Lecture 8: Sublinear Time Algorithms

Rafael Oliveira

University of Waterloo
Cheriton School of Computer Science

rafael.oliveira.teaching@gmail.com

May 26, 2025

Overview

- Introduction
  - Why Sublinear Time Algorithms?
  - Warm-up Problem
- Main Problem
  - Number of Connected Components
- Acknowledgements

How do we handle big data?

Sometimes big data does not come to us all at once (think streaming), but instead we *can query small pieces* of it.

Sometimes big data can also *change over time*, so we need a *robust* answer and/or be able to solve problem quickly multiple times.

How do we handle big data?

Sometimes big data does not come to us all at once (think streaming), but instead we *can query small pieces* of it.

Sometimes big data can also *change over time*, so we need a *robust* answer and/or be able to solve problem quickly multiple times.

- **Social graph:** each person is a node, edges if they are friends.

How do we handle big data?

Sometimes big data does not come to us all at once (think streaming), but instead we *can query small pieces* of it.

Sometimes big data can also *change over time*, so we need a *robust* answer and/or be able to solve problem quickly multiple times.

- **Social graph:** each person is a node, edges if they are friends.
  - Is graph connected?

How do we handle big data?

Sometimes big data does not come to us all at once (think streaming), but instead we *can query small pieces* of it.

Sometimes big data can also *change over time*, so we need a *robust* answer and/or be able to solve problem quickly multiple times.

- **Social graph:** each person is a node, edges if they are friends.
  - Is graph connected?
  - What is the degree of separation? Diameter of graph (6 degrees of separation)

How do we handle big data?

Sometimes big data does not come to us all at once (think streaming), but instead we *can query small pieces* of it.

Sometimes big data can also *change over time*, so we need a *robust* answer and/or be able to solve problem quickly multiple times.

- **Social graph:** each person is a node, edges if they are friends.
  - Is graph connected?
  - What is the degree of separation? Diameter of graph (6 degrees of separation)
- **Program checking:** checking that a computer program works correctly on all/most inputs

How do we handle big data?

Sometimes big data does not come to us all at once (think streaming), but instead we *can query small pieces* of it.

Sometimes big data can also *change over time*, so we need a *robust* answer and/or be able to solve problem quickly multiple times.

- **Social graph:** each person is a node, edges if they are friends.
  - Is graph connected?
  - What is the degree of separation? Diameter of graph (6 degrees of separation)
- **Program checking:** checking that a computer program works correctly on all/most inputs
  - Too many inputs to check your program on!

How do we handle big data?

Sometimes big data does not come to us all at once (think streaming), but instead we *can query small pieces* of it.

Sometimes big data can also *change over time*, so we need a *robust* answer and/or be able to solve problem quickly multiple times.

- **Social graph:** each person is a node, edges if they are friends.
  - Is graph connected?
  - What is the degree of separation? Diameter of graph (6 degrees of separation)
- **Program checking:** checking that a computer program works correctly on all/most inputs
  - Too many inputs to check your program on!
- Many more...

What problems is this used for?

- **Graphs:**

What problems is this used for?

- **Graphs:**
  - diameter
  - # connected components
  - Minimum Spanning Tree
  - Testing bipartiteness
  - Testing clusterability

What problems is this used for?

- **Graphs:**
  - diameter
  - # connected components
  - Minimum Spanning Tree
  - Testing bipartiteness
  - Testing clusterability
- **Functions:**
  - is a function monotone?
  - is function convex?
  - is function linear?

What problems is this used for?

- **Graphs:**
  - diameter
  - # connected components
  - Minimum Spanning Tree
  - Testing bipartiteness
  - Testing clusterability
- **Functions:**
  - is a function monotone?
  - is function convex?
  - is function linear?
- **Distributions:**
  - is distribution uniform?
  - is is independent?

What problems is this used for?

- **Graphs:**
  - diameter
  - $\#$ connected components
  - Minimum Spanning Tree
  - Testing bipartiteness
  - Testing clusterability
- **Functions:**
  - is a function monotone?
  - is function convex?
  - is function linear?
- **Distributions:**
  - is distribution uniform?
  - is is independent?

Connects to *randomized algorithms*, *approximation algorithms*, *parallel algorithms*, *complexity theory*, *statistics*, *learning*

What can we hope to do?

What we *can't* do:

- Can't answer **for all** or **there exists** or **exactly** type statements

What can we hope to do?

What we *can't* do:

- Can't answer **for all** or **there exists** or **exactly** type statements
  - are all individuals connected via friendships?
  - are all individuals connected by at most 6 degrees of separation?
  - is my program correct on all inputs

What can we hope to do?

What we *can't* do:

- Can't answer **for all** or **there exists** or **exactly** type statements
  - are <u>all</u> individuals connected via friendships?
  - are <u>all</u> individuals connected by at most 6 degrees of separation?
  - is my program correct on <u>all inputs</u>

What we *can* do:

- Can answer **for most** or **averages** or **approximate** type statements *with high probability*

What can we hope to do?

What we *can't* do:

- Can't answer **for all** or **there exists** or **exactly** type statements
  - are all individuals connected via friendships?
  - are all individuals connected by at most 6 degrees of separation?
  - is my program correct on all inputs

What we *can* do:

- Can answer **for most** or **averages** or **approximate** type statements *with high probability*
  - are most individuals connected via friendships?
  - are most individuals connected by at most 6 degrees of separation?
  - approximately how many people are left handed?
  - is my program correct on most inputs

What can we hope to do?

What we *can't* do:

- Can't answer **for all** or **there exists** or **exactly** type statements
  - are all individuals connected via friendships?
  - are all individuals connected by at most 6 degrees of separation?
  - is my program correct on all inputs

What we *can* do:

- Can answer **for most** or **averages** or **approximate** type statements *with high probability*
  - are most individuals connected via friendships?
  - are most individuals connected by at most 6 degrees of separation?
  - approximately how many people are left handed?
  - is my program correct on most inputs

Randomized & *Approximate* algorithms.

Sublinear Time Models of Computation

- Random Access Queries

Sublinear Time Models of Computation

- Random Access Queries
  - Can access any word of input in one step
  - How is input represented?

Sublinear Time Models of Computation

- Random Access Queries
  - Can access any word of input in one step
  - How is input represented?
    - Adjacency matrix
    - Adjacency list

Sublinear Time Models of Computation

- Random Access Queries
  - Can access any word of input in one step
  - How is input represented?
    - Adjacency matrix
    - Adjacency list
    - Location

Sublinear Time Models of Computation

- Random Access Queries
  - Can access any word of input in one step
  - How is input represented?
    - Adjacency matrix
    - Adjacency list
    - Location
    - many others...

Sublinear Time Models of Computation

- Random Access Queries
  - Can access any word of input in one step
  - How is input represented?
    - Adjacency matrix
    - Adjacency list
    - Location
    - many others...
- Samples
  - get samples from certain distribution/input at each step

Approximate Diameter of a Point Set

- **Input:** m points and a distance matrix D such that
  - $D_{ij} \leftarrow$ distance from i to j
  - D *symmetric* and satisfies *triangle inequality*

Input given in *adjacency matrix* representation

Approximate Diameter of a Point Set

- **Input:** m points and a distance matrix D such that
  - $D_{ij} \leftarrow$ distance from i to j
  - D *symmetric* and satisfies *triangle inequality*

Input given in *adjacency matrix* representation

- Input size: $N = m^2$

Approximate Diameter of a Point Set

- **Input:** m points and a distance matrix D such that
  - $D_{ij} \leftarrow$ distance from i to j
  - D *symmetric* and satisfies *triangle inequality*

- Input size: $N = m^2$
- Let a, b be indices that *maximize* distance D_{ab}. Then D_{ab} is *diameter*

Approximate Diameter of a Point Set

- **Input:** m points and a distance matrix D such that
  - $D_{ij} \leftarrow$ distance from i to j
  - D *symmetric* and satisfies *triangle inequality*

- Input size: $N = m^2$
- Let a, b be indices that *maximize* distance D_{ab}. Then D_{ab} is *diameter*
- **Output:** Indices k, ℓ such that

$$D_{k\ell} \geq D_{ab}/2$$

2-multiplicative algorithm

Algorithm & Analysis

- Pick k arbitrarily

Algorithm & Analysis

- Pick k arbitrarily
- Pick ℓ to maximize $D_{k\ell}$

Algorithm & Analysis

- Pick k arbitrarily
- Pick ℓ to maximize $D_{k\ell}$
- Output indices k, ℓ

Algorithm & Analysis

- Pick k arbitrarily
- Pick ℓ to maximize $D_{k\ell}$
- Output indices k, ℓ

Why does this work?

Algorithm & Analysis

- Pick k arbitrarily
- Pick ℓ to maximize $D_{k\ell}$
- Output indices k, ℓ

Why does this work?

- Correctness

$$
\begin{aligned}
D_{ab} &\leq D_{ak} + D_{kb} \\
&\leq D_{k\ell} + D_{k\ell} = 2 \cdot D_{k\ell}
\end{aligned}
$$

Algorithm & Analysis

- Pick k arbitrarily
- Pick ℓ to maximize $D_{k\ell}$
- Output indices k, ℓ

Why does this work?

- Correctness

$$\begin{aligned} D_{ab} &\leq D_{ak} + D_{kb} \\ &\leq D_{k\ell} + D_{k\ell} = 2 \cdot D_{k\ell} \end{aligned}$$

- Running time: $O(m) = O(N^{1/2})$

Algorithm & Analysis

- Pick k arbitrarily
- Pick ℓ to maximize $D_{k\ell}$
- Output indices k, ℓ

Why does this work?

- Correctness

$$\begin{aligned} D_{ab} &\leq D_{ak} + D_{kb} \\ &\leq D_{k\ell} + D_{k\ell} = 2 \cdot D_{k\ell} \end{aligned}$$

- Running time: $O(m) = O(N^{1/2})$

Is this the best we can do?

Lower Bound for Approximate Diameter

- Let D be following: distance matrix $D_{i,i} = 0, \ \forall i \in [m]$ and $D_{i,j} = 1$ otherwise

Lower Bound for Approximate Diameter

- Let D be following: distance matrix $D_{i,i} = 0, \ \forall i \in [m]$ and $D_{i,j} = 1$ otherwise
- Let D' be same matrix as D except that for one pair (a, b) we make

$$D'_{ab} = D'_{ba} = 2 - \delta$$

Lower Bound for Approximate Diameter

- Let D be following: distance matrix $D_{i,i} = 0, \ \forall i \in [m]$ and $D_{i,j} = 1$ otherwise
- Let D' be same matrix as D except that for one pair (a, b) we make

$$D'_{ab} = D'_{ba} = 2 - \delta$$

- Check that D' satisfies properties of a distance matrix (thus valid)

Lower Bound for Approximate Diameter

- Let D be following: distance matrix $D_{i,i} = 0, \ \forall i \in [m]$ and $D_{i,j} = 1$ otherwise
- Let D' be same matrix as D except that for one pair (a, b) we make

$$D'_{ab} = D'_{ba} = 2 - \delta$$

- Check that D' satisfies properties of a distance matrix (thus valid)
- **Practice problem:** prove that it would take $\Omega(N)$ time (i.e. number of queries) to decide if diameter is 1 or $2 - \delta$

- Introduction
  - Why Sublinear Time Algorithms?
  - Warm-up Problem
- Main Problem
  - Number of Connected Components
- Acknowledgements

Connected Components

How to approximate number of connected components of a graph:

Connected Components

How to approximate number of connected components of a graph:

- **Input:** graph $G(V, E)$ in *adjacency list* representation. $\epsilon > 0$.

$$n = |V|,\ m = |E|,\ N = m + n$$

- **Output:** if $C \leftarrow \#$ connected components of G, output with probability $\geq 3/4$ C' such that

$$|C' - C| \leq \epsilon n$$

Connected Components

How to approximate number of connected components of a graph:

- **Input:** graph $G(V, E)$ in *adjacency list* representation. $\epsilon > 0$.

$$n = |V|, \; m = |E|, \; N = m + n$$

- **Output:** if $C \leftarrow \#$ connected components of G, output with probability $\geq 3/4$ C' such that

$$|C' - C| \leq \epsilon n$$

- How can we even do this?

Connected Components

How to approximate number of connected components of a graph:

- **Input:** graph $G(V, E)$ in *adjacency list* representation. $\epsilon > 0$.

$$n = |V|, \ m = |E|, \ N = m + n$$

- **Output:** if $C \leftarrow \#$ connected components of G, output with probability $\geq 3/4$ C' such that

$$|C' - C| \leq \epsilon n$$

- How can we even do this?
- Different characterization of $\#$ connected components of graph

Connected Components

How to approximate number of connected components of a graph:

- **Input:** graph $G(V, E)$ in *adjacency list* representation. $\epsilon > 0$.

$$n = |V|, \; m = |E|, \; N = m + n$$

- **Output:** if $C \leftarrow \#$ connected components of G, output with probability $\geq 3/4$ C' such that

$$|C' - C| \leq \epsilon n$$

- How can we even do this?
- Different characterization of $\#$ connected components of graph

Lemma ($\#$ Connected Components)

Let $G(V, E)$ be a graph. For vertex $v \in V$, let $n_v \leftarrow \#$ vertices in connected component of v. Let C be number of connected components of G. Then:

$$C = \sum_{v \in V} \frac{1}{n_v}$$

Connected Components

Naive attempt: sample small number of vertices from G, compute n_v and output normalization.

Connected Components

Naive attempt: sample small number of vertices from G, compute n_v and output normalization.

- **Problem:** just computing n_v may take *linear time* if graph is connected!

Connected Components

Naive attempt: sample small number of vertices from G, compute n_v and output normalization.

- **Problem:** just computing n_v may take *linear time* if graph is connected!
- **Idea:** if n_v large, then $1/n_v$ small and we can drop it!

Connected Components

Naive attempt: sample small number of vertices from G, compute n_v and output normalization.

- **Problem:** just computing n_v may take *linear time* if graph is connected!
- **Idea:** if n_v large, then $1/n_v$ small and we can drop it!

Lemma (Estimating # components)

Let

$$n'_v = \min(n_v, 2/\epsilon)$$

Then

$$\left| \sum_{v \in V} \frac{1}{n_v} - \sum_{v \in V} \frac{1}{n'_v} \right| \leq \frac{\epsilon n}{2}.$$

Connected Components

Naive attempt: sample small number of vertices from G, compute n_v and output normalization.

- **Problem:** just computing n_v may take *linear time* if graph is connected!
- **Idea:** if n_v large, then $1/n_v$ small and we can drop it!

Lemma (Estimating # components)

Let

$$n'_v = \min(n_v, 2/\epsilon)$$

Then

$$\left| \sum_{v \in V} \frac{1}{n_v} - \sum_{v \in V} \frac{1}{n'_v} \right| \le \frac{\epsilon n}{2}.$$

How do we do this estimation?

Sample vertex v and run BFS starting at v, short-cutting if see $2/\epsilon$ vertices.

Connected Components - proof of lemma

Lemma (Estimating # components)

Let

$$n'_v = \min(n_v, 2/\epsilon)$$

Then

$$\left| \sum_{v \in V} \frac{1}{n_v} - \sum_{v \in V} \frac{1}{n'_v} \right| \leq \frac{\epsilon n}{2}.$$

Algorithm

- Choose $s = \Theta(1/\epsilon^2)$ vertices $v_1, \ldots, v_s$ uniformly at random.

Algorithm

- Choose $s = \Theta(1/\epsilon^2)$ vertices $v_1, \ldots, v_s$ uniformly at random.
- Compute n'_{v_i} using BFS
- Return

$$C' = \frac{n}{s} \cdot \sum_{i=1}^{s} \frac{1}{n'_{v_i}}$$

Algorithm

- Choose $s = \Theta(1/\epsilon^2)$ vertices $v_1, \ldots, v_s$ uniformly at random.
- Compute n'_{v_i} using BFS
- Return

$$C' = \frac{n}{s} \cdot \sum_{i=1}^{s} \frac{1}{n'_{v_i}}$$

- **Running Time:**

Algorithm

- Choose $s = \Theta(1/\epsilon^2)$ vertices $v_1, \ldots, v_s$ uniformly at random.
- Compute n'_{v_i} using BFS
- Return

$$C' = \frac{n}{s} \cdot \sum_{i=1}^{s} \frac{1}{n'_{v_i}}$$

- **Running Time:**
  - $\Theta(1/\epsilon^2)$ vertices sampled,

Algorithm

- Choose $s = \Theta(1/\epsilon^2)$ vertices $v_1, \ldots, v_s$ uniformly at random.
- Compute n'_{v_i} using BFS
- Return

$$C' = \frac{n}{s} \cdot \sum_{i=1}^{s} \frac{1}{n'_{v_i}}$$

- **Running Time:**
  - $\Theta(1/\epsilon^2)$ vertices sampled,
  - each run takes $O(1/\epsilon^2)$ time to compute.

Algorithm

- Choose $s = \Theta(1/\epsilon^2)$ vertices $v_1, \ldots, v_s$ uniformly at random.
- Compute n'_{v_i} using BFS
- Return
$$C' = \frac{n}{s} \cdot \sum_{i=1}^{s} \frac{1}{n'_{v_i}}$$

- **Running Time:**
  - $\Theta(1/\epsilon^2)$ vertices sampled,
  - each run takes $O(1/\epsilon^2)$ time to compute.
  - Adding results takes $O(s) = O(1/\epsilon^2)$ time.

Algorithm

- Choose $s = \Theta(1/\epsilon^2)$ vertices $v_1, \ldots, v_s$ uniformly at random.
- Compute n'_{v_i} using BFS
- Return

$$C' = \frac{n}{s} \cdot \sum_{i=1}^{s} \frac{1}{n'_{v_i}}$$

- **Running Time:**
  - $\Theta(1/\epsilon^2)$ vertices sampled,
  - each run takes $O(1/\epsilon^2)$ time to compute.
  - Adding results takes $O(s) = O(1/\epsilon^2)$ time.
- Total running time $O(1/\epsilon^4)$.

Algorithm - Correctness

To prove correctness we need to show that with probability $\geq 3/4$ we have

$$\left| \frac{n}{s} \cdot \sum_{i=1}^{s} \frac{1}{n'_{v_i}} - \sum_{v \in V} \frac{1}{n_v} \right| \leq \epsilon n$$

Algorithm - Correctness

To prove correctness we need to show that with probability $\geq 3/4$ we have

$$\left| \frac{n}{s} \cdot \sum_{i=1}^{s} \frac{1}{n'_{v_i}} - \sum_{v \in V} \frac{1}{n_v} \right| \leq \epsilon n$$

Dividing by n/s on both sides:

$$\left| \sum_{i=1}^{s} \frac{1}{n'_{v_i}} - \frac{s}{n} \cdot \sum_{v \in V} \frac{1}{n_v} \right| \leq \epsilon s$$

Algorithm - Correctness

To prove correctness we need to show that with probability $\geq 3/4$ we have

$$\left| \frac{n}{s} \cdot \sum_{i=1}^{s} \frac{1}{n'_{v_i}} - \sum_{v \in V} \frac{1}{n_v} \right| \leq \epsilon n$$

Dividing by n/s on both sides:

$$\left| \sum_{i=1}^{s} \frac{1}{n'_{v_i}} - \frac{s}{n} \cdot \sum_{v \in V} \frac{1}{n_v} \right| \leq \epsilon s$$

By our previous lemma, and triangle inequality, enough to prove that w.h.p.

$$\left| \sum_{i=1}^{s} \frac{1}{n'_{v_i}} - \frac{s}{n} \cdot \sum_{v \in V} \frac{1}{n'_v} \right| \leq \frac{\epsilon s}{2}$$

Lemma and Triangle Inequality

Lemma (Estimating # components)

Let

$$n'_v = \min(n_v, 2/\epsilon)$$

Then

$$\left| \sum_{v \in V} \frac{1}{n_v} - \sum_{v \in V} \frac{1}{n'_v} \right| \leq \frac{\epsilon n}{2}.$$

Algorithm - Correctness

Want to show that with probability $\geq 3/4$:

$$\left| \sum_{i=1}^{s} \frac{1}{n'_{v_i}} - \frac{s}{n} \cdot \sum_{v \in V} \frac{1}{n'_v} \right| \leq \frac{\epsilon \cdot s}{2}$$

Algorithm - Correctness

Want to show that with probability $\geq 3/4$:

$$\left| \sum_{i=1}^{s} \frac{1}{n'_{v_i}} - \frac{s}{n} \cdot \sum_{v \in V} \frac{1}{n'_v} \right| \leq \frac{\epsilon \cdot s}{2}$$

Theorem (Hoeffding's Inequality)

Let X_i be independent random variables, taking values in $[a_i, b_i]$, $X = \sum_{i=1}^{s} X_i$. Then

$$\Pr[|X - \mathbb{E}[X]| \geq \ell] \leq 2 \cdot \exp\left(-\frac{2\ell^2}{\sum_{i=1}^{s} (b_i - a_i)^2} \right)$$

Algorithm - Correctness

Want to show that with probability $\geq 3/4$:

$$\left| \sum_{i=1}^{s} \frac{1}{n'_{v_i}} - \frac{s}{n} \cdot \sum_{v \in V} \frac{1}{n'_v} \right| \leq \frac{\epsilon \cdot s}{2}$$

Theorem (Hoeffding's Inequality)

Let X_i be independent random variables, taking values in $[a_i, b_i]$, $X = \sum_{i=1}^{s} X_i$. Then

$$\Pr[|X - \mathbb{E}[X]| \geq \ell] \leq 2 \cdot \exp\left(- \frac{2\ell^2}{\sum_{i=1}^{s} (b_i - a_i)^2} \right)$$

Setting parameters of Hoeffing's theorem to our setting:

- $a_i = 0, \ b_i = 1$
- $X_i = 1/n'_v$ with probability $1/n$ (pick vertex uniformly at random)

Algorithm - Correctness

$$X = \sum_{i=1}^{s} X_i \quad \left(= \sum_{i=1}^{s} \frac{1}{n'_{v_i}} \right)$$

Algorithm - Correctness

$$X = \sum_{i=1}^{s} X_i \quad \left(= \sum_{i=1}^{s} \frac{1}{n'_{v_i}} \right)$$

$$\mu := \mathbb{E}[X] = \sum_{i=1}^{s} \mathbb{E}[X_i] = s \cdot \sum_{v \in V} \frac{1}{n'_v} \cdot \frac{1}{n} = \frac{s}{n} \cdot \sum_{v \in V} \frac{1}{n'_v}$$

Algorithm - Correctness

$$X = \sum_{i=1}^{s} X_i \quad \left(= \sum_{i=1}^{s} \frac{1}{n'_{v_i}} \right)$$

$$\mu := \mathbb{E}[X] = \sum_{i=1}^{s} \mathbb{E}[X_i] = s \cdot \sum_{v \in V} \frac{1}{n'_v} \cdot \frac{1}{n} = \frac{s}{n} \cdot \sum_{v \in V} \frac{1}{n'_v}$$

Hoeffding with the parameters from previous slide and $\ell = \epsilon \cdot s/2$:

Theorem (Hoeffding's Inequality)

Let X_i be independent random variables, taking values in $[0, 1]$, $X = \sum_{i=1}^{s} X_i$. Then

$$\Pr[|X - \mu| \geq \epsilon \cdot s/2] \leq 2 \cdot \exp\left(-\epsilon^2 s/2\right)$$

Algorithm - Correctness

$$X = \sum_{i=1}^{s} X_i \quad \left(= \sum_{i=1}^{s} \frac{1}{n'_{v_i}} \right)$$

$$\mu := \mathbb{E}[X] = \sum_{i=1}^{s} \mathbb{E}[X_i] = s \cdot \sum_{v \in V} \frac{1}{n'_v} \cdot \frac{1}{n} = \frac{s}{n} \cdot \sum_{v \in V} \frac{1}{n'_v}$$

Hoeffding with the parameters from previous slide and $\ell = \epsilon \cdot s/2$:

Theorem (Hoeffding's Inequality)

Let X_i be independent random variables, taking values in $[0,1]$, $X = \sum_{i=1}^{s} X_i$. Then

$$\Pr[|X - \mu| \geq \epsilon \cdot s/2] \leq 2 \cdot \exp\left(-\epsilon^2 s/2\right)$$

Since $s = \Theta(1/\epsilon^2)$, the result follows by choosing $s = 8 \cdot (1/\epsilon^2)$

Acknowledgement

- Lecture based largely on Ronitt's notes.
- See Ronitt's notes at `http://people.csail.mit.edu/ronitt/COURSE/F20/Handouts/scribe1.pdf`
- See also her notes for approximate MST `http://people.csail.mit.edu/ronitt/COURSE/F20/Handouts/scribe2.pdf`
- List of open problems in sublinear algorithms `https://sublinear.info/index.php?title=Main_Page`