Crash Course in Data Stream Theory

Part 2: Graphs, Geometry, and Future Directions

Andrew McGregor
University of Massachusetts Amherst

Outline

Outline

Graph Streams and Geometric Streams

- Graph Streams: Stream of edges $E = \{e_1, e_2, \ldots, e_m\}$ describe a graph G on n nodes. Estimate properties of G.

Graph Streams and Geometric Streams

- Graph Streams: Stream of edges $E = \{e_1, e_2, \ldots, e_m\}$ describe a graph G on n nodes. Estimate properties of G.
- Geometric Streams: Stream of points $P = \{p_1, p_2, \ldots, p_m\}$ from some metric space $(\mathcal{X}, d)$, e.g., $\mathbb{R}^t$. Estimate properties of P.

Outline

Warm-Up: Connectivity

- *Thm:* Can determine if a graph is connected in $O(n \log n)$ space.

Warm-Up: Connectivity

- *Thm:* Can determine if a graph is connected in $O(n \log n)$ space.
- *Algorithm:*
  1. Maintain label $\ell(u)$ for each node u where labels are initially distinct

Warm-Up: Connectivity

- *Thm:* Can determine if a graph is connected in $O(n \log n)$ space.
- *Algorithm:*
  1. Maintain label $\ell(u)$ for each node u where labels are initially distinct
  2. On seeing edge (u, v) with $\ell(u) \neq \ell(v)$,

$$\ell(w) \leftarrow \ell(u) \quad \text{for all } w \text{ with } \ell(w) = \ell(v)$$

Warm-Up: Connectivity

- *Thm:* Can determine if a graph is connected in $O(n \log n)$ space.
- *Algorithm:*
  1. Maintain label $\ell(u)$ for each node u where labels are initially distinct
  2. On seeing edge (u, v) with $\ell(u) \neq \ell(v)$,

$$\ell(w) \leftarrow \ell(u) \quad \text{for all } w \text{ with } \ell(w) = \ell(v)$$

  3. The graph is connected iff every node ends up with the same label

Warm-Up: Connectivity

- *Thm:* Can determine if a graph is connected in $O(n \log n)$ space.
- *Algorithm:*
  1. Maintain label $\ell(u)$ for each node u where labels are initially distinct
  2. On seeing edge (u, v) with $\ell(u) \neq \ell(v)$,
  $$\ell(w) \leftarrow \ell(u) \quad \text{for all } w \text{ with } \ell(w) = \ell(v)$$
  3. The graph is connected iff every node ends up with the same label
  4. If we collect (u, v) when $\ell(u) \neq \ell(v)$ we maintain a spanning forest

Warm-Up: Connectivity

- *Thm:* Can determine if a graph is connected in $O(n \log n)$ space.
- *Algorithm:*
  1. Maintain label $\ell(u)$ for each node u where labels are initially distinct
  2. On seeing edge (u, v) with $\ell(u) \neq \ell(v)$,

$$\ell(w) \leftarrow \ell(u) \quad \text{for all } w \text{ with } \ell(w) = \ell(v)$$

  3. The graph is connected iff every node ends up with the same label
  4. If we collect (u, v) when $\ell(u) \neq \ell(v)$ we maintain a spanning forest
- Can do something similar to determine if graph is bipartite

Warm-Up: Connectivity

- *Thm:* Can determine if a graph is connected in $O(n \log n)$ space.
- *Algorithm:*
  1. Maintain label $\ell(u)$ for each node u where labels are initially distinct
  2. On seeing edge (u, v) with $\ell(u) \neq \ell(v)$,

  $$\ell(w) \leftarrow \ell(u) \quad \text{for all } w \text{ with } \ell(w) = \ell(v)$$

  3. The graph is connected iff every node ends up with the same label
  4. If we collect (u, v) when $\ell(u) \neq \ell(v)$ we maintain a spanning forest
- Can do something similar to determine if graph is bipartite
- Most graph problems require space roughly proportional to the number of nodes... called the *"semi-streaming space restriction"*

Sparsify the graph as it arrives

- When an edge arrives, only store it if it satisfies some condition

Sparsify the graph as it arrives

- When an edge arrives, only store it if it satisfies some condition
- *Graph Sparsfiers:* Condition maintains $\tilde{O}(n\epsilon^{-2})$ edges but the resulting graph preserves all cuts up to a $1+\epsilon$ factor

Sparsify the graph as it arrives

- When an edge arrives, only store it if it satisfies some condition
- *Graph Sparsfiers:* Condition maintains $\tilde{O}(n\epsilon^{-2})$ edges but the resulting graph preserves all cuts up to a $1+\epsilon$ factor
- *Matchings:* Condition maintains $\tilde{O}(n)$ edges preserves the maximum weight matching up to a constant factor

Sparsify the graph as it arrives

- When an edge arrives, only store it if it satisfies some condition
- *Graph Sparsfiers:* Condition maintains $\tilde{O}(n\epsilon^{-2})$ edges but the resulting graph preserves all cuts up to a $1+\epsilon$ factor
- *Matchings:* Condition maintains $\tilde{O}(n)$ edges preserves the maximum weight matching up to a constant factor
- *Graph Spanners:* Condition maintains $\tilde{O}(n^{1+1/t})$ edges but the resulting graph preserves all graph distances up to a factor $2t-1$

Spanners and Distance Estimation

- The edges define a shortest path graph metric $d_G : V \times V \to \mathbb{N}$.

Spanners and Distance Estimation

- The edges define a shortest path graph metric $d_G : V \times V \rightarrow \mathbb{N}$.
- An α-spanner of a graph $G = (V, E)$ is a subgraph $H = (V, E')$ such that for all u, v,

$$d_G(u, v) \leq d_H(u, v) \leq \alpha d_G(u, v)$$

Spanners and Distance Estimation

- The edges define a shortest path graph metric $d_G : V \times V \rightarrow \mathbb{N}$.
- An α-spanner of a graph $G = (V, E)$ is a subgraph $H = (V, E')$ such that for all u, v,

$$d_G(u, v) \leq d_H(u, v) \leq \alpha d_G(u, v)$$

- *Thm:* Can construct a $2t - 1$ spanner in $\tilde{O}(n^{1+1/t})$ space.

Spanners and Distance Estimation

- The edges define a shortest path graph metric $d_G : V \times V \to \mathbb{N}$.
- An α-spanner of a graph $G = (V, E)$ is a subgraph $H = (V, E')$ such that for all u, v,

$$d_G(u, v) \leq d_H(u, v) \leq \alpha d_G(u, v)$$

- *Thm:* Can construct a $2t - 1$ spanner in $\tilde{O}(n^{1+1/t})$ space.
- *Algorithm:*
  1. Let E' be initially empty

Spanners and Distance Estimation

- The edges define a shortest path graph metric $d_G : V \times V \rightarrow \mathbb{N}$.
- An α-spanner of a graph $G = (V, E)$ is a subgraph $H = (V, E')$ such that for all u, v,

$$d_G(u, v) \leq d_H(u, v) \leq \alpha d_G(u, v)$$

- *Thm:* Can construct a $2t - 1$ spanner in $\tilde{O}(n^{1+1/t})$ space.
- *Algorithm:*
  1. Let E' be initially empty
  2. On seeing (u, v), $E' \leftarrow E' \cup (u, v)$ if $d_H(u, v) > 2t - 1$

Spanners and Distance Estimation

- The edges define a shortest path graph metric $d_G : V \times V \to \mathbb{N}$.
- An α-spanner of a graph $G = (V, E)$ is a subgraph $H = (V, E')$ such that for all u, v,

$$d_G(u, v) \leq d_H(u, v) \leq \alpha d_G(u, v)$$

- *Thm:* Can construct a $2t - 1$ spanner in $\tilde{O}(n^{1+1/t})$ space.
- *Algorithm:*
  1. Let E' be initially empty
  2. On seeing (u, v), $E' \leftarrow E' \cup (u, v)$ if $d_H(u, v) > 2t - 1$
- *Analysis:*
  1. Every distance has grown by at most a factor $2t - 1$

Spanners and Distance Estimation

- The edges define a shortest path graph metric $d_G : V \times V \rightarrow \mathbb{N}$.
- An α-spanner of a graph $G = (V, E)$ is a subgraph $H = (V, E')$ such that for all u, v,

$$d_G(u, v) \leq d_H(u, v) \leq \alpha d_G(u, v)$$

- *Thm:* Can construct a $2t - 1$ spanner in $\tilde{O}(n^{1+1/t})$ space.
- *Algorithm:*
  1. Let E' be initially empty
  2. On seeing (u, v), $E' \leftarrow E' \cup (u, v)$ if $d_H(u, v) > 2t - 1$
- *Analysis:*
  1. Every distance has grown by at most a factor $2t - 1$
  2. $|E'| = \tilde{O}(n^{1+1/t})$ because it's a graph with no cycles of length $\leq 2t$

Spanners and Distance Estimation

- The edges define a shortest path graph metric $d_G : V \times V \to \mathbb{N}$.
- An α-spanner of a graph $G = (V, E)$ is a subgraph $H = (V, E')$ such that for all u, v,

$$d_G(u, v) \leq d_H(u, v) \leq \alpha d_G(u, v)$$

- *Thm:* Can construct a $2t - 1$ spanner in $\tilde{O}(n^{1+1/t})$ space.
- *Algorithm:*
  1. Let E' be initially empty
  2. On seeing (u, v), $E' \leftarrow E' \cup (u, v)$ if $d_H(u, v) > 2t - 1$
- *Analysis:*
  1. Every distance has grown by at most a factor $2t - 1$
  2. $|E'| = \tilde{O}(n^{1+1/t})$ because it's a graph with no cycles of length $\leq 2t$
- Above algorithm is rather slow but faster algorithms exist

Outline

k-center

- Given a stream of distinct points $X = \{p_1, \ldots, p_n\}$ from a metric space $(\mathcal{X}, d)$, find the set of k points $Y \subset X$ that minimizes:

$$\max_i \min_{y \in Y} d(p_i, y)$$

k-center

- Given a stream of distinct points $X = \{p_1, \ldots, p_n\}$ from a metric space $(\mathcal{X}, d)$, find the set of k points $Y \subset X$ that minimizes:

$$\max_i \min_{y \in Y} d(p_i, y)$$

- Can find 2 approx. in $\tilde{O}(k)$ space if you know $\textsc{opt}$ ahead of time.

k-center

- Given a stream of distinct points $X = \{p_1, \ldots, p_n\}$ from a metric space $(\mathcal{X}, d)$, find the set of k points $Y \subset X$ that minimizes:

$$\max_i \min_{y \in Y} d(p_i, y)$$

- Can find 2 approx. in $\tilde{O}(k)$ space if you know OPT ahead of time.
- Can find $(2+\epsilon)$ approx. in $\tilde{O}(k\epsilon^{-1}\log(a/b))$ space if you know

$$a \leq \mathrm{OPT} \leq b$$

k-center

- Given a stream of distinct points $X = \{p_1, \ldots, p_n\}$ from a metric space $(\mathcal{X}, d)$, find the set of k points $Y \subset X$ that minimizes:

$$\max_i \min_{y \in Y} d(p_i, y)$$

- Can find 2 approx. in $\tilde{O}(k)$ space if you know OPT ahead of time.
- Can find $(2 + \epsilon)$ approx. in $\tilde{O}(k\epsilon^{-1} \log(a/b))$ space if you know

$$a \leq \text{OPT} \leq b$$

- *Thm:* $(2 + \epsilon)$ approx. in $\tilde{O}(k\epsilon^{-1} \log \epsilon^{-1})$ space.

k-center: Algorithm and Analysis

- Consider first $k+1$ points: this gives a lower bound a on OPT.

k-center: Algorithm and Analysis

- Consider first $k+1$ points: this gives a lower bound a on $\textsc{opt}$.
- Instantiate basic algorithm with guesses

$$\ell_1 = a,\ \ell_2 = (1+\epsilon)a,\ \ell_3 = (1+\epsilon)^2 a, \ldots\ \ell_{1+t} = O(\epsilon^{-1})a$$

k-center: Algorithm and Analysis

- Consider first $k + 1$ points: this gives a lower bound a on OPT.
- Instantiate basic algorithm with guesses

$$\ell_1 = a,\ \ell_2 = (1+\epsilon)a,\ \ell_3 = (1+\epsilon)^2 a, \ldots\ \ell_{1+t} = O(\epsilon^{-1})a$$

- Say instantiation goes bad if it tries to open $(k+1)$-th center

k-center: Algorithm and Analysis

- Consider first $k+1$ points: this gives a lower bound a on $\textsc{opt}$.
- Instantiate basic algorithm with guesses

$$\ell_1 = a,\ \ell_2 = (1+\epsilon)a,\ \ell_3 = (1+\epsilon)^2 a, \ldots\ \ell_{1+t} = O(\epsilon^{-1})a$$

- Say instantiation goes bad if it tries to open $(k+1)$-th center
- If instantiation for guess ℓ goes bad when processing $(j+1)$-th point

k-center: Algorithm and Analysis

- Consider first $k+1$ points: this gives a lower bound a on $\textsc{opt}$.
- Instantiate basic algorithm with guesses

$$\ell_1 = a,\ \ell_2 = (1+\epsilon)a,\ \ell_3 = (1+\epsilon)^2 a, \ldots\ \ell_{1+t} = O(\epsilon^{-1})a$$

- Say instantiation goes bad if it tries to open $(k+1)$-th center
- If instantiation for guess ℓ goes bad when processing $(j+1)$-th point
  - Let $q_1, \ldots, q_k$ be centers chosen so far.

k-center: Algorithm and Analysis

- Consider first $k+1$ points: this gives a lower bound a on OPT.
- Instantiate basic algorithm with guesses

$$\ell_1 = a,\ \ell_2 = (1+\epsilon)a,\ \ell_3 = (1+\epsilon)^2 a, \ldots\ \ell_{1+t} = O(\epsilon^{-1})a$$

- Say instantiation goes bad if it tries to open $(k+1)$-th center
- If instantiation for guess ℓ goes bad when processing $(j+1)$-th point
  - Let $q_1, \ldots, q_k$ be centers chosen so far.
  - Then $p_1, \ldots, p_j$ are all at most 2ℓ from a q_i.

k-center: Algorithm and Analysis

- Consider first $k+1$ points: this gives a lower bound a on $\textsc{opt}$.
- Instantiate basic algorithm with guesses

$$\ell_1 = a,\ \ell_2 = (1+\epsilon)a,\ \ell_3 = (1+\epsilon)^2 a, \ldots\ \ell_{1+t} = O(\epsilon^{-1})a$$

- Say instantiation goes bad if it tries to open $(k+1)$-th center
- If instantiation for guess ℓ goes bad when processing $(j+1)$-th point
  - Let $q_1, \ldots, q_k$ be centers chosen so far.
  - Then $p_1, \ldots, p_j$ are all at most 2ℓ from a q_i.
  - Optimum for $\{q_1, \ldots, q_k, p_{j+1}, \ldots, p_n\}$ is at most $\textsc{opt} + 2\ell$.

k-center: Algorithm and Analysis

- Consider first $k + 1$ points: this gives a lower bound a on $\textsc{opt}$.
- Instantiate basic algorithm with guesses

$$\ell_1 = a,\ \ell_2 = (1+\epsilon)a,\ \ell_3 = (1+\epsilon)^2 a, \ldots\ \ell_{1+t} = O(\epsilon^{-1})a$$

- Say instantiation goes bad if it tries to open $(k+1)$-th center
- If instantiation for guess ℓ goes bad when processing $(j+1)$-th point
  - Let $q_1, \ldots, q_k$ be centers chosen so far.
  - Then $p_1, \ldots, p_j$ are all at most 2ℓ from a q_i.
  - Optimum for $\{q_1, \ldots, q_k, p_{j+1}, \ldots, p_n\}$ is at most $\textsc{opt} + 2\ell$.
- Hence, for an instantiation with guess $2\ell/\epsilon$ only incurs a small error if we use $\{q_1, \ldots, q_k, p_{j+1}, \ldots, p_n\}$ rather than $\{p_1, \ldots, p_n\}$.

Other computational geometry problems

- Fixed-dimensional linear programming
- Minimum enclosing balls
- Convex hulls
- Diameter
- Clustering with other objective functions

Outline

Triangles

- Given a stream of edges, estimate the number of triangles T_3 up to a factor $(1+\epsilon)$ with probability $1-\delta$ given promise that $T_3 > t$.

Triangles

- Given a stream of edges, estimate the number of triangles T_3 up to a factor $(1+\epsilon)$ with probability $1-\delta$ given promise that $T_3 > t$.
- *Thm:* $\Omega(n^2)$ space required to determine if $t = 0$ (with $\delta = 1/3$).

Triangles

- Given a stream of edges, estimate the number of triangles T_3 up to a factor $(1+\epsilon)$ with probability $1-\delta$ given promise that $T_3 > t$.
- *Thm:* $\Omega(n^2)$ space required to determine if $t = 0$ (with $\delta = 1/3$).
- *Thm:* $\tilde{O}(\epsilon^{-2}(nm/t))$ space is sufficient.

Lower Bound

- *Thm:* $\Omega(n^2)$ space required to determine if $T_3 \neq 0$

Lower Bound

- *Thm:* $\Omega(n^2)$ space required to determine if $T_3 \neq 0$
- *Analysis:*
  1. Suppose Alice has $n \times n$ binary matrix A, Bob has $n \times n$ binary matrix B. Is $A_{ij} = B_{ij} = 1$ for some (i, j)?

Lower Bound

- *Thm:* $\Omega(n^2)$ space required to determine if $T_3 \neq 0$
- *Analysis:*
  1. Suppose Alice has $n \times n$ binary matrix A, Bob has $n \times n$ binary matrix B. Is $A_{ij} = B_{ij} = 1$ for some (i, j)?
  2. Problem requires $\Omega(n^2)$ bits of communication

Lower Bound

- *Thm:* $\Omega(n^2)$ space required to determine if $T_3 \neq 0$
- *Analysis:*
  1. Suppose Alice has $n \times n$ binary matrix A, Bob has $n \times n$ binary matrix B. Is $A_{ij} = B_{ij} = 1$ for some (i, j)?
  2. Problem requires $\Omega(n^2)$ bits of communication
  3. Consider graph $G = (V, E)$ with

$$V = \{v_1, \ldots, v_n, u_1, \ldots, u_n, w_1, \ldots, w_n\} \text{ and } E = \{(v_i, u_i) : i \in [n]\}$$

Lower Bound

- *Thm:* $\Omega(n^2)$ space required to determine if $T_3 \neq 0$
- *Analysis:*
  1. Suppose Alice has $n \times n$ binary matrix A, Bob has $n \times n$ binary matrix B. Is $A_{ij} = B_{ij} = 1$ for some (i, j)?
  2. Problem requires $\Omega(n^2)$ bits of communication
  3. Consider graph $G = (V, E)$ with

     $$V = \{v_1, \ldots, v_n, u_1, \ldots, u_n, w_1, \ldots, w_n\} \text{ and } E = \{(v_i, u_i) : i \in [n]\}$$

  4. Alice emulates streams algorithm on G and edges $\{(u_i, w_j) : A_{ij} = 1\}$

Lower Bound

- *Thm:* $\Omega(n^2)$ space required to determine if $T_3 \neq 0$
- *Analysis:*
  1. Suppose Alice has $n \times n$ binary matrix A, Bob has $n \times n$ binary matrix B. Is $A_{ij} = B_{ij} = 1$ for some (i, j)?
  2. Problem requires $\Omega(n^2)$ bits of communication
  3. Consider graph $G = (V, E)$ with

     $$V = \{v_1, \ldots, v_n, u_1, \ldots, u_n, w_1, \ldots, w_n\} \text{ and } E = \{(v_i, u_i) : i \in [n]\}$$

  4. Alice emulates streams algorithm on G and edges $\{(u_i, w_j) : A_{ij} = 1\}$
  5. Sends the memory state of the algorithm to Bob

Lower Bound

- *Thm:* $\Omega(n^2)$ space required to determine if $T_3 \neq 0$
- *Analysis:*
  1. Suppose Alice has $n \times n$ binary matrix A, Bob has $n \times n$ binary matrix B. Is $A_{ij} = B_{ij} = 1$ for some (i, j)?
  2. Problem requires $\Omega(n^2)$ bits of communication
  3. Consider graph $G = (V, E)$ with

     $$V = \{v_1, \ldots, v_n, u_1, \ldots, u_n, w_1, \ldots, w_n\} \text{ and } E = \{(v_i, u_i) : i \in [n]\}$$

  4. Alice emulates streams algorithm on G and edges $\{(u_i, w_j) : A_{ij} = 1\}$
  5. Sends the memory state of the algorithm to Bob
  6. Bob continues algorithm on edges $\{(v_i, w_j) : B_{ij} = 1\}$

Lower Bound

- *Thm:* $\Omega(n^2)$ space required to determine if $T_3 \neq 0$
- *Analysis:*
  1. Suppose Alice has $n \times n$ binary matrix A, Bob has $n \times n$ binary matrix B. Is $A_{ij} = B_{ij} = 1$ for some (i, j)?
  2. Problem requires $\Omega(n^2)$ bits of communication
  3. Consider graph $G = (V, E)$ with

     $$V = \{v_1, \ldots, v_n, u_1, \ldots, u_n, w_1, \ldots, w_n\} \text{ and } E = \{(v_i, u_i) : i \in [n]\}$$

  4. Alice emulates streams algorithm on G and edges $\{(u_i, w_j) : A_{ij} = 1\}$
  5. Sends the memory state of the algorithm to Bob
  6. Bob continues algorithm on edges $\{(v_i, w_j) : B_{ij} = 1\}$
  7. Memory is $\Omega(n^2)$ bits since $T_3 > 0$ iff $A_{ij} = B_{ij} = 1$ for some i, j

An Algorithm

- *Thm:* $\tilde{O}(\epsilon^{-2}(nm/t))$ space is sufficient if $T_3 \geq t$.

An Algorithm

- *Thm:* $\tilde{O}(\epsilon^{-2}(nm/t))$ space is sufficient if $T_3 \geq t$.
- *Algorithm:*
  - Pick an edge $e_i = (u, v)$ uniformly at random from the stream.

An Algorithm

- *Thm:* $\tilde{O}(\epsilon^{-2}(nm/t))$ space is sufficient if $T_3 \geq t$.
- *Algorithm:*
  - Pick an edge $e_i = (u, v)$ uniformly at random from the stream.
  - Pick w uniformly at random from $V \setminus \{u, v\}$

An Algorithm

- *Thm:* $\tilde{O}(\epsilon^{-2}(nm/t))$ space is sufficient if $T_3 \geq t$.
- *Algorithm:*
  - Pick an edge $e_i = (u, v)$ uniformly at random from the stream.
  - Pick w uniformly at random from $V \setminus \{u, v\}$
  - If $e_j = (u, w)$, $e_k = (v, w)$ for $j, k > i$ exist return $3m(n-2)$; else 0.

An Algorithm

- *Thm:* $\tilde{O}(\epsilon^{-2}(nm/t))$ space is sufficient if $T_3 \geq t$.
- *Algorithm:*
  - Pick an edge $e_i = (u, v)$ uniformly at random from the stream.
  - Pick w uniformly at random from $V \setminus \{u, v\}$
  - If $e_j = (u, w)$, $e_k = (v, w)$ for $j, k > i$ exist return $3m(n-2)$; else 0.
- *Analysis:*
  - Expected outcome of algorithm is T_3

An Algorithm

- *Thm:* $\tilde{O}(\epsilon^{-2}(nm/t))$ space is sufficient if $T_3 \geq t$.
- *Algorithm:*
  - Pick an edge $e_i = (u, v)$ uniformly at random from the stream.
  - Pick w uniformly at random from $V \setminus \{u, v\}$
  - If $e_j = (u, w)$, $e_k = (v, w)$ for $j, k > i$ exist return $3m(n-2)$; else 0.
- *Analysis:*
  - Expected outcome of algorithm is T_3
  - Repeat $O(\epsilon^{-2}(mn/t))$ times in parallel and average

Outline

Random Order Streams and Space-Efficient Sampling

- Past work assumes stream is ordered by an all-powerful adversary

Random Order Streams and Space-Efficient Sampling

- Past work assumes stream is ordered by an all-powerful adversary
- Can we design smaller-space algorithms if we assume random order?

Random Order Streams and Space-Efficient Sampling

- Past work assumes stream is ordered by an all-powerful adversary
- Can we design smaller-space algorithms if we assume random order?
- Perform average-case analysis to understand performance in practice

Random Order Streams and Space-Efficient Sampling

- Past work assumes stream is ordered by an all-powerful adversary
- Can we design smaller-space algorithms if we assume random order?
- Perform average-case analysis to understand performance in practice
- What about processing stochastically generated streams such as a stream of i.i.d. samples? Learning algorithms...

Probabilistic Data

- Previous work assumes all input is specified exactly

Probabilistic Data

- Previous work assumes all input is specified exactly
- What if each data item has some inherent uncertainty

Probabilistic Data

- Previous work assumes all input is specified exactly
- What if each data item has some inherent uncertainty
- Can we compute the expected value or distribution of aggregates?

Annotations and Stream Verification

- Suppose we have help processing the stream by a third party who "annotates" the stream

$$\langle x_1, x_2, x_3, x_4, \ldots, x_m \rangle \longrightarrow \langle x_1, x_2, a_2, x_3, x_4, \ldots, x_m, a_m \rangle$$

Annotations and Stream Verification

- Suppose we have help processing the stream by a third party who "annotates" the stream

$$\langle x_1, x_2, x_3, x_4, \ldots, x_m \rangle \longrightarrow \langle x_1, x_2, {\color{red}a_2}, x_3, x_4, \ldots, x_m, {\color{red}a_m} \rangle$$

- Can we reduce our space use if assisted by an honest helper but not be misled by a malicious helper?

Thanks!

- *Blog:* `http://polylogblog.wordpress.com`
- *Lectures:* Piotr Indyk, MIT

  `http://stellar.mit.edu/S/course/6/fa07/6.895/`
- *Books:*

  "Data Streams: Algorithms and Applications"
  S. Muthukrishnan (2005)

  "Algorithms and Complexity of Stream Processing"
  A. McGregor, S. Muthukrishnan (forthcoming)