SIPping from the firehose: Streaming Interactive Proofs for verifying computations

Graham Cormode

graham@research.att.com

Amit Chakrabarti (Dartmouth)

Andrew McGregor (U Mass Amherst)

Michael Mitzenmacher (Harvard)

Justin Thaler (Harvard)

Ke Yi (HKUST)

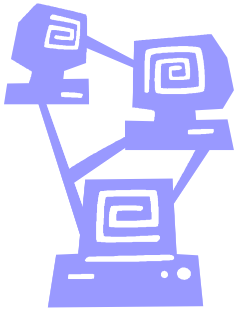
Data Streams

- The data stream model requires computation in small space with a single pass over input data
  - Models large network data, database transactions

- Fundamental challenge of data stream analysis: Too much information to store or transmit
- So process data as it arrives: one pass, small space: the *data stream* approach.
- Approximate answers to many questions are OK, if there are guarantees of result quality
  - Parameters: space needed, time per update as function of approximation accuracy, probability of error

Data Stream Algorithms

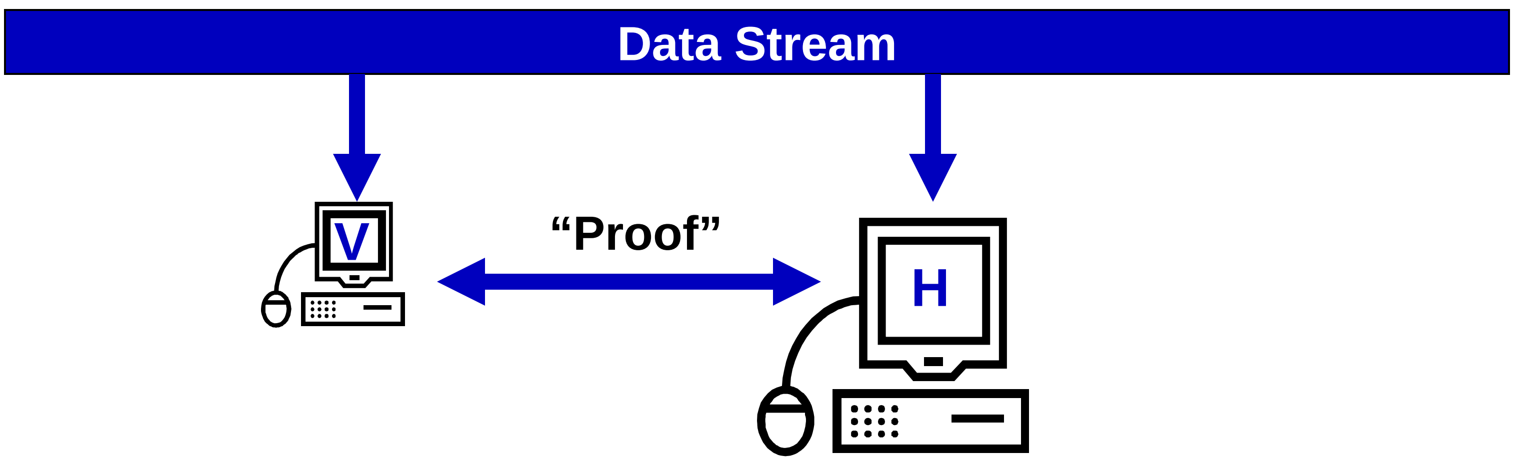
- Many problems solved efficiently in streaming model
  - F_0: How many distinct items (out of 10^{12} possible)?
  - HH: Which items occur most frequently?
  - H: What is the (empirical) entropy of the observed dbn?
- But many other natural problems are "hard" in this model
  - Hardness means large amount of space is needed
  - E.g. Was a particular item in the stream?
  - E.g. What is inner product of two vectors?



- Lower bounds proved via communication complexity
  - Independent of any assumptions on computational power

Streaming Interactive Proofs

- "Practical" solution: outsource to a more powerful "helper"
  - Fundamental problem: how to be sure that the helper is being honest?
- Helper provides "proof" of the correct answer
  - Ensure that "verifier" has very low probability of being fooled
  - Related to communication complexity Arthur-Merlin model, and Algebrization, with additional streaming constraints

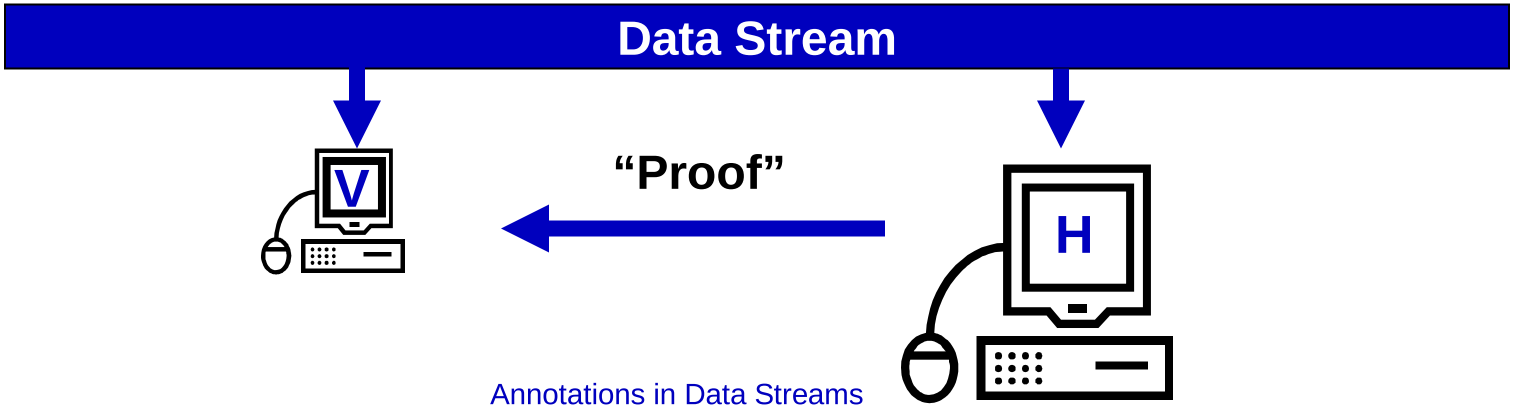


Motivating Applications

- Cloud Computing
  - To save money, and energy, outsource data to a 3rd party
  - But want to know they are honest, without duplicating!
  - Use a streaming interactive proof to verify computation
- Trusted Hardware
  - Hardware components within a (distributed) system (e.g. video card, additional computing cores)
  - Use streaming interactive proofs for (mutual) trust

One Round Model

- One-round model [Chakrabarti, C, McGregor 09]
  - Define protocol with help function h over input length N
  - Maximum length of h over all inputs defines *help cost*, H
  - Verifier has V bits of memory to work in
  - Verifier uses randomness so that:
    - For all help strings, $\Pr[\text{output} \neq f(x)] \leq \delta$
    - Exists a help string so that $\Pr[\text{output} = f(x)] \geq 1-\delta$
  - H = 0, V = N is trivial; but H = N, V = polylog N is not



Index Problem

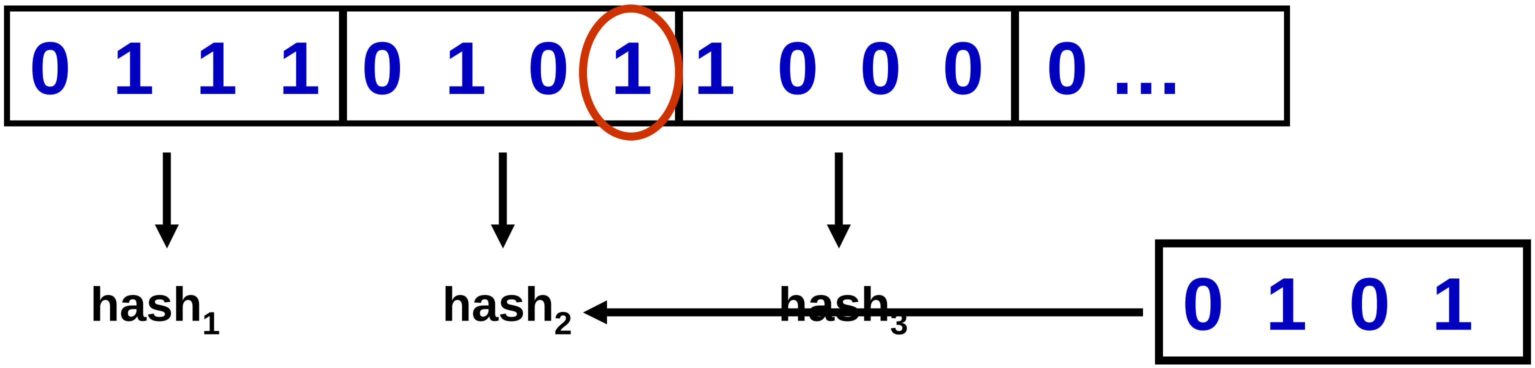
0 1 1 1 0 1 0 1 1 0 0 0 0 …	2010

- Fundamental (hard) problem in data streams
  - Input is a length N binary string x followed by index y
  - Desired output is x[y]
  - Requires $\Omega(N)$ space even probabilistically
- Result: can obtain protocols for HV = O(N log N)
  - E.g. $H = O(\sqrt{N})$, $V = O(\sqrt{N} \log N)$
  - $HV = \Omega(N)$ is necessary

Lower Bound

- Show that a protocol implies solution in traditional model
- Pick k so that $\Pr[\text{ Binomial}(k,1/3) > k/2\ ] < 2^{-H}/3$
- Start protocol independently $k = \Theta(H)$ times in parallel
  - Cost in bits is $k * V = O(HV)$
- Search for a H bit help string so that majority of instances output 0 or 1, and output that value.
- If protocol is correct with $\delta < 1/3$, must exist some help string that does not 'fail' w/prob $2/3$
  - And low probability that it leads to the wrong output value
- By choice of k, 2^H strings each fail with prob $2^{-H}/3$
  - Gives a traditional protocol with cost $O(HV)$, must be $\Omega(N)$

Index Upper Bound



- Divide the bit string into blocks of H bits
- Verifier remembers a hash on each block
- After seeing index, Helper replays its block
- Verifier checks hash agrees, and outputs x[y]

- Cost: H bits of help, V = N/H hashes
  - So HV = O(N log N), any point on tradeoff is possible

Median Finding

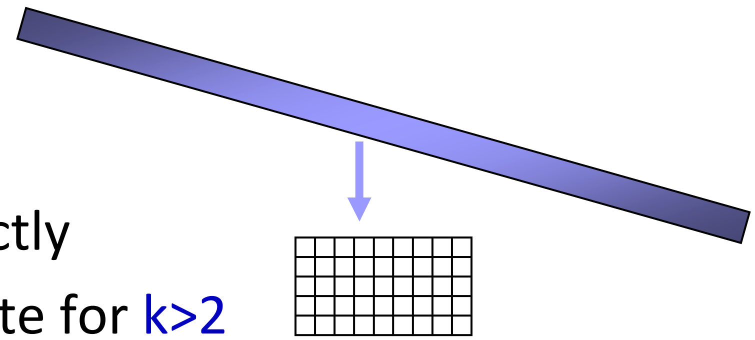
- Similar ideas allow tracking any vector
- Use to find median of m items $\in \{1 \dots N\}$

- Define rank vector s.t. rank[i] = number of items seen $< i$
- Arrival of item j means rank[i] $\leftarrow$ rank[i] + 1 for all $i > j$
- Divide rank[] into blocks of H counters
  – Can update hash of a block without knowing value of rank[i]
- Helper claims median is M, and shows rank[M], rank[M+1]
  – Verifier checks that rank[M] $\leq N/2$, rank[M+1] $\geq N/2$

- Gives solution for any HV s.t. $HV = \Omega(N \log N)$

Frequency Moments

- Given a sequence of m items, let w_i denote frequency of item i
- Define $F_k = \sum_i |w_i|^k$
  - Core computation in data streams
  - Requires $\Omega(N)$ space to compute exactly
  - Need polynomial space to approximate for $k>2$



- Results: for h,v s.t. $(hv) > N$, exists a protocol with $H = k^2\, h \log m$, $V = O(k\, v \log m)$ to compute F_k
  - Lower bounds: $HV = \Omega(N)$ necessary for exact, and $HV = \Omega(N^{1-5/k})$ for approximate F_k computation

Frequency Moments

3	7	1	2	0	8	5	9	1	1	1	0

- Map $[N]$ to $h \times v$ array
- Interpolate entries in array as a polynomial $f(x,y)$
- Verifier picks random r, evaluates $f(r, j)$ for $j \in [v]$
- Helper sends $s(x) = \sum_{j\in [v]} f(x, j)^k$ (degree kh)
  - Verifier checks $s(r) = \sum_{j\in [v]} f(r,j)^k$
  - Output $F_k = \sum_{i \in [h]} s(i)$ if test passed
- Probability of failure small if evaluated over large enough field

3	7	1	2
0	8	5	9
1	1	1	0

Streaming Computation

- Must evaluate $f(r,i)$ incrementally as $f()$ is defined by stream
- Structure of polynomial means updates to (a,b) cause

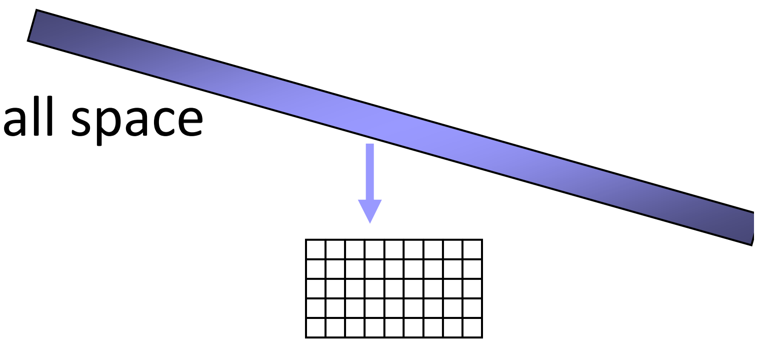
$$f(r,i) \leftarrow f(r,i) + p_{a,b}(r,i)$$

where $p_{a,b}(x,y) = \prod_{i \in [h]\setminus\{a\}} (x-i)(a-i)^{-1} \cdot \prod_{j \in [v]\setminus\{b\}} (y-j)(b-j)^{-1}$

- Can be computed quickly, using appropriate precomputed look-up tables

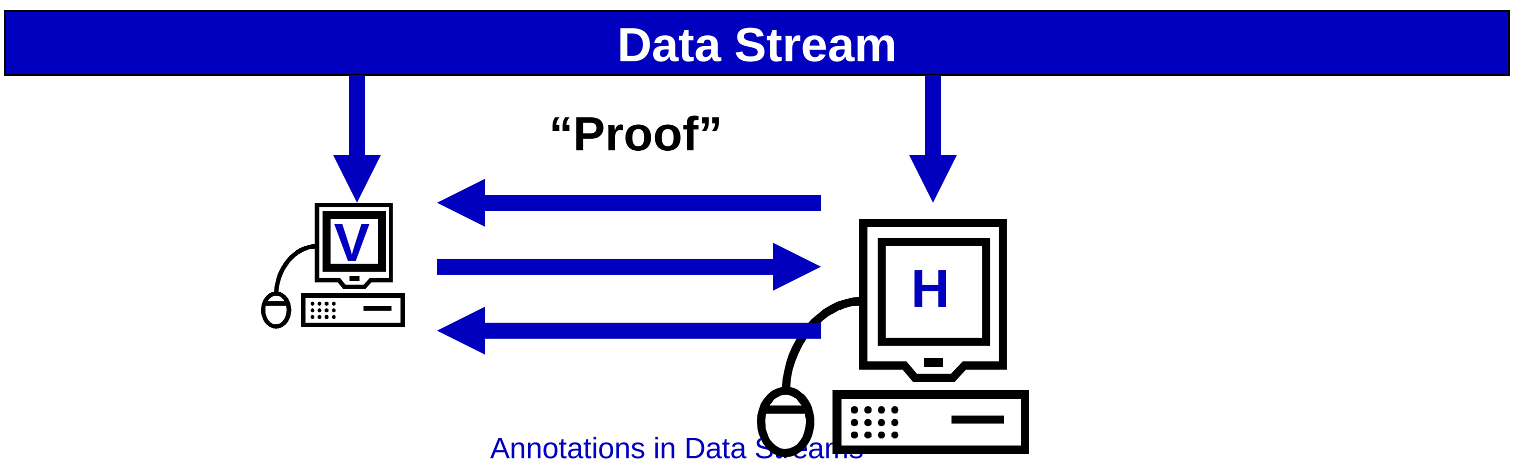
Applications of Frequency Moments

- Inner products: $x \cdot y = ½ (F_2(x+y) – (F_2(x) + F_2(y)))$
  - Adapt previous protocol to verify directly
- Approximate F_2:
  - Methods known to $(1 \pm \varepsilon)$ approximate F_2 by computing F_2 of a random projection
  - Random projection computable in small space
  - Gives $HV = \Theta(1/\varepsilon^2)$ tradeoff
- Approximate $F_\infty = \max_i m_i$:
  - Observe that $F_\infty^t \le F_t \le N F_\infty^t$
  - Pick $t = \log N/\log (1+\varepsilon)$ to get $(1+\varepsilon)$ approx to F_∞
  - Gives $HV = \Theta(1/\varepsilon^3 \text{ poly-log } N)$ tradeoff



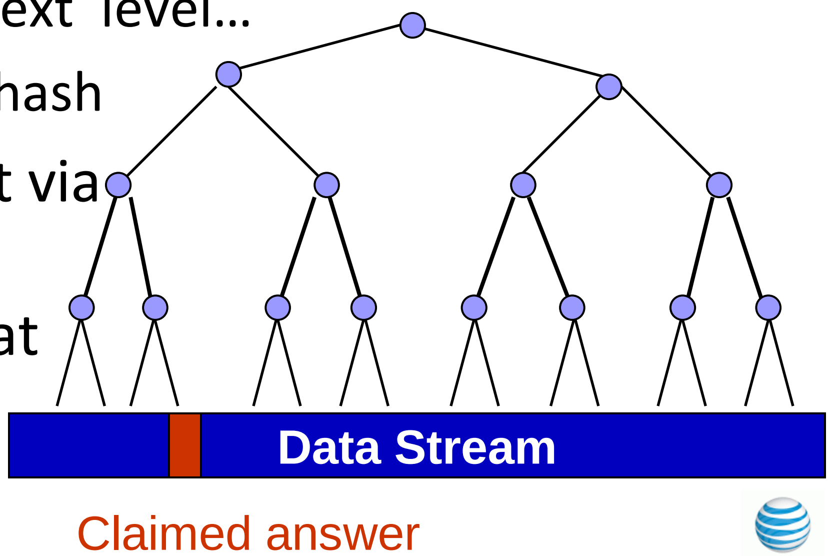
Multi-Round Protocol

- Advantage of one-round protocols: Helper can provide proof without direct interaction (e.g. publish + go offline)
- Disadvantage: Resources still polynomial in input size
- Multi-round protocol can improve exponentially [C, Yi 10]:
  - Helper and Verifier follow communication protocol
  - H now denotes upper bound on total communication
  - V is verifier's space, study tradeoff between H and V as before



Multi-Round Index Protocol

- Basic idea: V keeps hash of whole stream, use helper to help check hash of stream containing claimed answer
  - Verifier imposes a binary tree, and a (secret) hash for each level
  - Round 1: Helper sends answer, and its sibling
    Verifier sends hash for leaf level
  - Round 2: Helper sends hash of answer's parent's sibling
    Verifier sends hash for next level…
  - Round log N: Verifier checks root hash
- Correctness: Helper can only cheat via hash collisions—but doesn't know hash function until too late to cheat
  - Small chance over log N levels



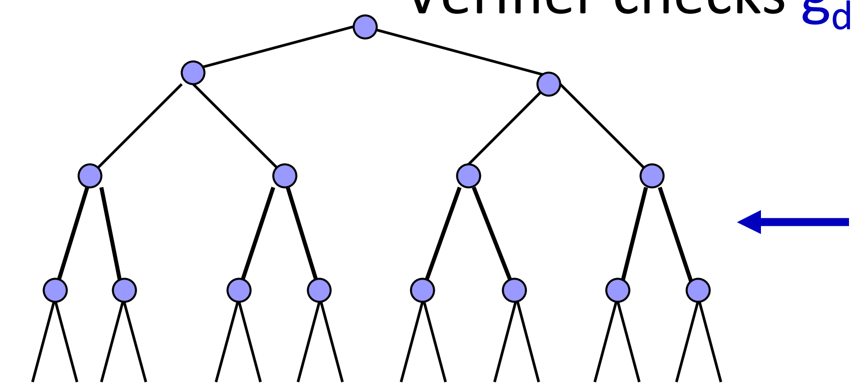
Multi-Round Index Protocol

- Challenge: Verifier must compute hash of root in small space
- h(root) $= h_{\log N}(h_{\log N-1}(\text{left half}), h_{\log N-1}(\text{right half}))$
  $= h_{\log N}(h_{\log N} \dots h_2 (h_1 (x_1, x_2) \dots))$
- Solution: appropriate choice of each hash function
  - $h_i(x, y) = x + r_i\, y \bmod p$ gives sufficient security ($1/p \log N$ error)
  - Then $h(\text{root}) = \sum_i (w_i \prod_{j=1}^{\log N} r_j^{\text{bit}(j,i)})$ where $\text{bit}(j,i)$ = i'th bit of j
  - So each update requires only $\log N$ field multiplications
- Final bounds: $O(\log^2 N)$ communication, $O(\log^2 N)$ space

Multi-Round Frequency Moments

Now index data using $\{0,1\}^d$ in $d = \log N$ dimensional space

- Verifier picks one $(r_1 \ldots r_d) \in [p]^d$, and evaluates $f^k(r_1, r_2, \ldots r_d)$
- Round 1: Helper sends $g_1(x_1)=\sum_{x_2 \ldots x_d} f^k(x_1, x_2 \ldots x_d)$, V sends r_1
- Round i: Helper sends $g_i(x_i) = \sum_{x_{i+1} \ldots x_d} f^k(r_1, r_2 \ldots r_{i-1}, x_i, x_{i+1} \ldots x_d)$
  Verifier checks $g_{i-1}(r_{i-1}) = g_i(0) + g_i(1)$, sends r_i
- Round d: Helper sends $g_d(x_d) = f^k(r_1, \ldots r_{d-1}, x_d)$
  Verifier checks $g_d(r_d) = f^k(r_1, r_2, \ldots r_d)$



3 7 1 2 0 8 5 9 1 1 1 0 …

3 7 1 2 0 8 5 9 1 1 1 0

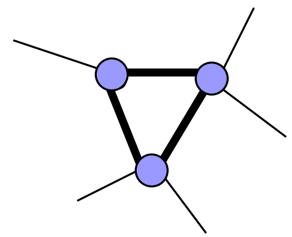
Multi-Round Frequency Moments

- Correctness: helper can't cheat last round without knowing r_d
- Then can't cheat round i without knowing r_i...
  - Similar to protocols from "traditional" Interactive Proofs
- Inductive proof, conditioned on each later round succeeding
- Bounds: $O(k^2 \log N)$ total communication, $O(k \log N)$ space
- V's incremental computation possible in small space, via

$$\prod_{j=1}^{d} (r_j + \mathrm{bit}(j,i)(1-2r_j))$$

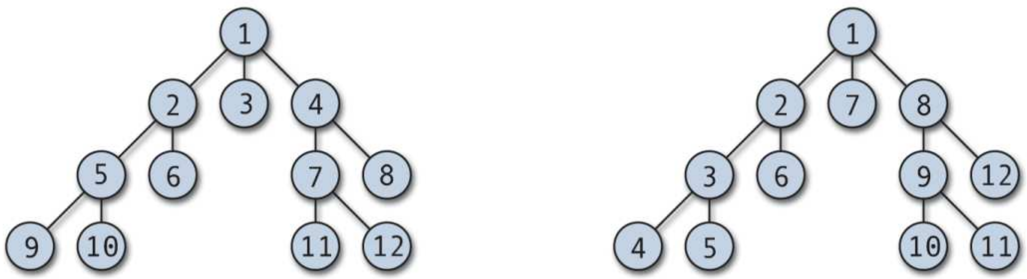
- Intermediate polynomials relatively cheap for helper to find

Graph Problems



- Count the number of triangles in a graph [CCM09]
  - $HV = \Omega(N^2)$ is necessary in one round
  - $H = O(N^2)$, $V = O(\log N)$ via verifying matrix multiplication
  - $HV = O(N^3)$ tradeoff via Frequency Moments in one round
- Connectivity and Bipartite Perfect Matchings with $V = O(\log N)$ space in one round
  - Different witnesses presented for positive/negative answers
  - No tradeoffs known

Graph Problems



- H=|E|, V=log|E| graph protocols [C, Mitzenmacher, Thaler 10]
  - BFS: List edges in BFS order, nodes with depth information
  - DFS: List edges in DFS order, with information about stack
  - MST: List edges in weight order, with component information
  - Maximum matching: prove matching upper and lower bounds
- Connection to unimodular integer programs
  - Can formulate many flow problems as unimodular IPs
  - Use verification on matching feasible solutions for primal/dual

Vector Problems

- Find and verify frequent items with $V = O(\log N)$ space
  - Complexity comes from verifying none are missing
- F_0: Count the number of distinct items
  - $HV = O(N^{2/3})$ by extension of arguments for F_k
  - In parallel use HH protocol to remove very high frequency items
- F_∞: Find the most frequently occurring item
  - "Harder" than finding just items above a frequency threshold
  - $HV = O(N^{2/3})$, solution similar to F_0 approach

at&t Your world. Delivered.

Open Challenges

- Lower bounds for multi-round versions of the protocols
  - May need new communication complexity models
- Characterize problems that can be solved in this model
  - NP is known to be solvable with H = poly(N), V = log N [Lipton 90]
  - But we want H=O(N), and ideally H=o(N)
- Use these protocols
  - Protocols seem practical, but are they compelling?
  - For what problems are protocols most needed?