Streaming Graph Computations with a Helpful Advisor

Justin Thaler Graham Cormode and Michael Mitzenmacher



Data Streaming Model

- Stream: m elements from universe of size n
 - e.g., $S = \langle x_1, x_2, \dots, x_m \rangle = 3, 5, 3, 7, 5, 4, 8, 7, 5, 4, 8, 6, 3, 2, \dots$
- Goal: Compute a function of stream, e.g., median, number of distinct elements, frequency moments, heavy hitters.
- Challenge:

(i) Limited working memory, i.e., sublinear(n,m).

(ii) Sequential access to adversarially ordered data.

(iii) Process each update quickly.

Slide derived from [McGregor 10]

Graph Streams

- $S = \langle x_1, x_2, ..., x_m \rangle; x_i \in [n] \times [n]$
- *A* defines a graph G on n vertices.
- Goal: compute properties of G.
- Challenge: subject to usual streaming constraints.



Snapshot of Internet Graph Source: Wikipedia

Bad News

- Many graph problems are impossible in standard streaming model (require linear space or many passes over data).
- E.g. Ω (n) space needed for connectivity, bipartiteness. Ω (n²) space needed for counting triangles, diameter, perfect matching.



- Often hard even to approximate.
- Graph problems ripe for outsourcing.

Outsourcing Models

 Stream Punctuation [Tucker et al. 05], Proof Infused Streams [Li et al. 07], Stream Outsourcing [Yi et al. 08], Best-Order Model [Das Sarma et al. 09] (is a special case of our model)

Outsourcing Models

- Stream Punctuation [Tucker et al. 05], Proof Infused Streams [Li et al. 07], Stream Outsourcing [Yi et al. 08], Best-Order Model [Das Sarma et al. 09] (is a special case of our model)
- [Chakrabarti et al. 09] Online Annotation Model: Give streaming algorithm access to powerful *helper* H who can annotate the stream.
 - Main motivation: Commercial cloud computing services such as Amazon EC2. Helper is untrusted.
 - Also, Volunteer Computing (SETI@home. Great Internet Mersenne Prime Search, etc.)
 - Weak peripheral devices.

Online Annotation Model

- **<u>Problem</u>**: Given stream *S*, want to compute f(S):
- $S = \langle x_1, x_2, x_3, x_4, x_5, x_6, \dots, x_m \rangle$
- <u>Helper H</u>: augments stream with *h*-word annotation:
 (S,a)=<x₁, x₂, x₃, x₄, x₅, x₆, ..., x_m, a₁, a₂, ..., a_h>
- Verifier V: using v words of space and random string r, run verification algorithm to compute g(S,a,r) such that for all a either:
 a)Pr_r[g(S,a,r) =f(S)]=1 (we say a is valid for S) or
 b) Pr_r[g(S,a,r) = ⊥]≥1-δ (we say a is δ -invalid for S)
 c) And at least one a is valid for S.

Note: this model differs slightly from [Chakrabarti et al. 09].

Online Annotation Model

- Two costs: words of annotation *h* and working memory *v*.
 - We refer to (*h*, *v*)-protocols.
 - Primarily interested in minimizing *v*.
 - But strive for optimal tradeoffs between *h* and *v*.
 - Proves more challenging for graph streams than numerical streams. Algebraic structure seems critical.

Fingerprinting

- Need a way to test multiset equality (e.g. to see if two streams have the same frequency distribution).
 - But need to do so in a streaming fashion.
 - We often use this to make sure H is "consistent".
- Solution: fingerprints.
 - Hash functions that can be computed by a streaming verifier.
 - If $S \neq S'$ as frequency distributions, then $f(S) \neq f(S')$ w.h.p.
- We choose a fingerprint function *f* that is linear. *f*(S °S') = *f*(S) + *f*(S') where ° denotes concatenation. Will need this for matrix-vector multiplication.

Two Approaches To Designing Protocols

- 1. Prove matching upper and lower bounds on a quantity.
 - One bound often easy: just give feasible solution.
 - Proving optimality more difficult. Usually requires problem structure.
- 2. Use H to "verify" execution of a non-streaming algorithm.

• Suppose stream *A* contains (only the non-zero) entries of matrix **A**, vectors **b** and **c**, interleaved in any order (updates are of the form e.g. "add y to entry (i,j) of **A**"). The LP streaming problem on *A* is to determine max $\{\mathbf{c}^T \mathbf{x} \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$.

- Suppose stream *A* contains (only the non-zero) entries of matrix **A**, vectors **b** and **c**, interleaved in any order (updates are of the form e.g. "add y to entry (i,j) of **A**"). The LP streaming problem on *A* is to determine max $\{\mathbf{c}^T \mathbf{x} \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$.
- Theorem: There is a (|**A**|, 1) protocol for the LP streaming problem, where |**A**| is number of non-zero entries in **A**.

- Suppose stream *A* contains (only the non-zero) entries of matrix **A**, vectors **b** and **c**, interleaved in any order (updates are of the form e.g. "add y to entry (i,j) of **A**"). The LP streaming problem on *A* is to determine max $\{\mathbf{c}^T \mathbf{x} \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$.
- Theorem: There is a (|**A**|, 1) protocol for the LP streaming problem, where |**A**| is number of non-zero entries in **A**.
 - Protocol ("naïve" matrix-vector multiplication):
 - 1. H provides primal-feasible solution **x**.
 - 2. For each row i of **A**:

Repeat entries of **x** and row i of **A** in order to prove feasibility. Fingerprints ensure consistency.

3. Repeat for dual-feasible solution **y**. Accept if $value(\mathbf{x}) = value(\mathbf{y})$.

- Suppose stream *A* contains (only the non-zero) entries of matrix **A**, vectors **b** and **c**, interleaved in any order (updates are of the form e.g. "add y to entry (i,j) of **A**"). The LP streaming problem on *A* is to determine max $\{\mathbf{c}^T \mathbf{x} \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$.
- Theorem: There is a (|**A**|, 1) protocol for the LP streaming problem, where |**A**| is number of non-zero entries in **A**.
 - Protocol ("naïve" matrix-vector multiplication):
 - 1. H provides primal-feasible solution **x**.
 - 2. For each row i of **A**:

Repeat entries of \mathbf{x} and row i of \mathbf{A} in order to prove feasibility. Fingerprints ensure consistency.

- 3. Repeat for dual-feasible solution **y**. Accept if $value(\mathbf{x})=value(\mathbf{y})$.
- Details on precision of rationals are skipped.

Application to Graph Streams

• Corollary: Protocol for TUM IPs, since optimality can be proven via a solution to the dual of its LP relaxation.

Application to Graph Streams

- Corollary: Protocol for TUM IPs, since optimality can be proven via a solution to the dual of its LP relaxation.
- Corollary: (m, 1) protocols for max-flow, min-cut, minimum-weight bipartite perfect matching, and shortest *s*-*t* path. Lower bound of $hv = \Omega(n^2)$ for all four.

Application to Graph Streams

- Corollary: Protocol for TUM IPs, since optimality can be proven via a solution to the dual of its LP relaxation.
- Corollary: (m, 1) protocols for max-flow, min-cut, minimum-weight bipartite perfect matching, and shortest *s*-*t* path. Lower bound of $hv = \Omega(n^2)$ for all four.
- A is sparse for the problems above, which suits the naïve protocol. For denser A, can get optimal tradeoffs between *h* and *v*.

- We will get optimal $(n^{1+\alpha}, n^{1-\alpha})$ protocol. Lower bound: $hv = \Omega(n^2)$.
 - Corollary I: Protocols for dense LPs, effective resistances, verifying eigenvalues of Laplacian.

- We will get optimal $(n^{1+\alpha}, n^{1-\alpha})$ protocol. Lower bound: $hv = \Omega(n^2)$.
 - Corollary I: Protocols for dense LPs, effective resistances, verifying eigenvalues of Laplacian.
 - Corollary II: Optimal tradeoffs for Quadratic Programs, Second-Order Cone Programs. (n², 1) protocol for Semidefinite Programs.

- First idea: Treat as n separate inner-product queries, one for each row of A.
 - Worse than "naïve" solution.
 - Multiplies *both h* and *v* by n, as compared to a single innerproduct query.

- First idea: Treat as n separate inner-product queries, one for each row of A.
 - Worse than "naïve" solution.
 - Multiplies *both h* and *v* by n, as compared to a single innerproduct query.
- Key observation: one vector, **x**, in each inner-product query is constant.
 - This plus linear fingerprints lets us just multiply *h* by n.
 - *v* will be the same as for a *single* inner product query.

Approach 2: Simulate an Algorithm

- Main tool: Offline memory checker [Blum et al. '94]. Allows efficient verification of a sequence of accesses to a large memory.
- Lets us convert any deterministic algorithm into a protocol in our model.
- Running time of the algorithm in the RAM model becomes annotation size *h*.

Memory Checker [Blum et al. '94]

- Consider a *memory transcript* of a sequence of reads and writes to memory.
- A transcript is *valid* if each read of address i returns the last value written to that address.
- Memory checker requires transcript be provided in a carefully chosen ("augmented") format.
 - Augmentation blows up transcript size only by constant factor.
- *V* checks validity by keeping a constant number of fingerprints and performing simple local checks on the transcript.

- Any graph algorithm *M* in RAM model requiring time t can be (verifiably) simulated by an (m+t, 1)-protocol.
- *Proof sketch:*
 - Step 1: H first plays a valid adjacency-list representation of *G* to "initialize memory".
 - Step 2: H provides a valid augmented transcript T of the read and write operations performed by algorithm.
 - V checks validity using memory-checker. V also checks all read/ write operations are as prescribed by *M*.

• Corollary: (m, 1)-protocol for MST; (m + n log n, 1)-protocol to verify single-source shortest paths; (n³,1)-protocol for all-pairs shortest paths.

- Corollary: (m, 1)-protocol for MST; (m + n log n, 1)-protocol to verify single-source shortest paths; (n³,1)-protocol for all-pairs shortest paths.
- Proof for MST: Given a spanning tree T, there exists a lineartime algorithm *M* for verifying that T is minimum e.g. [King '97].

- Corollary: (m, 1)-protocol for MST; (m + n log n, 1)-protocol to verify single-source shortest paths; (n³,1)-protocol for all-pairs shortest paths.
- Proof for MST: Given a spanning tree T, there exists a lineartime algorithm *M* for verifying that T is minimum e.g. [King '97].
- Lower bounds: $hv = \Omega(n^2)$ for single source and all-pairs shortest paths. $hv = \Omega(n^2)$ for MST if edge weights specified incrementally.

Pitfall of Memory-Checking

Cannot simulate *randomized* algorithms



• Theorem: $(n^2 \log n, 1)$ protocol. Lower bound: $hv = \Omega(n^2)$.

- Theorem: $(n^2 \log n, 1)$ protocol. Lower bound: $hv = \Omega(n^2)$.
- [Chakrabarti et al. 09]: (n², 1) protocol for matrix-matrix multiplication.

- Theorem: $(n^2 \log n, 1)$ protocol. Lower bound: $hv = \Omega(n^2)$.
- [Chakrabarti et al. 09]: (n², 1) protocol for matrix-matrix multiplication.
- Let A be adjacency matrix of <u>G</u>.

- Theorem: $(n^2 \log n, 1)$ protocol. Lower bound: $hv = \Omega(n^2)$.
- [Chakrabarti et al. 09]: (n², 1) protocol for matrix-matrix multiplication.
- Let A be adjacency matrix of <u>G.</u>
- (I + A)^l_{ij} >0 if and only if there is a path of length at most l from i to j.

- Theorem: $(n^2 \log n, 1)$ protocol. Lower bound: $hv = \Omega(n^2)$.
- [Chakrabarti et al. 09]: (n², 1) protocol for matrix-matrix multiplication.
- Let A be adjacency matrix of <u>G.</u>
- (I + A)^l_{ij} >0 if and only if there is a path of length at most l from i to j.
- Protocol:
- 1. H claims diameter is l
- 2. Use repeated squaring to prove $(I+A)^1$ has an entry that is 0, and $(I+A)^{1+1} \neq 0$ for all i(j).

Summary

- (m, 1)-protocol for max-matching. $hv = \Omega$ (n²) lower bound for dense graphs, so we can't do better.
- (m, 1)-protocols for LPs TUM IPs. $hv = \Omega$ (n²) lower bound for several TUM IPs.
- Optimal (n^{1+α}, n^{1-α})-protocol for dense matrix-vector multiplication. (n^{1+α}, n^{1-α})-protocols for effective resistance, verifying eigenvalues of Laplacian or Adjacency matrix, LPs, QPs, SOCPs.
- General simulation theorem; applications to MST, shortest paths.
- (n²log n, 1) protocol for Diameter. $hv = \Omega$ (n²) lower bound.

Open questions

- Tradeoffs between *h*, *v* for matching, MST, diameter?
- Distributed computation: Protocols that work with Map-Reduce.
- What if we allow multiple rounds of interaction between H and V? Can we get exponentially better protocols?

Thank you!