarantee of the fast algo

## Theorem 1

*Let $O = (V_O, E_O)$ be a densest subgraph and let $H = (V_H, E_H)$ be the subgraph found by our algo, with parameter $\epsilon > 0$. Then, $\rho(H) \geq \frac{\rho(O)}{2(1+\epsilon)}$.*

## Proof.

Let $O = (V_O, E_O)$ be a densest subgraph. Consider the first step $t$ in the algo such that we remove a node $v \in V_O$ from the current graph $G_t$ (there must be such a step). From Lemma **??**, $\delta_{G_t}(v) \geq \delta_O(v) \geq \rho(O)$. Hence,

$$\rho(O) \leq \delta_{G_t}(v)$$

# Approx. guarantee of the fast algo

## Theorem 1

*Let $O = (V_O, E_O)$ be a densest subgraph and let $H = (V_H, E_H)$ be the subgraph found by our algo, with parameter $\epsilon > 0$. Then, $\rho(H) \geq \frac{\rho(O)}{2(1+\epsilon)}$.*

## Proof.

Let $O = (V_O, E_O)$ be a densest subgraph. Consider the first step $t$ in the algo such that we remove a node $v \in V_O$ from the current graph $G_t$ (there must be such a step). From Lemma **??**, $\delta_{G_t}(v) \geq \delta_O(v) \geq \rho(O)$. Hence,

$$\begin{aligned}\rho(O) &\leq \delta_{G_t}(v) \\ &\leq 2(1+\epsilon)\rho(G_t) \\ &\leq 2(1+\epsilon)\rho(H)\end{aligned}$$

# Running time of the fast algo

## Theorem 2

*The number of iterations of the fast algo with input* $G = (V_G, E_G)$ *and* $\epsilon > 0$ *is at most* $\lceil \log_{1+\epsilon}(|V_G|) \rceil$.

## Proof.

Consider any step $t$ of the algo and let $G_t = (V_{G_t}, E_{G_t})$ be the subgraph at the beginning of that step. Let $R_t$ be the set of nodes removed at the end of such step, i.e. the degree of any node in $R_t$ is $\leq 2(1+\epsilon)\rho(G_t)$. Then,

$$2|E_{G_t}| = \sum_{v \in R_t} \delta_{G_t}(v) + \sum_{v \in V_{G_t} \setminus R_t} \delta_{G_t}(v)$$

# Running time of the fast algo

## Theorem 2

*The number of iterations of the fast algo with input $G = (V_G, E_G)$ and $\epsilon > 0$ is at most $\lceil \log_{1+\epsilon}(|V_G|) \rceil$.*

## Proof.

Consider any step $t$ of the algo and let $G_t = (V_{G_t}, E_{G_t})$ be the subgraph at the beginning of that step. Let $R_t$ be the set of nodes removed at the end of such step, i.e. the degree of any node in $R_t$ is $\leq 2(1+\epsilon)\rho(G_t)$. Then,

$$\begin{aligned} 2|E_{G_t}| &= \sum_{v \in R_t} \delta_{G_t}(v) + \sum_{v \in V_{G_t} \setminus R_t} \delta_{G_t}(v) \\ &> 2(1+\epsilon)(|V_{G_t}| - |R_t|)\rho(G_t) \end{aligned}$$

# Running time of the fast algo

## Theorem 2

*The number of iterations of the fast algo with input $G = (V_G, E_G)$ and $\epsilon > 0$ is at most $\lceil \log_{1+\epsilon}(|V_G|) \rceil$.*

## Proof.

Consider any step $t$ of the algo and let $G_t = (V_{G_t}, E_{G_t})$ be the subgraph at the beginning of that step. Let $R_t$ be the set of nodes removed at the end of such step, i.e. the degree of any node in $R_t$ is $\leq 2(1+\epsilon)\rho(G_t)$. Then,

$$
\begin{aligned}
2|E_{G_t}| &= \sum_{v \in R_t} \delta_{G_t}(v) + \sum_{v \in V_{G_t} \setminus R_t} \delta_{G_t}(v) \\
&> 2(1+\epsilon)(|V_{G_t}| - |R_t|)\rho(G_t) \\
&= 2(1+\epsilon)(|V_{G_t}| - |R_t|)\frac{|E_{G_t}|}{|V_{G_t}|}.
\end{aligned}
$$

# Running time of the fast algo...

## Proof.

Then,

# Running time of the fast algo...

## Proof.

Then,

$$2|E_{G_t}| > 2(1+\epsilon)(|V_{G_t}| - |R_t|)\frac{|E_{G_t}|}{|V_{G_t}|}, \qquad \Leftrightarrow$$

□

# Running time of the fast algo...

## Proof.

Then,

$$
\begin{aligned}
2|E_{G_t}| > 2(1+\epsilon)(|V_{G_t}| - |R_t|)\frac{|E_{G_t}|}{|V_{G_t}|}, &\qquad \Leftrightarrow \\
|V_{G_t}| > (1+\epsilon)(|V_{G_t}| - |R_t|), &\qquad \Leftrightarrow
\end{aligned}
$$

$\square$

# Running time of the fast algo...

## Proof.

Then,

$$
\begin{aligned}
2|E_{G_t}| > 2(1+\epsilon)(|V_{G_t}| - |R_t|)\frac{|E_{G_t}|}{|V_{G_t}|}, &\qquad \Leftrightarrow \\
|V_{G_t}| > (1+\epsilon)(|V_{G_t}| - |R_t|), &\qquad \Leftrightarrow \\
|V_{G_{t+1}}| = |V_{G_t}| - |R_t| < \frac{|V_{G_t}|}{1+\epsilon}. &
\end{aligned}
$$

□

# Running time of the fast algo...

## Proof.

Then,

$$2|E_{G_t}| > 2(1+\epsilon)(|V_{G_t}| - |R_t|)\frac{|E_{G_t}|}{|V_{G_t}|}, \qquad \Leftrightarrow$$

$$|V_{G_t}| > (1+\epsilon)(|V_{G_t}| - |R_t|), \qquad \Leftrightarrow$$

$$|V_{G_{t+1}}| = |V_{G_t}| - |R_t| < \frac{|V_{G_t}|}{1+\epsilon}.$$

Therefore $|V_{G_t}| \leq 1$ in $\leq t$ steps for any $t$ such that $\frac{|V_G|}{(1+\epsilon)^t} \leq 1$, in particular when $t = \lceil \log_{1+\epsilon} |V_G| \rceil$. $\square$

# Streaming Algorithm for Densest Subgraph

The algorithm makes multiple passes over the data. During each pass:

- computes the current degree of each node as well as the current density;
- produces a new graph containing only the nodes with sufficiently large degree.
- maintains the current densest subgraph

**Theorem 3 (From [4])**

*There is a (semi-) streaming algorithm that for any $\epsilon > 0$, computes a $2(1+\epsilon)$-approximation of the densest subgraph while making $\lceil \log_{1+\epsilon}(|V_G|) \rceil$ passes over the data and requiring $O(n \log n)$ total memory.*

# References I


[1] M. Mitzenmacher and E. Upfal.
Probability and Computing : Randomized Algorithms and Probabilistic Analysis.
Cambridge University Press, New York (NY), 2005. Section 1.3, page 9.

[2] Nemhauser, George L., Laurence A. Wolsey, and Marshall L. Fisher.
An analysis of approximations for maximizing submodular set functions.
Mathematical Programming 14.1 (1978): 265-294.

[3] Kempe, David, Jon Kleinberg, and va Tardos.
Maximizing the spread of influence through a social network.
Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2003.

[4] Bahmani, Bahman, Ravi Kumar, and Sergei Vassilvitskii.
Densest subgraph in streaming and mapreduce.
Proceedings of the VLDB Endowment 5.5 (2012): 454-465.

# References II


[5] McGregor, Andrew.
Graph stream algorithms: a survey.
ACM SIGMOD Record 43.1 (2014): 9-20.

[6] Badanidiyuru, A., Mirzasoleiman, B., Karbasi, A., Krause, A.
Streaming submodular maximization: Massive data summarization on the fly.
Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2014.

[7] Bhattacharya, S., Henzinger, M., Nanongkai, D., Tsourakakis, C.
Space-and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams.
Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing. ACM, 2015.

[8] Epasto, A., Lattanzi, S., Sozio, M.
Efficient densest subgraph computation in evolving graphs.
In Proceedings of the 24th International Conference on World Wide Web (pp. 300-310). ACM.

# References III


[9] Lattanzi, S., Moseley, B., Suri, S., Vassilvitskii, S.
Filtering: a method for solving graph problems in mapreduce.
In Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures (pp. 85-94). ACM.

[10] Mirrokni, V., Zadimoghaddam, M.
Randomized composable core-sets for distributed submodular maximization.
In Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing (pp. 153-162). ACM.