Sublinear Algorithms for Big Data Analysis

Michael Kapralov

Theory of Computation Lab 4
EPFL

7 September 2017
The age of big data: massive amounts of data collected in various areas of science and technology
Challenges of big data: computational costs

Data produced at a very high rate, is hard to process due to runtime/memory/communication bottlenecks
Challenges of big data: computational costs

Data produced at a very high rate, is hard to process due to runtime/memory/communication bottlenecks

- 50K Google search queries per second
- 9K tweets per second
- 26TB of Internet traffic per second
Challenges of big data: computational costs

Data produced at a very high rate, is hard to process due to runtime/memory/communication bottlenecks

- 50K Google search queries per second
- 9K tweets per second
- 26TB of Internet traffic per second

Sometimes an approximate solution obtained quickly is better than an exact solution that is slow to compute
Challenges of big data: computational costs

Data produced at a very high rate, is hard to process due to runtime/memory/communication bottlenecks

- 50K Google search queries per second
- 9K tweets per second
- 26TB of Internet traffic per second

Sometimes an approximate solution obtained quickly is better than an exact solution that is slow to compute

Fundamental tradeoffs between precision/memory/communication
Challenges of big data: acquisition cost

Sometimes very expensive to **acquire** the data in the first place: how do we use the data we have efficiently?

Magnetic resonance imaging
Theoretical foundations of big data analysis

Classical algorithmic techniques too expensive in context of large data analysis

Need to operate under extremely stringent resource constraints
Theoretical foundations of big data analysis

Classical algorithmic techniques too expensive in context of large data analysis

Need to operate under extremely stringent resource constraints

**Sublinear algorithms**: use resources much smaller than the size of input they operate on
Sublinear algorithms

- **time**
  - faster than reading all input!

- **space**
  - scan through all input but maintain small state

- **communication complexity**
  - data shared by multiple parties, solve problem from sketches

- **sample complexity**
  - access few input locations
Sublinear algorithms

- **time**
  - faster than reading all input!

- **space**
  - scan through all input but maintain small state

- **communication complexity**
  - data shared by multiple parties, solve problem from sketches

- **sample complexity**
  - access few input locations
Sublinear algorithms

- **time**
  - faster than reading all input!

- **space**
  - scan through all input but maintain small state

- **communication complexity**
  - data shared by multiple parties, solve problem from sketches

- **sample complexity**
  - access few input locations
Sublinear algorithms

- **time**
  - faster than reading all input!

- **space**
  - scan through all input but maintain small state

- **communication complexity**
  - data shared by multiple parties, solve problem from sketches

- **sample complexity**
  - access few input locations
Sublinear algorithms

- **time**
  - faster than reading all input!

- **space**
  - scan through all input but maintain small state

- **communication complexity**
  - data shared by multiple parties, solve problem from sketches

- **sample complexity**
  - access few input locations
In this talk

Sublinear

- space
  - scan through all input but maintain small state

- runtime and samples

Approximating matching size from graph streams

Sparse Fourier Transform in $O(k \log N)$ samples and sublinear time
In this talk

Sublinear

- space
  - scan through all input but maintain small state

Approximating matching size from graph streams
Online advertisement

>89% of Google’s revenue in 2014, more than $50 billion
Bipartite graph: advertisers vs queries
Bipartite graph: advertisers vs queries

monthly budgets

SWISS 5 → 3 UNITED
Bipartite graph: advertisers vs queries

Geneva → Paris, 4

queries

Geneva → NYC, 7

advertisers

SWISS

5

monthly budgets

UNITED

3
Bipartite graph: advertisers vs queries

queries

Geneva → Paris, 4

advertisers

Geneva → NYC, 7

5

monthly budgets

3
Bipartite graph: advertisers vs queries

queries

<table>
<thead>
<tr>
<th>Advertisers</th>
<th>Geneva → Paris, 4</th>
<th>Geneva → NYC, 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWISS</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>UNITED</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

monthly budgets
Bipartite graph: advertisers vs queries

Test if a massive graph contains a large matching
Matching

- Let $G = (P, Q, E)$ be a bipartite graph

\[ |P| + |Q| = n, \quad |E| = m \]
Matching

- Let $G = (P, Q, E)$ be a bipartite graph

  $|P| + |Q| = n, |E| = m$

- $M \subseteq E$ is a matching if no two edges in $M$ share an endpoint
Matching

- Let $G = (P, Q, E)$ be a bipartite graph
  
  $|P| + |Q| = n, |E| = m$

- $M \subseteq E$ is a matching if no two edges in $M$ share an endpoint
Matching

- Let $G = (P, Q, E)$ be a bipartite graph
  
  $|P| + |Q| = n, |E| = m$

- $M \subseteq E$ is a matching if no two edges in $M$ share an endpoint
Maximum matching problem

Given a graph $G$, find a matching $M \subseteq E$ of maximum possible cardinality.
Maximum matching problem

Given a graph $G$, find a matching $M \subseteq E$ of maximum possible cardinality.
Maximum matching problem

Given a graph $G$, find a matching $M \subseteq E$ of maximum possible cardinality
Maximum matching problem

One of the most well-studied problems in combinatorial optimization

Solvable in polynomial time assuming that edges can be loaded into memory
(e.g. Hopcroft-Karp, Mądry)
Maximum matching problem

One of the most well-studied problems in combinatorial optimization

Solvable in polynomial time assuming that edges can be loaded into memory
(e.g. Hopcroft-Karp, Mądry)

Space efficient algorithms for matchings?
Maximum matching problem

One of the most well-studied problems in combinatorial optimization

Solvable in polynomial time assuming that edges can be loaded into memory (e.g. Hopcroft-Karp, Mądry)

Space efficient algorithms for matchings?

Streaming model = computational model for small space algorithms
1. Classical streaming and scalable data analytics

2. Graph streaming
Streaming model (classical)

Observe a (very long) stream of data, e.g. IP packets, tweets, search queries....

Task: maintain (approximate) statistics of the stream
Streaming model (classical)

Observe a (very long) stream of data, e.g. IP packets, tweets, search queries....

**Task:** maintain (approximate) statistics of the stream

Number of distinct elements? Most frequent items (heavy hitters)?
Streaming model (classical)

Widely used in practice for scalable data analytics

# of distinct searches on google.com over a time period

# of tweets from a given user each hour
Streaming model (classical)

Widely used in practice for **scalable data analytics**

# of distinct searches on google.com over a time period

# of tweets from a given user each hour

Data stream algorithms for graph problems?
1. Classical streaming and scalable data analytics

2. Graph streaming
Graph streams

A more recent model Feigenbaum-Kannan-McGregor-Suri-Zhang’2004

- input graph $G = (V, E), |V| = n, |E| = m$
- edges of $G$ arrive in an arbitrary order in a stream
- algorithm can only use $n \cdot \text{poly}(\log n)$ space
Graph streams

A more recent model \text{Feigenbaum-Kannan-McGregor-Suri-Zhang'2004}

- input graph $G = (V, E)$, $|V| = n$, $|E| = m$
- edges of $G$ arrive in an arbitrary order in a stream
- algorithm can only use $n \cdot \text{poly}(\log n)$ space
Graph streams

A more recent model Feigenbaum-Kannan-McGregor-Suri-Zhang’2004

- input graph $G = (V, E)$, $|V| = n$, $|E| = m$
- edges of $G$ arrive in an arbitrary order in a stream
- algorithm can only use $n \cdot \text{poly}(\log n)$ space
Graph streams

A more recent model Feigenbaum-Kannan-McGregor-Suri-Zhang’2004

- input graph $G = (V, E)$, $|V| = n$, $|E| = m$
- edges of $G$ arrive in an arbitrary order in a stream
- algorithm can only use $n \cdot \text{poly} (\log n)$ space
Graph streams

A more recent model Feigenbaum-Kannan-McGregor-Suri-Zhang’2004

- input graph $G = (V, E)$, $|V| = n$, $|E| = m$
- edges of $G$ arrive in an arbitrary order in a stream
- algorithm can only use $n \cdot \text{poly}(\log n)$ space
Graph streams

A more recent model Feigenbaum-Kannan-McGregor-Suri-Zhang'2004

- input graph $G = (V, E)$, $|V| = n$, $|E| = m$
- edges of $G$ arrive in an arbitrary order in a stream
- algorithm can only use $n \cdot \text{poly}(\log n)$ space
Graph streams

A more recent model Feigenbaum-Kannan-McGregor-Suri-Zhang’2004

- input graph $G = (V, E)$, $|V| = n$, $|E| = m$
- edges of $G$ arrive in an arbitrary order in a stream
- algorithm can only use $n \cdot \text{poly}(\log n)$ space
Graph streams

A more recent model Feigenbaum-Kannan-McGregor-Suri-Zhang’2004

- input graph $G = (V, E)$, $|V| = n$, $|E| = m$
- edges of $G$ arrive in an arbitrary order in a stream
- algorithm can only use $n \cdot \text{poly}(\log n)$ space
Graph streams

A more recent model Feigenbaum-Kannan-McGregor-Suri-Zhang’2004

- input graph $G = (V, E)$, $|V| = n$, $|E| = m$
- edges of $G$ arrive in an arbitrary order in a stream
- algorithm can only use $n \cdot \text{poly}(\log n)$ space
Graph streams

A more recent model Feigenbaum-Kannan-McGregor-Suri-Zhang’2004

- input graph $G = (V, E)$, $|V| = n$, $|E| = m$
- edges of $G$ arrive in an arbitrary order in a stream
- algorithm can only use $n \cdot \text{poly} (\log n)$ space
Graph streams

A more recent model Feigenbaum-Kannan-McGregor-Suri-Zhang’2004

- input graph $G = (V, E)$, $|V| = n$, $|E| = m$
- edges of $G$ arrive in an arbitrary order in a stream
- algorithm can only use $n \cdot \text{poly}(\log n)$ space
Graph streams

A more recent model Feigenbaum-Kannan-McGregor-Suri-Zhang’2004

- input graph $G = (V, E)$, $|V| = n$, $|E| = m$
- edges of $G$ arrive in an arbitrary order in a stream
- algorithm can only use $n \cdot \text{poly} (\log n)$ space
Graph streams

A more recent model Feigenbaum-Kannan-McGregor-Suri-Zhang’2004

- input graph $G = (V, E)$, $|V| = n$, $|E| = m$
- edges of $G$ arrive in an arbitrary order in a stream
- algorithm can only use $n \cdot \text{poly} (\log n)$ space
Graph streams

A more recent model Feigenbaum-Kannan-McGregor-Suri-Zhang’2004

- input graph $G = (V, E)$, $|V| = n$, $|E| = m$
- edges of $G$ arrive in an arbitrary order in a stream
- algorithm can only use $n \cdot \text{poly}(\log n)$ space
Graph streams

A more recent model Feigenbaum-Kannan-McGregor-Suri-Zhang’2004

- input graph $G = (V, E)$, $|V| = n$, $|E| = m$
- edges of $G$ arrive in an arbitrary order in a stream
- algorithm can only use $n \cdot \text{poly}(\log n)$ space
Graph streams

A more recent model \textit{Feigenbaum-Kannan-McGregor-Suri-Zhang’2004}

- input graph $G = (V, E)$, $|V| = n$, $|E| = m$
- edges of $G$ arrive in an arbitrary order in a stream
- algorithm can only use $n \cdot \text{poly}(\log n)$ space
Graph streams

A more recent model Feigenbaum-Kannan-McGregor-Suri-Zhang’2004

- input graph \( G = (V, E), |V| = n, |E| = m \)
- edges of \( G \) arrive in an arbitrary order in a stream
- algorithm can only use \( n \cdot \text{poly}(\log n) \) space
Graph streams

A more recent model Feigenbaum-Kannan-McGregor-Suri-Zhang’2004

- input graph $G = (V, E)$, $|V| = n$, $|E| = m$
- edges of $G$ arrive in an arbitrary order in a stream
- algorithm can only use $n \cdot \text{poly}(\log n)$ space
Graph streams

A more recent model Feigenbaum-Kannan-McGregor-Suri-Zhang’2004

- input graph $G = (V, E)$, $|V| = n$, $|E| = m$
- edges of $G$ arrive in an arbitrary order in a stream
- algorithm can only use $n \cdot \text{poly}(\log n)$ space
Graph streams

A more recent model Feigenbaum-Kannan-McGregor-Suri-Zhang’2004

- input graph $G = (V, E)$, $|V| = n$, $|E| = m$
- edges of $G$ arrive in an arbitrary order in a stream
- algorithm can only use $n \cdot \text{poly}(\log n)$ space

- shortest path distances (spanners)
- graph sparsification
- matchings
- counting subgraphs
- …
Graph streams

A more recent model Feigenbaum-Kannan-McGregor-Suri-Zhang’2004

- input graph $G = (V, E)$, $|V| = n$, $|E| = m$
- edges of $G$ arrive in an arbitrary order in a stream
- algorithm can only use $n \cdot \text{poly}(\log n)$ space

- shortest path distances (spanners)
- graph sparsification
- matchings
- counting subgraphs
- ...

$\Omega(n)$ space required
Is it possible to approximate matching size in $O(n)$ space?

Theorem (K.-Khanna-Sudan'14) YES
Is it possible to approximate matching size in $o(n)$ space?
Is it possible to approximate matching size in $o(n)$ space?

Theorem (K.-Khanna-Sudan’14)

YES
Theorem (K.-Khanna-Sudan’14)

Can approximate matching size in a graph on n nodes to factor $\text{poly}(\log n)$ using a single pass over a random stream of edges and $\text{poly}(\log n)$ space.

Note: assume that edges arrive in a random order
(Edges often partitioned using a random hash function)
Theorem (K.-Khanna-Sudan’14)

Can approximate matching size in a graph on $n$ nodes to factor $\text{poly}(\log n)$ using a single pass over a random stream of edges and $\text{poly}(\log n)$ space.

Note: assume that edges arrive in a random order
   (Edges often partitioned using a random hash function)

Space complexity comparable to distinct elements problem!
Theorem (K.-Khanna-Sudan’14)
Can approximate matching size in a graph on $n$ nodes to factor $\text{poly}(\log n)$ using a single pass over a random stream of edges and $\text{poly}(\log n)$ space.

Note: assume that edges arrive in a random order
(Edges often partitioned using a random hash function)

Space complexity comparable to distinct elements problem!

The first streaming algorithm for a graph problem that uses $o(n)$ space.
Theorem (K.-Khanna-Sudan’14)

Can approximate matching size in a graph on $n$ nodes to factor $\text{poly}(\log n)$ using a single pass over a random stream of edges and $\text{poly}(\log n)$ space.

Note: assume that edges arrive in a random order
(Edges often partitioned using a random hash function)

Space complexity comparable to distinct elements problem!

The first streaming algorithm for a graph problem that uses $o(n)$ space.

Solutions to other graph problems in $o(n)$ space?
In this talk

Sublinear

- space
  - scan through all input but maintain small state

- runtime and samples

Approximating matching size from graph streams

Sparse Fourier Transform in $O(k \log N)$ samples and sublinear time
In this talk:

- Sublinear runtime and samples

Sparse Fourier Transform in $O(k \log N)$ samples and sublinear time
Given $x \in \mathbb{C}^N$, compute the Discrete Fourier Transform (DFT) of $x$:

$$\hat{x}_i = \frac{1}{N} \sum_{j \in [N]} x_j \cdot \omega^{-ij},$$

where $\omega = e^{2\pi i / N}$ is the $N$-th root of unity.
Given $x \in \mathbb{C}^N$, compute the Discrete Fourier Transform (DFT) of $x$:

$$\hat{x}_i = \frac{1}{N} \sum_{j \in [N]} x_j \cdot \omega^{-ij},$$

where $\omega = e^{2\pi i/N}$ is the $N$-th root of unity.

Assume that $N$ is a power of 2.
Given $x \in \mathbb{C}^N$, compute the Discrete Fourier Transform (DFT) of $x$:
\[
\hat{x}_i = \frac{1}{N} \sum_{j \in [N]} x_j \cdot \omega^{-ij},
\]
where $\omega = e^{2\pi i / N}$ is the $N$-th root of unity.

Assume that $N$ is a power of 2.
Sparse approximations

Given $x$, compute $\hat{x}$, then keep top $k$ coefficients only for $k \ll N$

Used in image and video compression schemes
(e.g. JPEG, MPEG)
Sparse approximations

Given $x$, compute $\hat{x}$, then keep top $k$ coefficients only for $k \ll N$

Used in image and video compression schemes (e.g. JPEG, MPEG)
Computing approximation fast

Basic approach:

- FFT computes $\hat{x}$ from $x$ in $O(N\log N)$ time.
- Compute top $k$ coefficients in $O(N)$ time.
Computing approximation fast

**Basic approach:**

- FFT computes $\hat{x}$ from $x$ in $O(N \log N)$ time
- compute top $k$ coefficients in $O(N)$ time.

**Sparse FFT:**

- directly computes $k$ largest coefficients of $\hat{x}$ (approximately)
- Running time $k \log^{O(1)} N$
- Sublinear time!
Sublinear algorithms – a perfect field to apply mathematics to important practical problems
Sublinear algorithms – a perfect field to apply mathematics to important practical problems

Very diverse field: from graph algorithms to signal processing and beyond
Sublinear algorithms – a perfect field to apply mathematics to important practical problems

Very diverse field: from graph algorithms to signal processing and beyond

Questions?