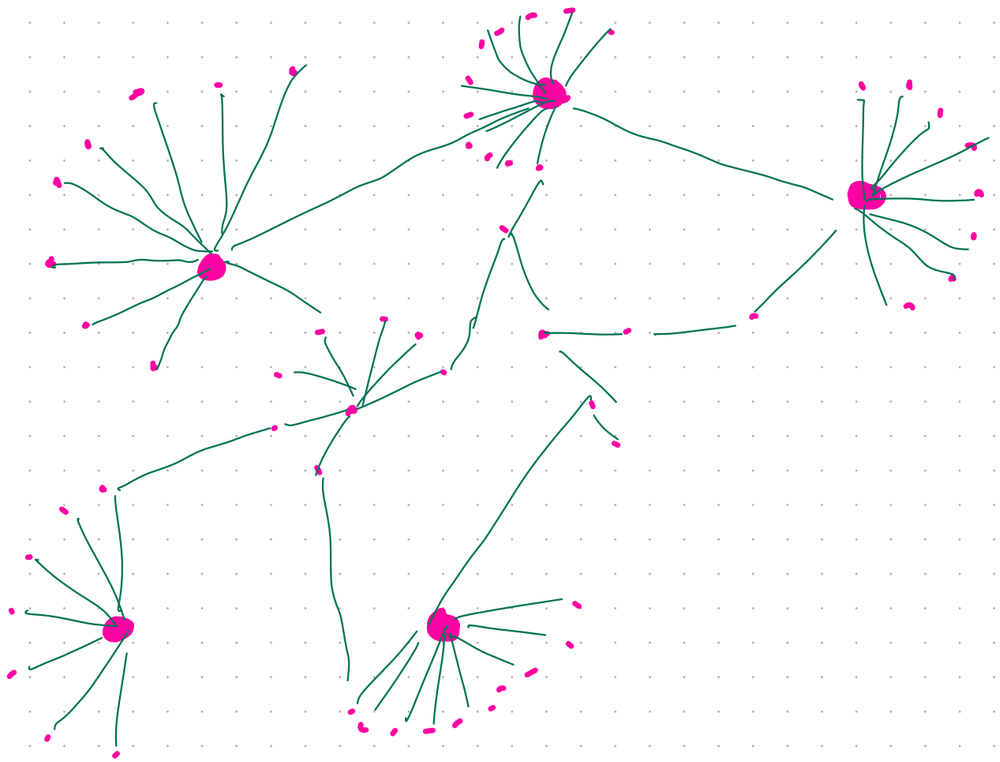
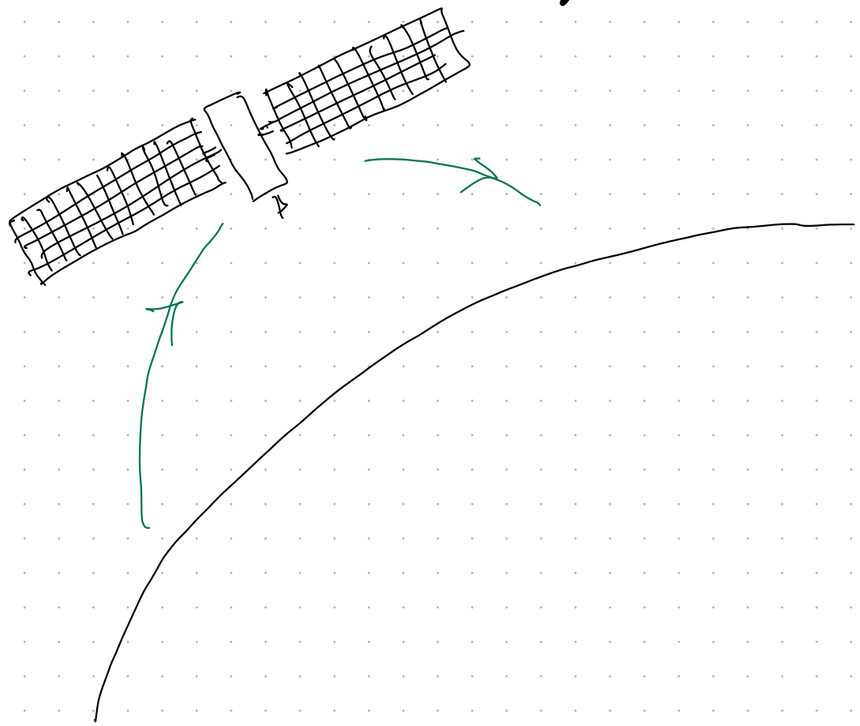
Something For Almost Nothing:

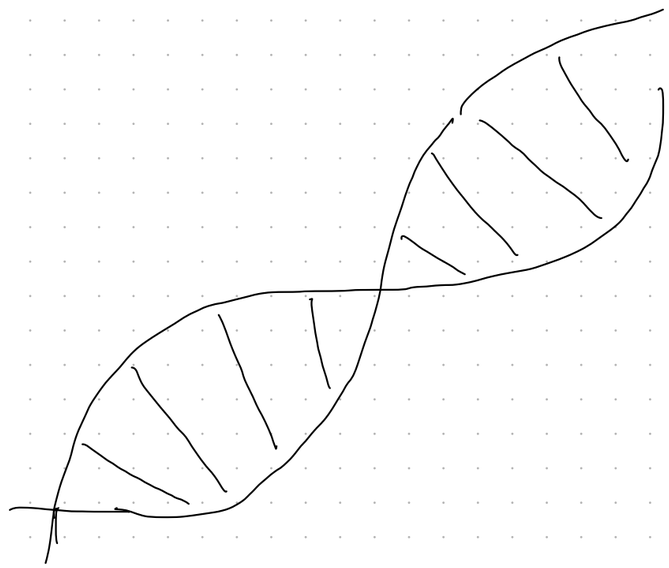
A gentle Introduction to Sublinear Algorithms

Nithin Varma

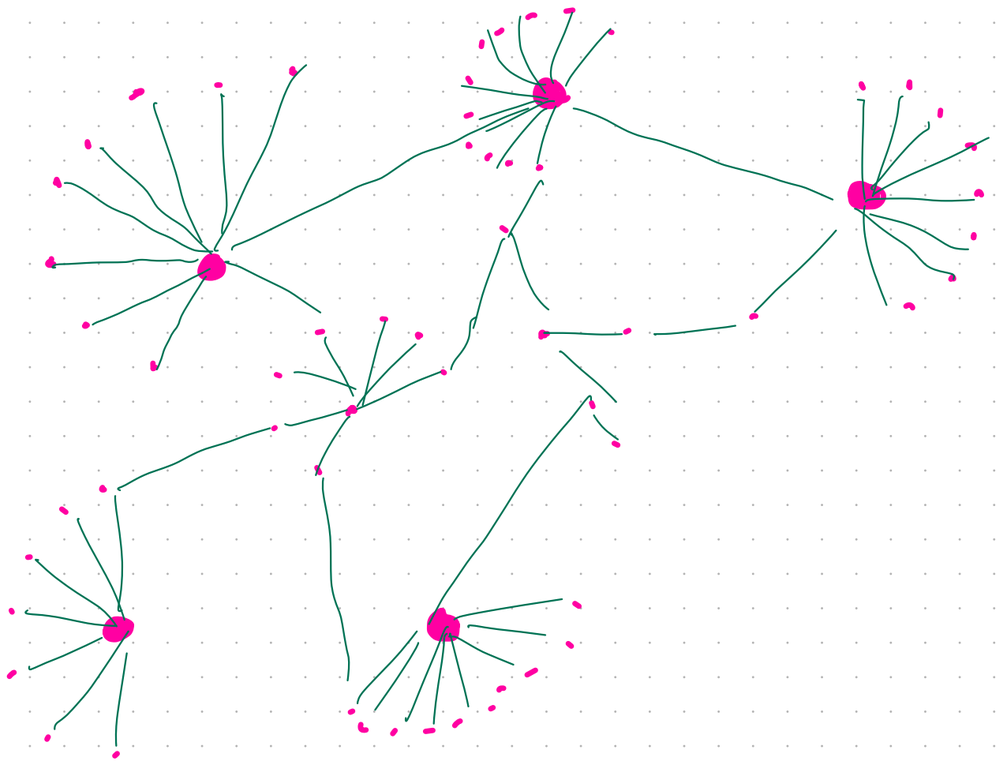
Massive Datasets are Everywhere!



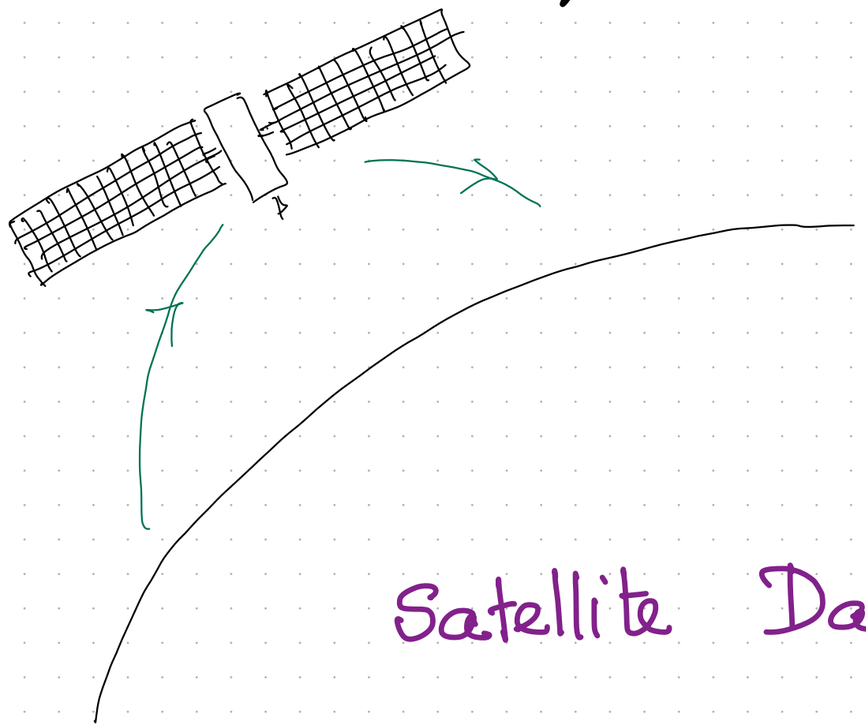




Massive Datasets are Everywhere!



Large Networks



Satellite Data

Genetic Data

- Can draw useful conclusions by analyzing such datasets

- Can draw useful conclusions by analyzing such datasets

- Main Question: How to efficiently analyze large datasets ?

- Can draw useful conclusions by analyzing such datasets

- Main Question: How to efficiently analyze large datasets ?

Even reading a large dataset can take a lot of time

SUBLINEAR-TIME COMPUTATION

- Cannot read the entire input

SUBLINEAR-TIME COMPUTATION

- Cannot read the entire input
- Running time of algorithm is a sublinear function of input size n

SUBLINEAR-TIME COMPUTATION

- Cannot read the entire input
- Running time of algorithm is a sublinear function of input size n

Do you already know any sublinear algorithms?

- Can we solve more challenging computational problems in sublinear-time ?

- Can we solve more challenging computational problems in sublinear-time ?

- YES !

- Can we solve more challenging computational problems in sublinear-time ?

- YES ! IF WE ARE ALLOWED TO

- Can we solve more challenging computational problems in sublinear-time ?

- YES ! IF WE ARE ALLOWED TO

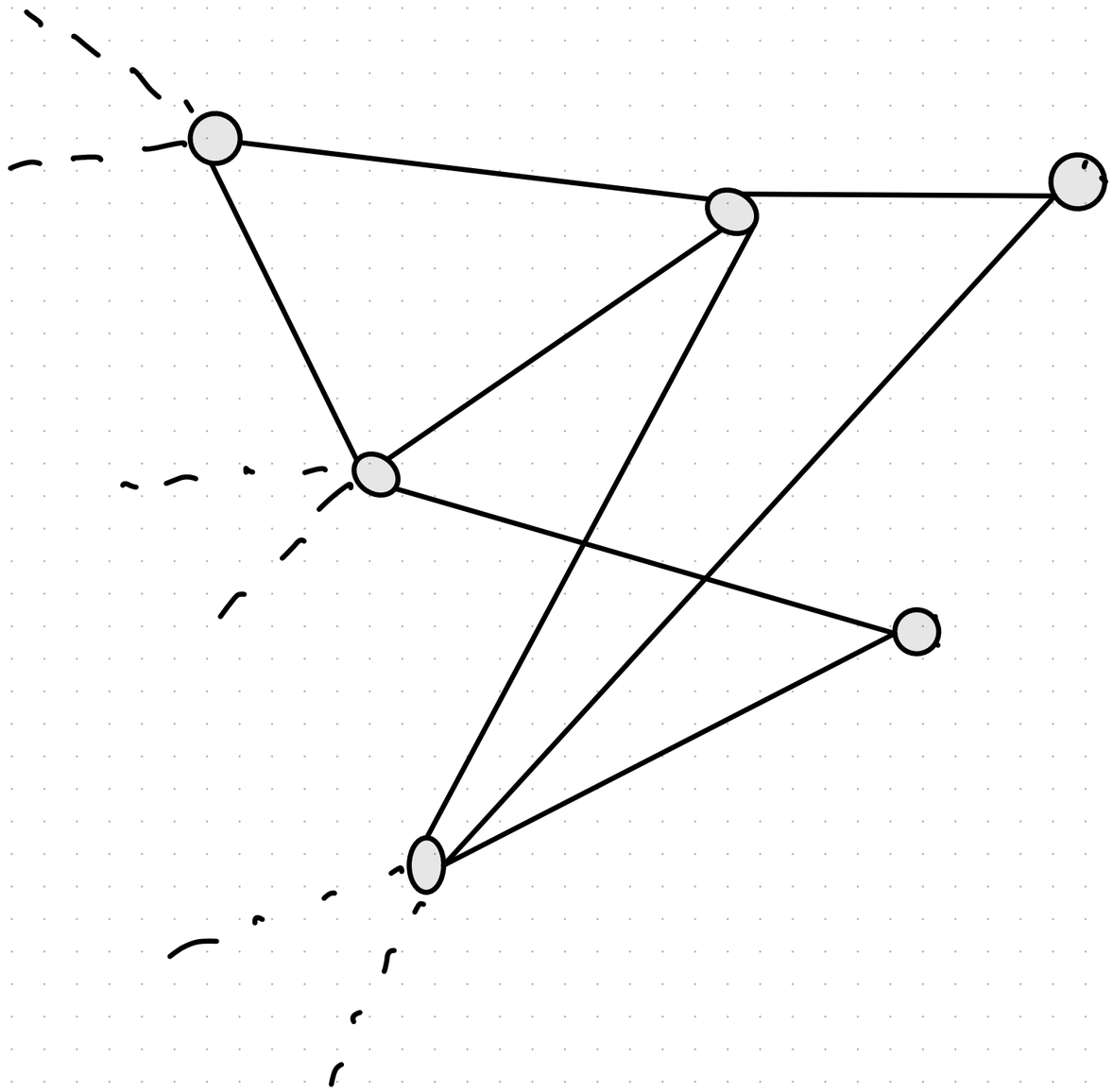
  * Output approximate answers

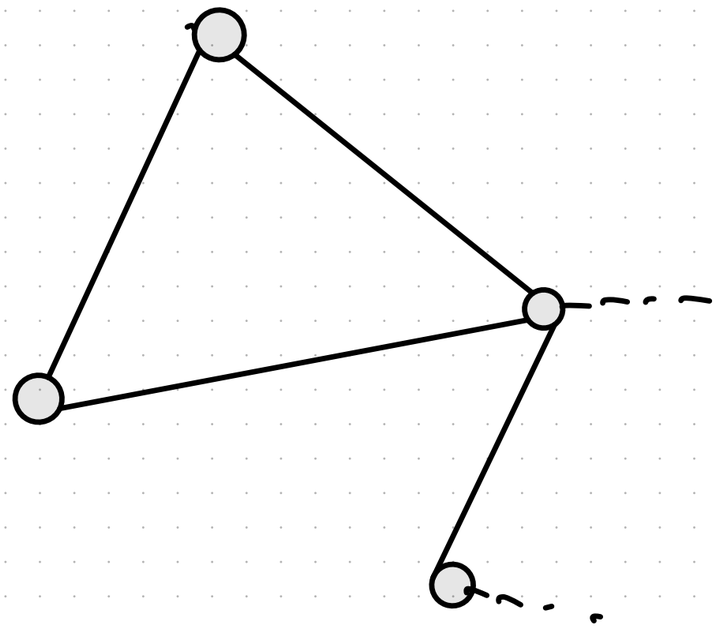
- Can we solve more challenging computational problems in sublinear-time ?

- YES ! IF WE ARE ALLOWED TO
  - * Output approximate answers
  - * Fail 1% of the time

Today: Sublinear-Time Algorithm for a Fundamental Graph Problem

Massive Graphs: Social Network

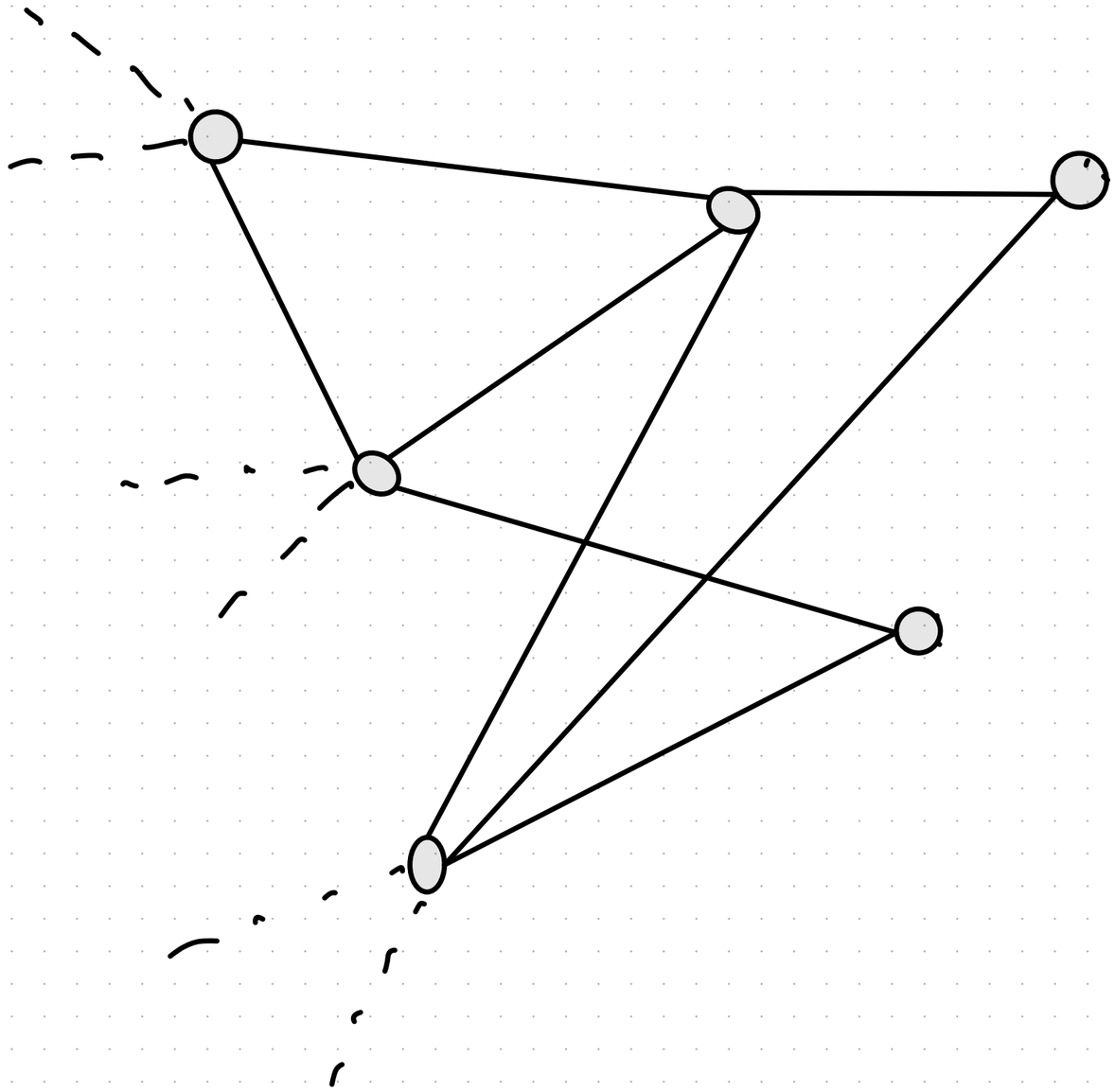


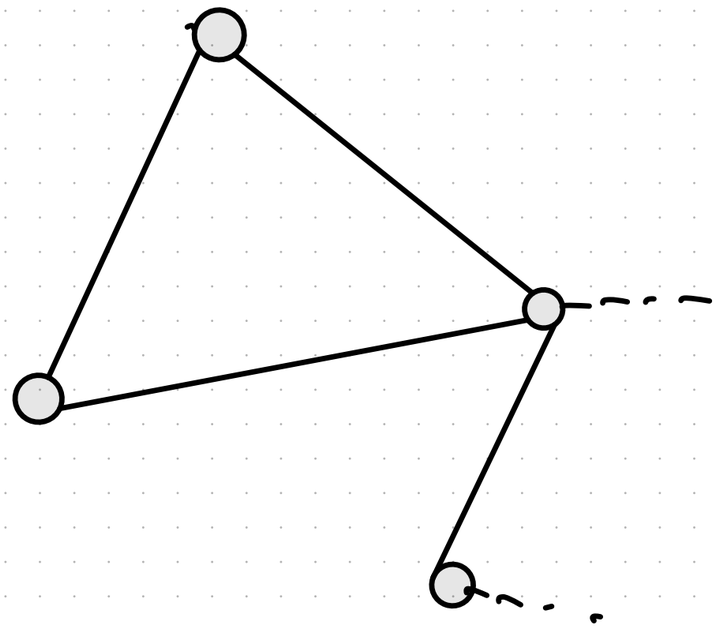


Massive Graphs : Social Network

Vertices : Users

Edges : Friendship

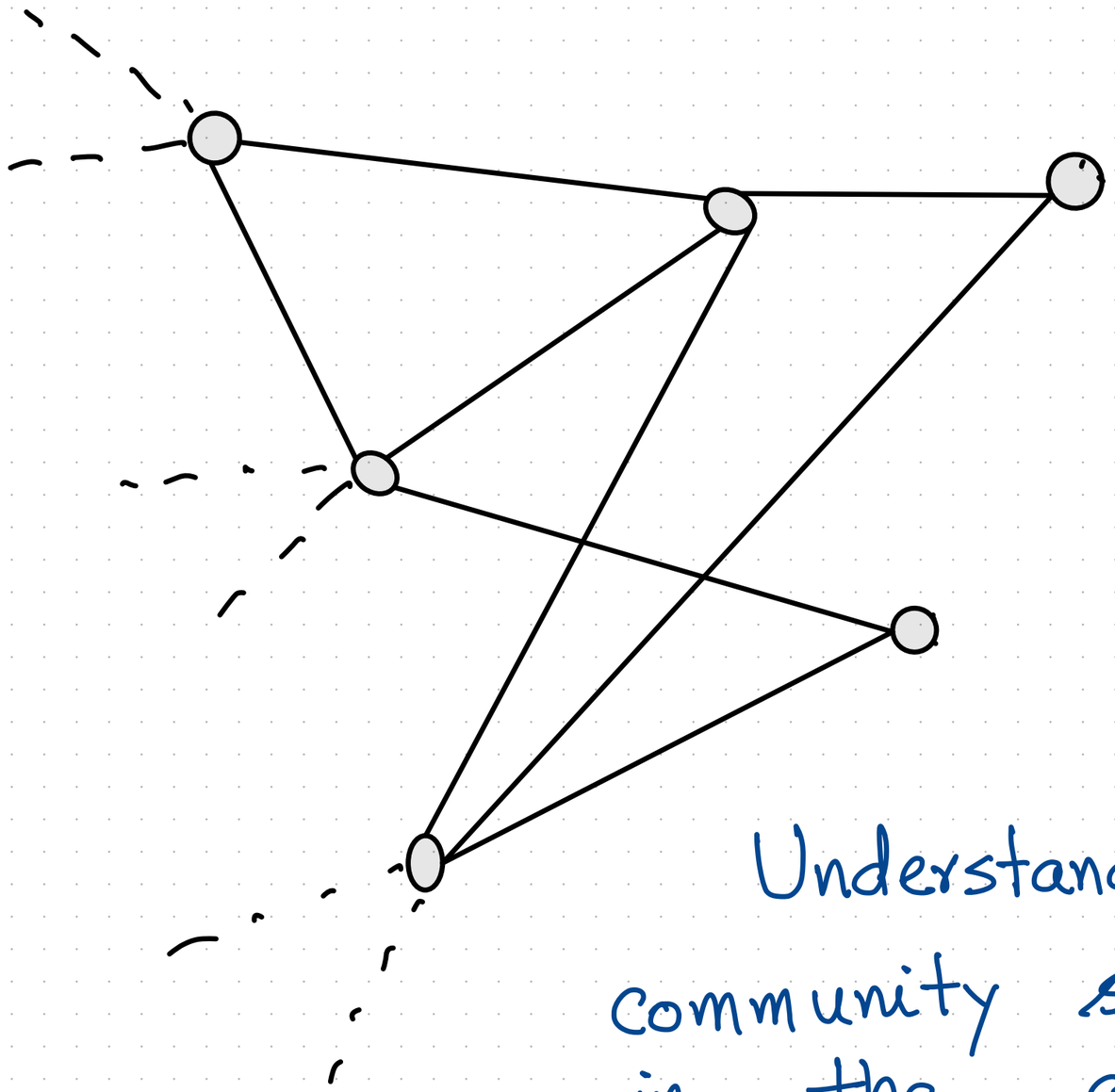


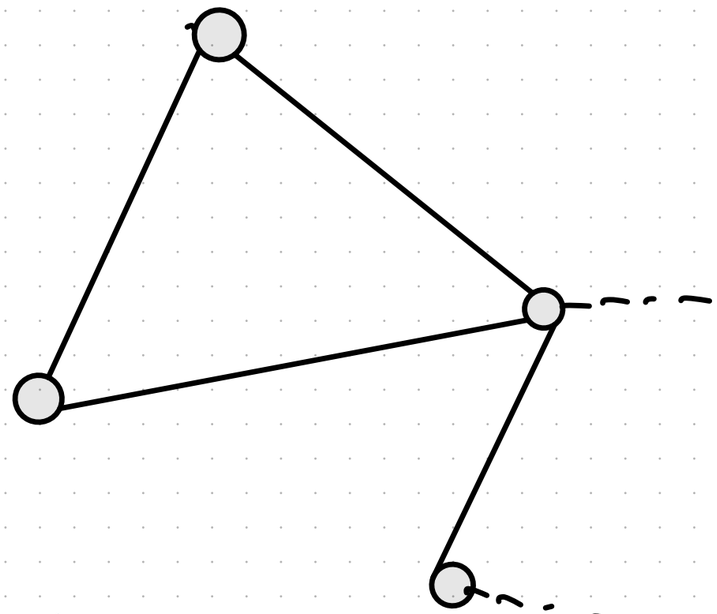


Massive Graphs : Social Network

Vertices : Users

Edges : Friendship





Understand community structure in the graph

Massive Graphs: Social Network

Big Goal: Understand community structure in the social network.

Massive Graphs: Social Network

Big Goal: Understand community structure in the social network.

What does this mean?

Massive Graphs: Social Network

Big Goal: Understand community structure in the social network.

What does this mean?

* Several definitions of "community" possible

Massive Graphs: Social Network

Big Goal: Understand community structure in the social network.

What does this mean?

* Several definitions of "community" possible

Simpler Goal: Is the graph connected or not?

Problem: Given a graph, is it connected or not?

Problem: Given a graph, is it connected or not?

Goal: Design a sublinear-time algorithm for the problem

Problem: Given a graph, is it connected or not?

Goal: Design a sublinear-time algorithm for the problem

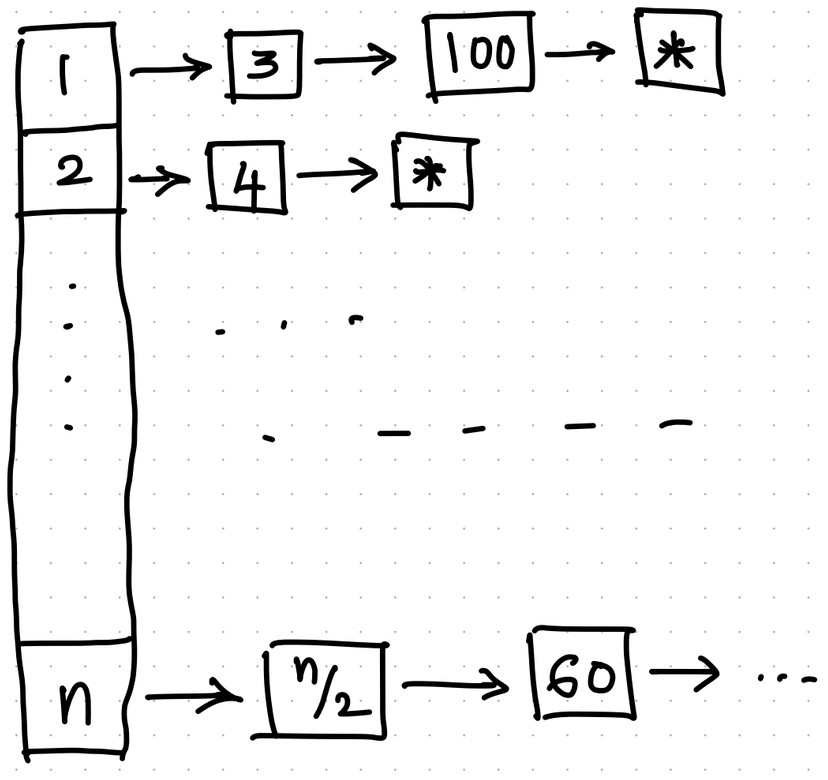
Connectedness in Sublinear-Time ?

Graph $G = (V, E)$ n vertices m edges

Connectedness in Sublinear-Time ?

Graph $G = (V, E)$ n vertices m edges

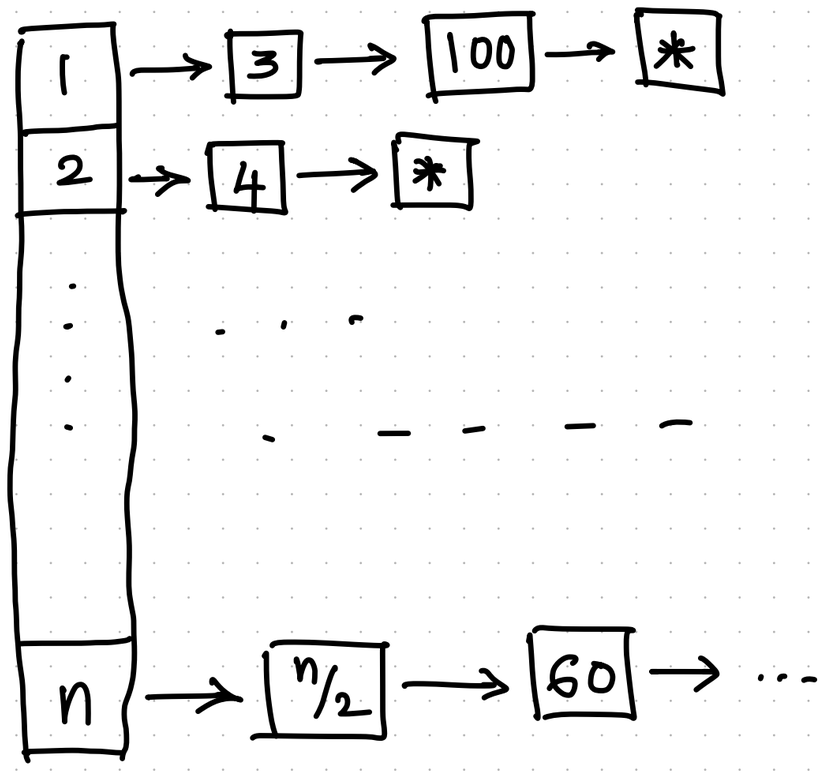
★ Represented as adjacency lists



Connectedness in Sublinear-Time ?

Graph $G = (V, E)$ n vertices m edges

★ Represented as adjacency lists



★ Access via queries

Connectedness in Sublinear-Time ?

Graph $G = (V, E)$ n vertices m edges

★ Represented as adjacency lists

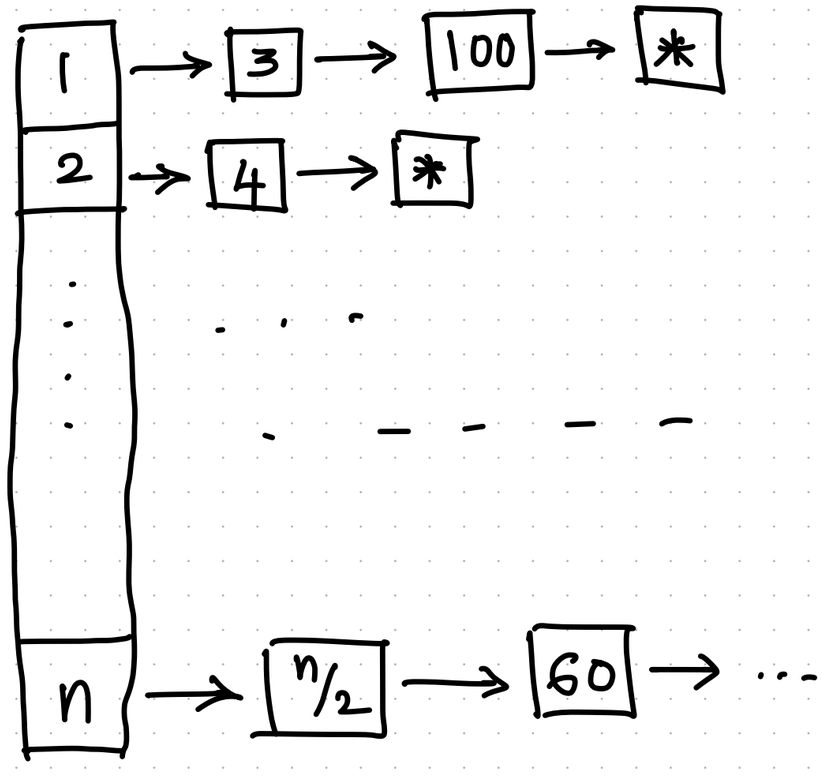
★ Access via queries

- Degree query:

  What is degree of vertex v?

- Neighbor query:

  Who is i^{th} nbr of v?



Can we check if a graph is connected without examining the entire adjacency lists ?

Can we check if a graph is connected without examining the entire adjacency lists ?

NO !

Can we check if a graph is connected without examining the entire adjacency lists ?

NO !

Need to "approximately" check connectedness

Relaxed Problem

Design an algorithm that

Relaxed Problem

Design an algorithm that

- Outputs YES if graph connected

Relaxed Problem

Design an algorithm that

- Outputs YES if graph connected
- Outputs NO if graph contains "many" connected components

Problem

Design an algorithm that gets query access to a graph G, input $\varepsilon \in (0,1)$

- Outputs YES if G connected
- Outputs NO if G contains $\geq \varepsilon m$ components

Problem

Design an algorithm that gets query access to a graph G, input $\varepsilon \in (0,1)$

- Outputs YES if G connected
- Outputs NO if G contains $\geq \varepsilon m$ components connected

m → # edges in G

Problem [Property Testing Problem]

Design an algorithm that gets query access to a graph G, input $\varepsilon \in (0,1)$

- Outputs YES if G connected
- Outputs NO if G contains $\geq \varepsilon m$ components

m → # edges in G

Problem [Property Testing Problem]

Design an algorithm that gets query access to a graph G, input $\varepsilon \in (0,1)$

- Outputs YES if G connected
- Outputs NO if G contains $\geq \varepsilon m$ components connected

m → # edges in G

* Trivial if $\varepsilon > \frac{n}{m}$

Problem [Property Testing Problem]

Design an algorithm that gets query access to a graph G, input $\varepsilon \in$ ~~$(0,1)$~~ $(0, \frac{n}{m})$

- Outputs YES if G connected
- Outputs NO if G contains $\geq \varepsilon m$ components connected

m → #edges in G

* Trivial if $\varepsilon > \frac{n}{m}$

→ output YES without querying

Problem [Property Testing Problem]

Design an algorithm that gets query access to a graph G, input $\varepsilon \in$ ~~$(0,1)$~~ $(0, \frac{n}{m})$

- Outputs YES if G connected
- Outputs NO if G contains $\geq \varepsilon m$ components connected

m → #edges in G

* Smaller ε $\Rightarrow$ More challenging problem

Problem [Property Testing Problem]

Design an algorithm that gets query access to a graph G, input $\varepsilon \in$ ~~$(0,1)$~~ $(0, \frac{n}{m})$

- Outputs YES if G connected
- Outputs NO if G contains $\geq \varepsilon m$ components

G connected

$\varepsilon m \rightarrow$ # edges in G

} Sufficient to be correct with probability $\geq \frac{2}{3}$

Towards an Algorithm: How do NO instances look like ?

Towards an Algorithm: How do NO instances look like?

- $\geq \varepsilon m$ connected components
- Overall, n vertices

Towards an Algorithm: How do NO instances look like?

- $\geq \varepsilon m$ connected components
- Overall, n vertices

Claim: At least $\frac{\varepsilon m}{2}$ components that each contain $\leq \frac{2n}{\varepsilon m}$ vertices each

Towards an Algorithm: How do NO instances look like?

Claim: At least $\frac{\varepsilon m}{2}$ components that each contain $\leq \frac{2n}{\varepsilon m}$ vertices each

Proof:

$$\left.\begin{array}{l}\#\text{components with} \\ > \frac{2n}{\varepsilon m} \text{ vertices}\end{array}\right\} < \frac{n}{2n/\varepsilon m} = \frac{\varepsilon m}{2}$$

Towards an Algorithm: How do NO instances look like?

Claim: At least $\frac{\varepsilon m}{2}$ components that each contain $\leq \frac{2n}{\varepsilon m}$ vertices each

Proof:

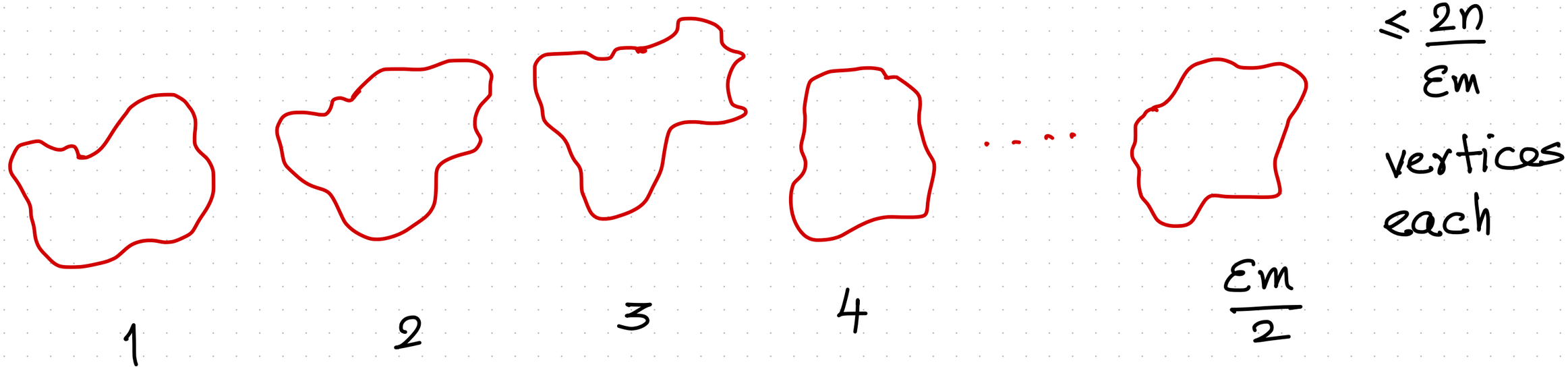
$$\left.\begin{array}{l}\#\text{components with} \\ > \frac{2n}{\varepsilon m} \text{ vertices}\end{array}\right\} < \frac{n}{2n/\varepsilon m} = \frac{\varepsilon m}{2}$$

Remaining $\geq \varepsilon m - \frac{\varepsilon m}{2}$ components are "small"

Towards an Algorithm: How do NO instances look like?

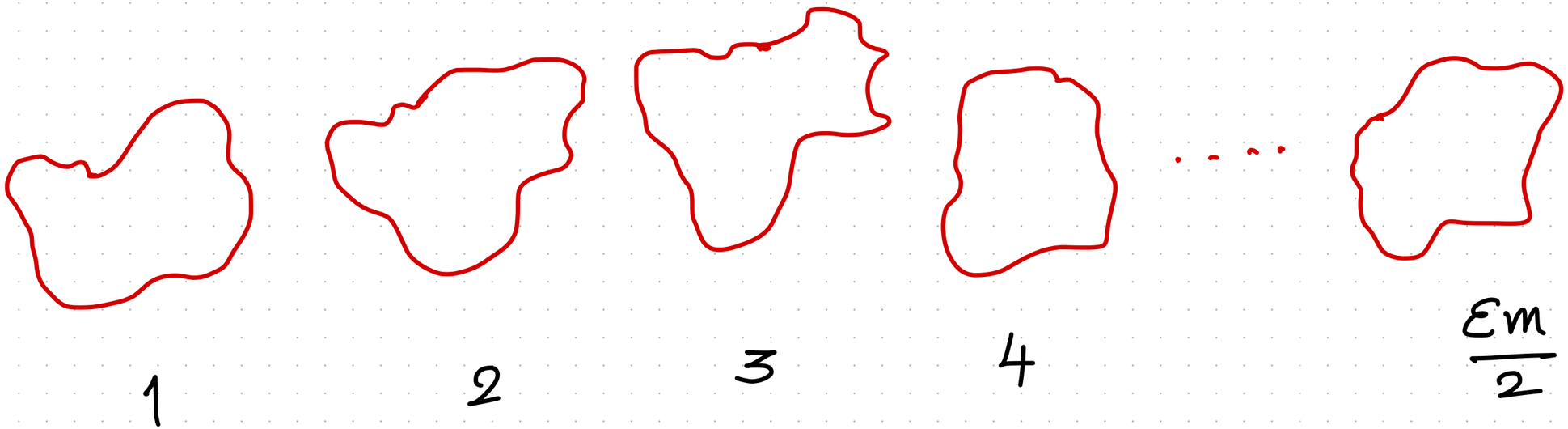
Claim: At least $\frac{\varepsilon m}{2}$ components that each contain $\leq \frac{2n}{\varepsilon m}$ vertices each

A NO instance G



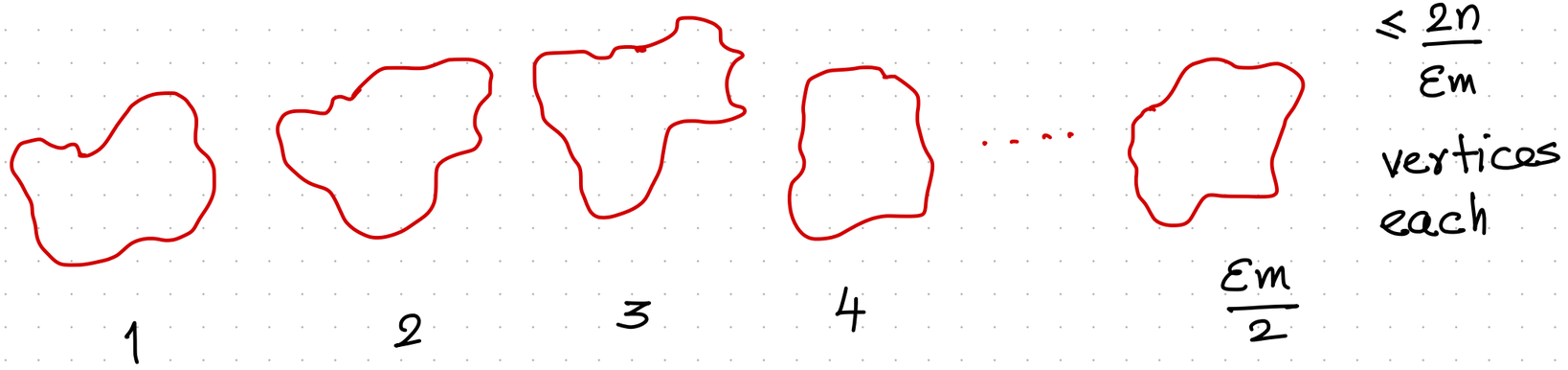
Towards an Algorithm: How do NO instances look like?

A NO instance G



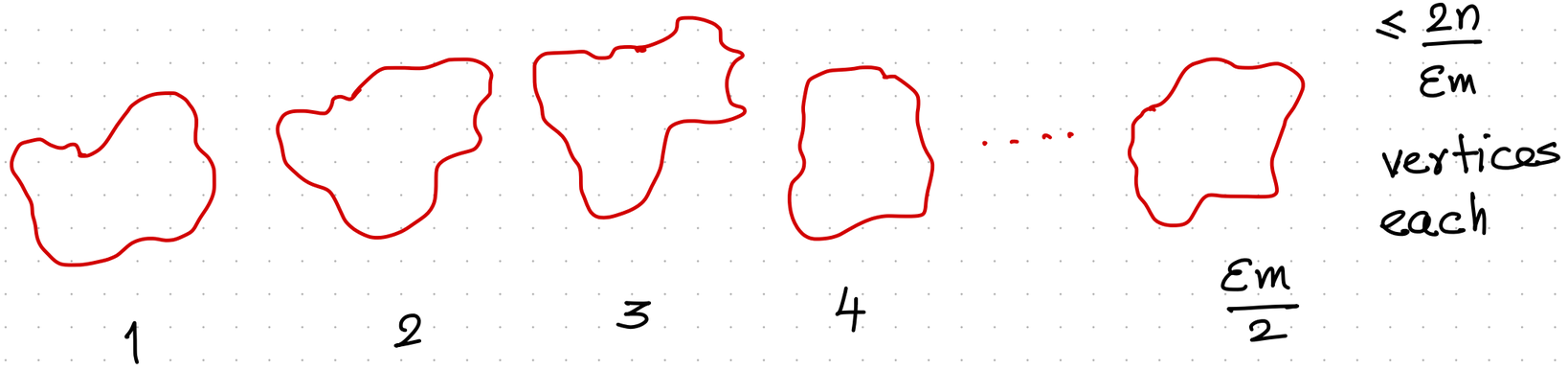
If we can detect one such component, we have <u>proof</u> of disconnectedness.

A NO instance G



- If we can detect one such component, we have proof of disconnectedness.
- How to detect a component like this?

A NO instance G



- If we can detect one such component, we have proof of disconnectedness.
- How to detect a component like this?
  - ★ A uniformly random vertex belongs to such a component with "good" probability

Connectedness Testing Algorithm (Goldreich Ron '02)

Given query access to $G = (V, E)$, input ε

Connectedness Testing Algorithm (Goldreich Ron '02)

Given query access to $G = (V, E)$, input ε

- Sample $v \sim V$ uniformly at random

Connectedness Testing Algorithm (Goldreich Ron '02)

Given query access to $G = (V, E)$, input ε

- Sample $v \sim V$ uniformly at random
- Check whether v belongs to a component with $\leq \frac{2n}{\varepsilon m}$ vertices

Connectedness Testing Algorithm (Goldreich Ron '02)

Given query access to $G = (V, E)$, input ε

- Sample $v \sim V$ uniformly at random
- Check whether v belongs to a component with $\leq \frac{2n}{\varepsilon m}$ vertices by performing a BFS from v until seeing $\leq \frac{2n}{\varepsilon m}$ vertices

Connectedness Testing Algorithm (Goldreich Ron '02)

Given query access to $G = (V, E)$, input ε

- Sample $v \sim V$ uniformly at random
- Check whether v belongs to a component with $\leq \frac{2n}{\varepsilon m}$ vertices by performing a BFS from v until seeing $\leq \frac{2n}{\varepsilon m}$ vertices

If True, output NO!

Analyzing the Algorithm

- If G is connected, algo. never outputs NO

Analyzing the Algorithm

- If G is connected, algo. never outputs NO
- If G has $\geq \varepsilon m$ connected components,

Analyzing the Algorithm

- If G is connected, algo. never outputs NO
- If G has $\geq \varepsilon m$ connected components, $\geq \frac{\varepsilon m}{2}$ vertices belong to "small" components

Analyzing the Algorithm

- If G is connected, algo. never outputs NO
- If G has $\geq \varepsilon m$ connected components,

$\geq \frac{\varepsilon m}{2}$ vertices belong to "small" components

$\therefore$ Pr [algo. outputting NO] $\geq \frac{\varepsilon m}{2} \times \frac{1}{n}$

- Algo. correct with probability $\geq \frac{\varepsilon m}{2n}$
- Time Complexity ?

- Algo. correct with probability $\geq \frac{\varepsilon m}{2n}$

- Time Complexity ?

  * BFS until seeing $\frac{2n}{\varepsilon m}$ vertices

    can take $\leq \frac{4n^2}{\varepsilon^2 m^2}$ time

- Algo. correct with probability $\geq \frac{\varepsilon m}{2n}$

- Time Complexity ? $\boxed{O\left(\frac{n^2}{\varepsilon^2 m^2}\right)}$

  * BFS until seeing $\frac{2n}{\varepsilon m}$ vertices

can take $\leq \frac{4n^2}{\varepsilon^2 m^2}$ time

- Algo. correct with probability $\geq \frac{\varepsilon m}{2n}$

- Time Complexity ? $O\left(\frac{n^2}{\varepsilon^2 m^2}\right)$

  * BFS until seeing $\frac{2n}{\varepsilon m}$ vertices

    can take $\leq \frac{4n^2}{\varepsilon^2 m^2}$ time

We need a correctness probaility $\geq \frac{2}{3}$!

- Algo. correct with probability $\geq \frac{\varepsilon m}{2n}$

- Time Complexity ? $O\left(\frac{n^2}{\varepsilon^2 m^2}\right)$

  * BFS until seeing $\frac{2n}{\varepsilon m}$ vertices

    can take $\leq \frac{4n^2}{\varepsilon^2 m^2}$ time

Repeat algo. $\Theta\left(\frac{n}{\varepsilon m}\right)$ times independently

$\Rightarrow$ Correctness probability $\geq \frac{2}{3}$.

- Algo. correct with probability $\geq \frac{\varepsilon m}{2n}$

- Time Complexity ? $O\left(\frac{n^3}{\varepsilon^3 m^3}\right)$

  * BFS until seeing $\frac{2n}{\varepsilon m}$ vertices

    can take $\leq \frac{4n^2}{\varepsilon^2 m^2}$ time

Repeat algo. $\Theta\left(\frac{n}{\varepsilon m}\right)$ times independently

$\Rightarrow$ Correctness probability $\geq \frac{2}{3}$.

Theorem: Algo. to ε-test connectedness of n-vertex m-edge graphs with time complexity $O\left(\frac{n^3}{\varepsilon^3 m^3}\right)$

Theorem: Algo. to ε-test connectedness of n-vertex m-edge graphs with time complexity $O\left(\frac{n^3}{\varepsilon^3 m^3}\right)$

$$\Big\Vert$$

$$O\left(\frac{1}{\varepsilon^3 \bar{d}^3}\right)$$

$\bar{d}$: average degree

Theorem: Algo. to ε-test connectedness of n-vertex m-edge graphs with time complexity $O\left(\frac{n^3}{\varepsilon^3 m^3}\right)$

$$= O\left(\frac{1}{\varepsilon^3 \bar{d}^3}\right)$$

Very very small compared to size of input

$\bar{d}$: average degree

CAN WE DO BETTER?

CAN WE DO BETTER?

YES!

Connectedness Testing Algorithm (Goldreich Ron '02)

Given query access to $G = (V, E)$, input ε

- Sample $v \sim V$ uniformly at random
- Check whether v belongs to a component with $\leq \frac{2n}{\varepsilon m}$ vertices by performing a BFS from v until seeing $\leq \frac{2n}{\varepsilon m}$ vertices

If True, output NO!

Connectedness Testing Algorithm

(Levi, Pallavoor, Raskhodnikova, Varma '21)

Given query access to $G = (V, E)$, input ε

- Sample $v \sim V$ uniformly at random
- Check whether v belongs to a component with $\leq \frac{2n}{\varepsilon m}$ vertices by performing a BFS from v until seeing $\leq \frac{2n}{\varepsilon m} \times d_v$ edges
  - d_v → degree of v

If True, output NO!

Analyzing the algorithm

- For a component with $\leq \frac{2n}{\varepsilon m}$ vertices, our BFS "budget" is sufficient in expectation

Analyzing the algorithm

- For a component with $\leq \frac{2n}{\varepsilon m}$ vertices, our BFS "budget" is sufficient in expectation
- Expected Time Complexity

$$\leq \frac{2n}{\varepsilon m} \cdot \mathop{\mathbb{E}}_{v \sim V}[d_v] = O\left(\frac{1}{\varepsilon \bar{d}} \cdot \bar{d}\right)$$

$$= O\left(\frac{1}{\varepsilon}\right)$$

Analyzing the algorithm

- For a component with $\leq \frac{2n}{\varepsilon m}$ vertices, our BFS "budget" is sufficient in expectation

For $\geq \frac{2}{3}$ success probability, $O\left(\frac{1}{\varepsilon^2 \bar{d}}\right)$

- Expected Time Complexity

$$\leq \frac{2n}{\varepsilon m} \cdot \mathop{\mathbb{E}}_{v \sim V}[d_v] = O\left(\frac{1}{\varepsilon \bar{d}} \cdot \bar{d}\right)$$

$$= O\left(\frac{1}{\varepsilon}\right)$$

Theorem: Algo. to ε-test connectedness of n-vertex m-edge graphs with time complexity $O\left(\min\left\{\frac{1}{\varepsilon^3\bar{d}^3}, \frac{1}{\varepsilon^2\bar{d}}\right\}\right)$

- $\frac{1}{\varepsilon^3\bar{d}^3}$ → [Goldreich Ron '02]
- $\frac{1}{\varepsilon^2\bar{d}}$ → our result

$\bar{d}$: average degree

Theorem: Algo. to ε-test connectedness of n-vertex m-edge graphs with time complexity $O\left(\min\left\{\frac{1}{\varepsilon^3\bar{d}^3}, \frac{1}{\varepsilon^2\bar{d}}\right\}\right)$

- $\frac{1}{\varepsilon^3\bar{d}^3}$ → [Goldreich Ron '02]
- $\frac{1}{\varepsilon^2\bar{d}}$ → our result

Better when $\bar{d}$ is small

$\bar{d}$: average degree

Summary

- Sublinear-time computation to tackle big data challenges
- Can solve non-trivial problems in sublinear-time
- Very efficient algorithms to test connectedness of graphs

Summary

- ★ Sublinear-time computation to tackle big data challenges

- ★ Can solve non-trivial problems in sublinear-time

- ★ Very efficient algorithms to test connectedness of graphs

THANK YOU!