Streaming Graph Mining

Mauro Sozio

Telecom ParisTech

January 22, 2019

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].

Problem Definition: Streaming Submodular Max. [6]

Definition 1

Given a streaming $v_1, \ldots, v_n$ of elements from a domain X as well as a non-negative *monotone submodular* function $f : 2^X \to \mathbb{R}^+$, an integer $k > 0$, find $S \subseteq X$ which maximizes $f(S)$, $|S| \leq k$, while making only one pass over the data and using *sublinear* amount of memory.

The v_i's must be processed sequentially one after the other.

Since the problem is NP-hard we resort to approximation algorithms.

Towards a Streaming Algorithm

Goal: A 1-pass streaming algo with $O(\frac{k \log k}{\epsilon})$ memory, constant approx.

We have very little memory, so we need to decide as soon as we see v_i.

Towards a Streaming Algorithm

Goal: A 1-pass streaming algo with $O(\frac{k \log k}{\epsilon})$ memory, constant approx.

We have very little memory, so we need to decide as soon as we see v_i.

Idea: Suppose it holds that for any $A \subseteq X$, $f(A) = \sum_{v \in A} f(v)$.

Towards a Streaming Algorithm

Goal: A 1-pass streaming algo with $O(\frac{k \log k}{\epsilon})$ memory, constant approx.

We have very little memory, so we need to decide as soon as we see v_i.

Idea: Suppose it holds that for any $A \subseteq X$, $f(A) = \sum_{v \in A} f(v)$. Suppose we know OPT. We include v_i iff $f(v_i) \geq \frac{OPT}{2k}$. This gives a 2-approx.

Towards a Streaming Algorithm

Goal: A 1-pass streaming algo with $O(\frac{k \log k}{\epsilon})$ memory, constant approx.

We have very little memory, so we need to decide as soon as we see v_i.

Idea: Suppose it holds that for any $A \subseteq X$, $f(A) = \sum_{v \in A} f(v)$. Suppose we know OPT. We include v_i iff $f(v_i) \geq \frac{OPT}{2k}$. This gives a 2-approx.

Issues:

- we don't know OPT (we can guess it!)
- general submodular functions?

Knowing the OPT or an Approx

Let $\Delta_f(v|S) = f(S \cup \{v\}) - f(S)$, i.e. the marginal contribution of v given the current solution S.

1: **Input**: ϕ such that $\alpha \cdot OPT \leq \phi \leq OPT, \epsilon > 0$
2: $S \leftarrow \emptyset$
3: **for** $i = 1, \ldots, n$ **do**
4: **if** $|S| < k$ and $\Delta_f(v_i|S) \geq \frac{\phi/2 - f(S)}{k - |S|}$ **then**
5: $S \leftarrow S \cup \{v_i\}$
6: return S

Approximation Guarantee and Update Time

Theorem 2

Algorithm 1 outputs a set S such that $|S| \leq k$ and $f(S) \geq \frac{\alpha}{2} OPT$. It makes one pass over the data, stores at most k elements and has $O(1)$ update time on average.

Knowing the Max Value Helps

Suppose we know the maximum value $f_{\max} = \max_{v \in X} f(\{v\})$. From submodularity it follows:

$$f_{\max} \leq OPT \leq k \cdot f_{\max}.$$

Knowing the Max Value Helps

Suppose we know the maximum value $f_{\max} = \max_{v \in X} f(\{v\})$. From submodularity it follows:

$$f_{\max} \leq OPT \leq k \cdot f_{\max}.$$

Then, for any $\epsilon > 0$, we can guess the OPT by considering all integral powers of $(1+\epsilon)$ in the range $[f_{\max}, k \cdot f_{\max}]$ i.e.

$$B := \{(1+\epsilon)^i | i \in \mathbb{Z}, f_{\max} \leq (1+\epsilon)^i \leq k \cdot f_{\max}.\}$$

Knowing the Max Value Helps

Suppose we know the maximum value $f_{\max} = \max_{v \in X} f(\{v\})$. From submodularity it follows:

$$f_{\max} \leq OPT \leq k \cdot f_{\max}.$$

Then, for any $\epsilon > 0$, we can guess the OPT by considering all integral powers of $(1+\epsilon)$ in the range $[f_{\max}, k \cdot f_{\max}]$ i.e.

$$B := \{(1+\epsilon)^i | i \in \mathbb{Z}, f_{\max} \leq (1+\epsilon)^i \leq k \cdot f_{\max}.\}$$

We run our algorithm for each value in B and we take the max.

Knowing the Max Value

Input: $f_{\max} \leftarrow \max_{v \in X} f(\{v\}), \epsilon > 0$
$B \leftarrow \{(1+\epsilon)^i | i \in \mathbb{Z}, f_{\max} \leq (1+\epsilon)^i \leq k \cdot f_{\max}\}$
For each $\phi \in B$, $S_\phi \leftarrow \emptyset$
for $i = 1, \ldots, n$ **do**
  for $\phi \in B$ **do**
    if $|S_\phi| < k$ and $\Delta_f(v_i|S_\phi) \geq \frac{\phi/2 - f(S_\phi)}{k - |S_\phi|}$ **then**
      $S_\phi \leftarrow S_\phi \cup \{v_i\}$
return $\arg\max_{\phi \in B} f(S_\phi)$

Approximation Guarantee and Update Time

We don't know $f_{\max}$ but we can compute it with one pass over the data. This would require two passes.

Theorem 3

Given $\epsilon > 0$, Algorithm 3 outputs a set S such that $|S| \leq k$ and $f(S) \geq \left(\frac{1}{2} - \epsilon\right) \cdot OPT$. It makes two passes over the data, stores at most $O(\frac{k \log k}{\epsilon})$ elements and has $O(\frac{\log k}{\epsilon})$ update time per element (on average).

Towards the Final Algorithm

We can do it in one pass:

- $f_{\max}$ can be estimated as we process the stream.
- we need to expand B as follows. Let $\hat{f}_{\max}$ be current guess for $f_{\max}$:

$$B := \{(1+\epsilon)^i | i \in \mathbb{Z}, \hat{f}_{\max} \leq (1+\epsilon)^i \leq \mathbf{2} \cdot k \cdot \hat{f}_{\max}, \}$$

otherwise, when we see a new element with marginal value $f > \hat{f}_{\max}$, $S_{k \cdot f}$ is empty while we might have seen elements with marginal value $\frac{k \cdot f}{2 \cdot k} = \frac{f}{2}$ that should have been place in $S_{k \cdot f}$ (see line 6).

Towards the Final Algorithm

We can do it in one pass:

- $f_{\max}$ can be estimated as we process the stream.
- we need to expand B as follows. Let $\hat{f}_{\max}$ be current guess for $f_{\max}$:

$$B := \{(1+\epsilon)^i | i \in \mathbb{Z}, \hat{f}_{\max} \leq (1+\epsilon)^i \leq \mathbf{2} \cdot k \cdot \hat{f}_{\max}, \}$$

otherwise, when we see a new element with marginal value $f > \hat{f}_{\max}$, $S_{k \cdot f}$ is empty while we might have seen elements with marginal value $\frac{k \cdot f}{2 \cdot k} = \frac{f}{2}$ that should have been place in $S_{k \cdot f}$ (see line 6).

With such a change, if S_f is empty then $f > 2 \cdot k \cdot \hat{f}_{\max}$ which implies that S_f should contain elements with marginal value $> \frac{2 \cdot k \cdot \hat{f}_{\max}}{2k} = \hat{f}_{\max}$ which have not been seen before.

Final Algorithm

Input: $\epsilon > 0$
$\phi \leftarrow 0$
for $i = 1, \ldots, n$ **do**
  $\hat{f}_{\max} \leftarrow \max(\hat{f}_{\max}, f(\{v_i\}))$
  $B_i \leftarrow \{(1+\epsilon)^i | \hat{f}_{\max} \leq (1+\epsilon)^i \leq 2 \cdot k \cdot \hat{f}_{\max}\}$
  Delete all S_ϕ such that $\phi \notin B_i$
  For each $\phi \notin B_i$ such that S_ϕ is not initialized, $S_\phi \leftarrow \emptyset$
  for $\phi \in B_i$ **do**
    if $|S_\phi| < k$ and $\Delta_f(v_i|S_\phi) \geq \frac{\phi/2 - f(S_\phi)}{k - |S_\phi|}$ **then**
      $S_\phi \leftarrow S_\phi \cup \{v_i\}$
return $\arg\max_{\phi \in B_n} f(S_\phi)$

Approximation Guarantee and Update Time

Theorem 4

Given $\epsilon > 0$, Algorithm 3 outputs a set S such that $|S| \leq k$ and $f(S) \geq \left(\frac{1}{2} - \epsilon\right) \cdot OPT$. It makes one pass over the data, stores at most $O(\frac{k \log k}{\epsilon})$ elements and has $O(\frac{\log k}{\epsilon})$ update time per element (on average).

Densest Subgraph Computation in Streaming

Definition 5

Given a streaming $e_1, \ldots, e_n$ of edges from an undirected graph $G = (V, E)$, compute a densest subgraph in G, while using limited amount of memory. Multiple passes over the data are allowed.

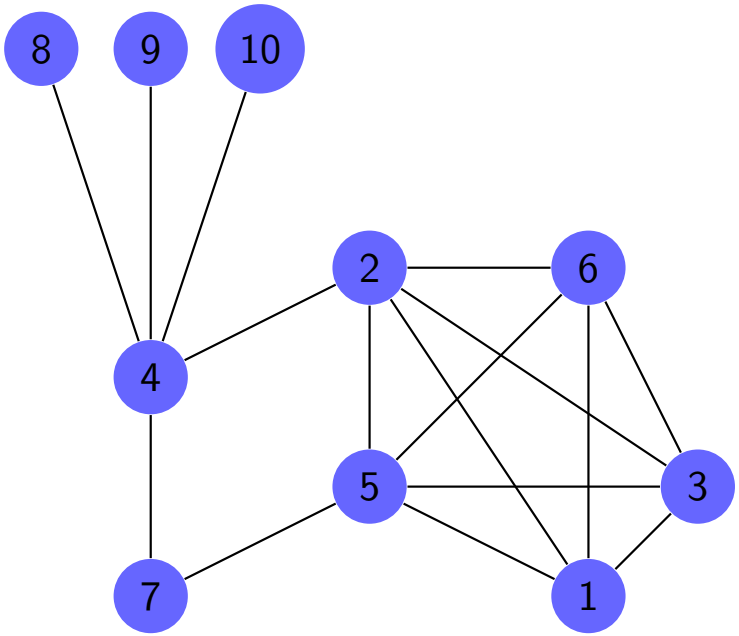
Centralized Algorithm for Densest Subgraph

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

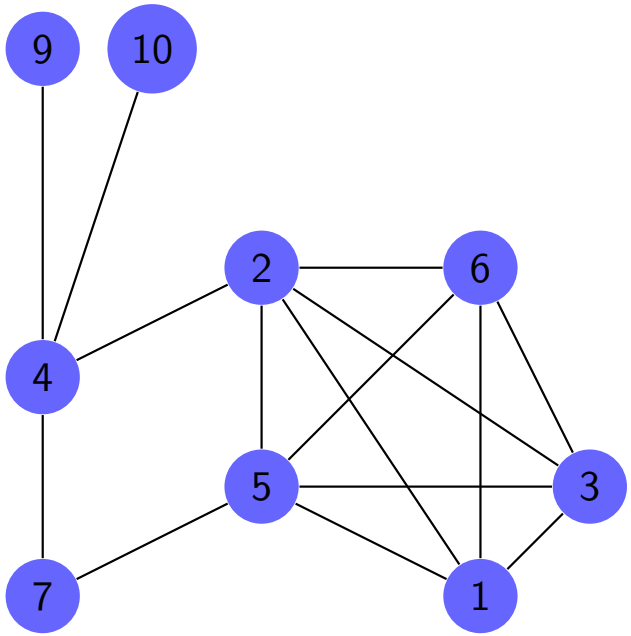
- let v be the node with minimum degree $\delta_G(v)$ in G;
- remove v and all its edges from G;
- if $\rho(G) > \rho(H)$ then $H \leftarrow G$;

return H;

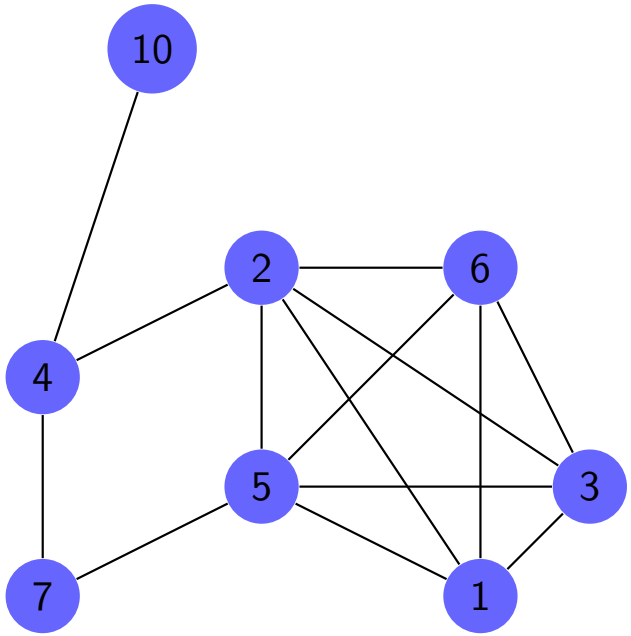
Densest Subgraph Algorithm:Example



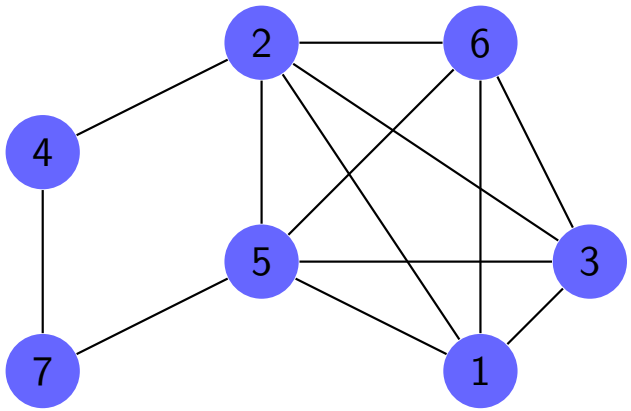
Densest Subgraph Algorithm:Example



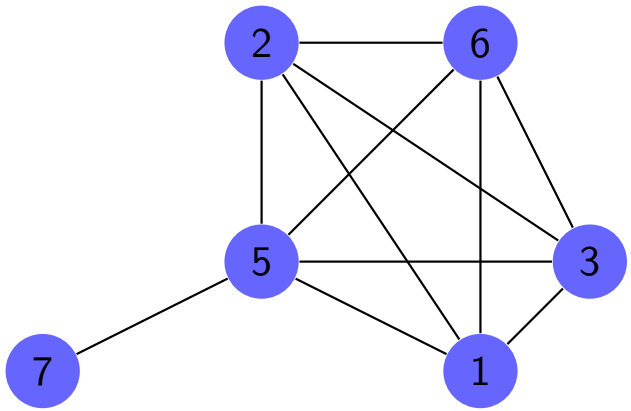
Densest Subgraph Algorithm:Example



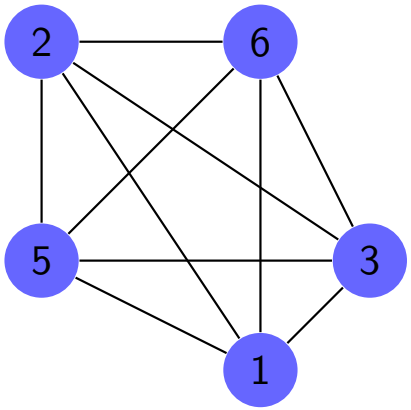
Densest Subgraph Algorithm:Example



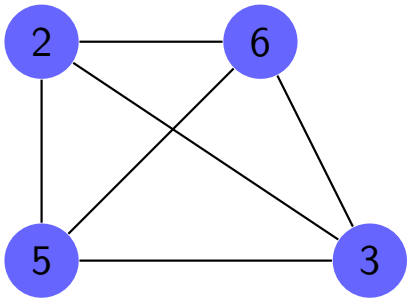
Densest Subgraph Algorithm:Example



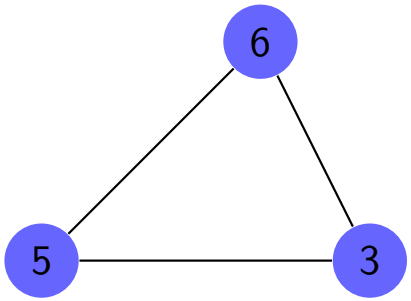
Densest Subgraph Algorithm:Example



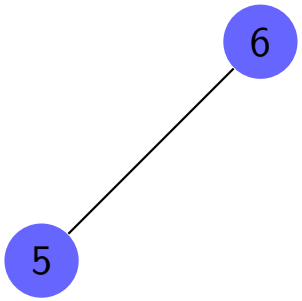
Densest Subgraph Algorithm:Example



Densest Subgraph Algorithm:Example

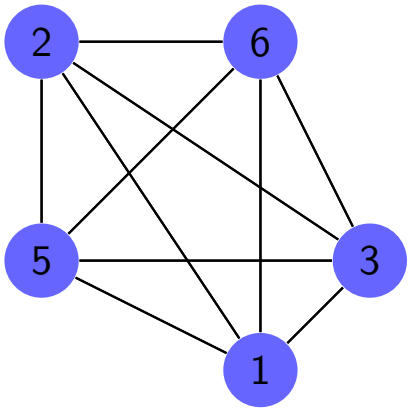


Densest Subgraph Algorithm:Example



Densest Subgraph Algorithm:Example

Densest Subgraph Algorithm:Example



Approximation Guarantee

Theorem 6

Let O be a densest subgraph in G. Our algorithm finds a subgraph H s.t.

$$\rho(H) \geq \frac{\rho(O)}{2}.$$

Approximation Guarantee

Lemma 7

Let O be a densest subgraph in G, then:

$$\forall v \in V_O \quad \delta_O(v) \geq \rho(O).$$

Proof.

We show that if there is v in O with $\delta_O(v) < \rho(O)$, then O is not densest.

$$\rho(O \setminus \{v\}) = \frac{|E_O| - \delta_O(v)}{|V_O| - 1}$$

Approximation Guarantee

Lemma 7

Let O be a densest subgraph in G, then:

$$\forall v \in V_O \quad \delta_O(v) \geq \rho(O).$$

Proof.

We show that if there is v in O with $\delta_O(v) < \rho(O)$, then O is not densest.

$$\begin{aligned} \rho(O \setminus \{v\}) &= \frac{|E_O| - \delta_O(v)}{|V_O| - 1} \\ &> \frac{|E_O| - \rho(O)}{|V_O| - 1} \end{aligned}$$

Approximation Guarantee

Lemma 7

Let O be a densest subgraph in G, then:

$$\forall v \in V_O \quad \delta_O(v) \geq \rho(O).$$

Proof.

We show that if there is v in O with $\delta_O(v) < \rho(O)$, then O is not densest.

$$\begin{aligned}
\rho(O \setminus \{v\}) &= \frac{|E_O| - \delta_O(v)}{|V_O| - 1} \\
&> \frac{|E_O| - \rho(O)}{|V_O| - 1} \\
&= \frac{|V_O|\rho(O) - \rho(O)}{|V_O| - 1} = \rho(O)\frac{|V_O| - 1}{|V_O| - 1} = \rho(O).
\end{aligned}$$

A Faster Algorithm for Densest Subgraph

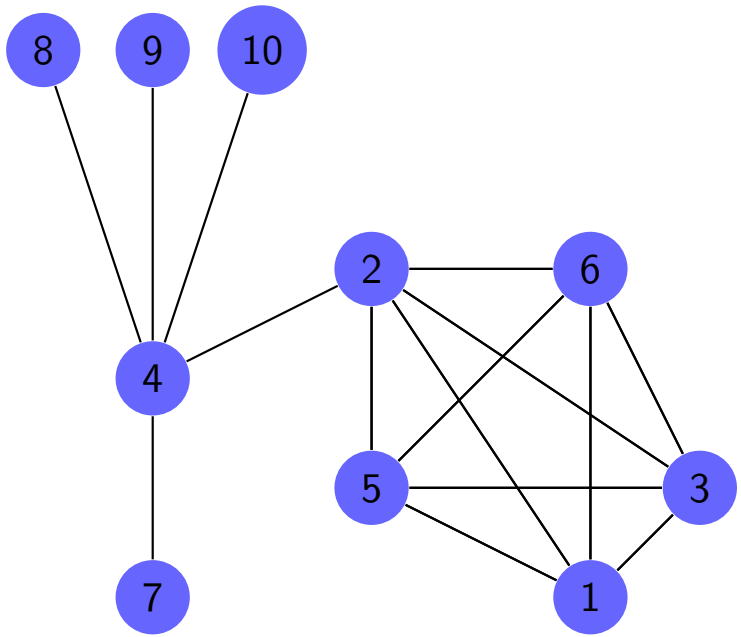
Require: an undirected graph G, a value $\epsilon > 0$

$H = G$;
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 $H \leftarrow G$;

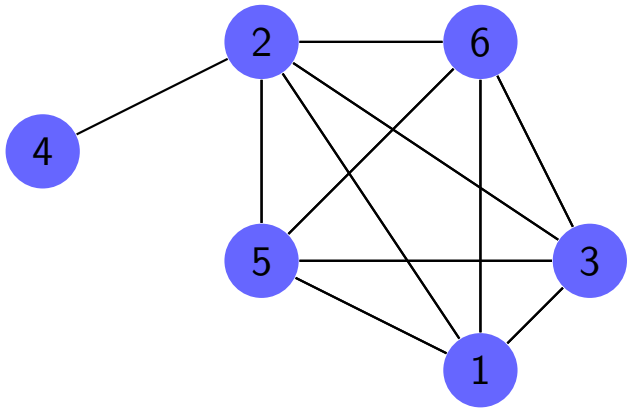
return H;

Faster algorithm: Example



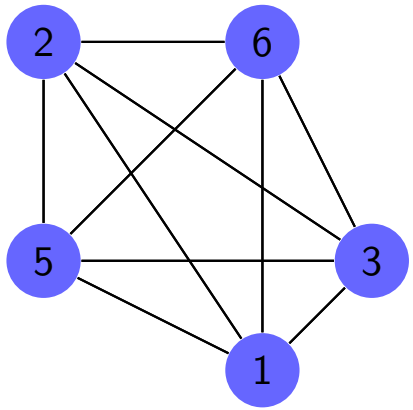
$\epsilon = 0.1$
Iteration 1:
$\rho(G) = \frac{16}{10}$, remove nodes with degree $\leq 2 * (1.1) * 1.6 = 3.52$.

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



$\epsilon = 0.1$

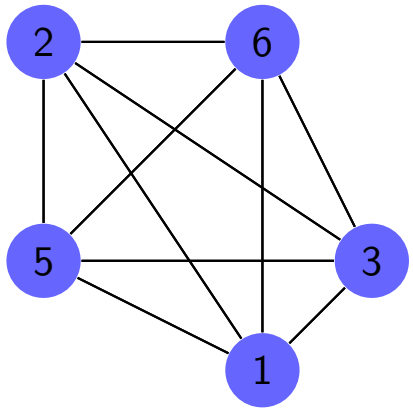
Iteration 3:

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

Faster algorithm: Example

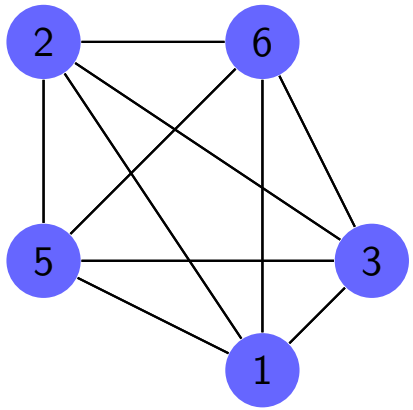
$\epsilon = 0.1$
Iteration 4:
Empty Graph.

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 ϵ is large? (say $\epsilon = 0.5$)

Approx. guarantee of the fast algo

Theorem 8

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 7, $\delta_{G_t}(v) \geq \delta_O(v) \geq \rho(O)$. Hence,

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

Approx. guarantee of the fast algo

Theorem 8

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 7, $\delta_{G_t}(v) \geq \delta_O(v) \geq \rho(O)$. 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 9

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 9

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 9

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,

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

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

Running time of the fast algo...

Proof.

Then,

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

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

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

Running time of the fast algo...

Proof.

Then,

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

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

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

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 $t = \lceil \log_{1+\epsilon} |V_G| \rceil$.

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 10 (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.