# Sublinear-Space Approximations of Vertex Centrality in Evolving Graphs

Benjamin W. Priest

Thayer School of Engineering
Dartmouth College

*benjamin.w.priest.th@dartmouth*

December 6, 2018

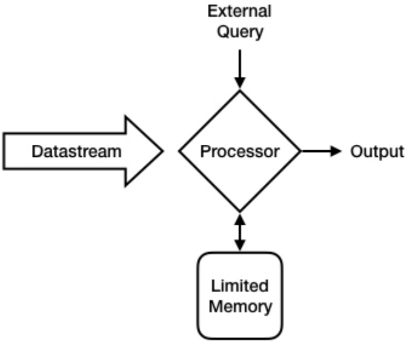
# Overview

# Motivation

- Many modern computing problem focus on complex relational data
- Data are phrased as large graphs
  - e.g. the Internet, communication networks, transportation systems, protein networks, epidemiological models, social networks
- Often want to identify which vertices are “important”
  - Robust to changes?
  - Sublinear memory?

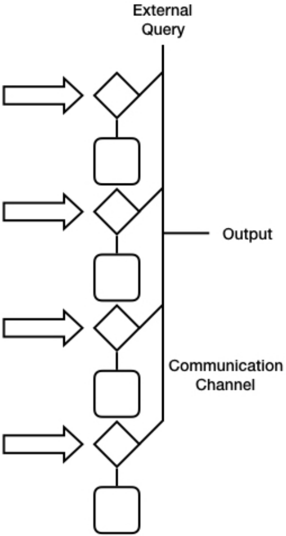
# Overcoming Data Scale: Data Streaming

- Traditional RAM algorithms scale poorly
  - Awkward to store data in memory
  - Superlinear scaling unacceptable
- Data stream model to the rescue!
  - Sequential data access
  - Sublinear memory
  - Linear amortized time
  - Constrained number of passes
  - Monte Carlo Approximations



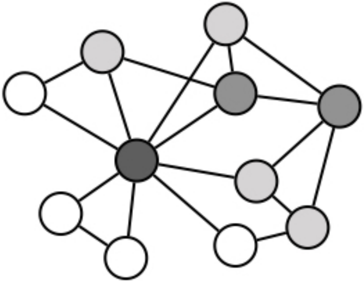
# Overcoming Data Scale: Distributed Data Streaming

- Distributed memory model a staple of HPC
  - Divide computation across many processors
  - Communication an important resource
  - Immense scaling of exact algorithms
- Why not distributed data streams!
  - Sketch data structures afford stream composition
  - Works nicely with vertex-centric algorithms
  - Even greater scaling
  - Linear communication



# Centrality Indices

- Assign scores to vertices
  - Higher score $\rightarrow$ more important
  - Depends on graph structure
  - Different indices in different domains
- Scores are not informative
  - Usually want top $k$ vertices
- Relative order-preserving approximation is acceptable



# Large Scale Graph Centrality

**The Problem**

- Memory overhead
- Compuational Overhead
- Communication Overhead
- Wasted effort
  - Generally only need top elements vis-á-vis a centrality index

**Our Solution**

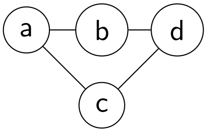
- Sketch data structures
  - Utilize composable streaming summaries of vertex-local information
- Distributed memory
  - Partition graph and distribute sketches
  - Polyloglinear computation, memory, and communication

# Graph Primitives

Assume throughout that $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{w})$, where $|\mathcal{V}| = n$ and $|\mathcal{E}| = m$

- $\mathbf{w}_e$ is the weight of edge $e$ if $e \in \mathcal{E}$ and zero otherwise
- $\mathcal{G}$ has adjacency matrix $A \in \mathbb{R}^{n \times n}$ so that $A_{x,y} = \mathbf{w}_{xy}$ for $xy \in \mathcal{E}$
- $\mathcal{G}$ has vertex-edge incidence matrix $B \in \mathbb{R}^{\binom{n}{2} \times n}$ so that

$$B_{xy,z} = \begin{cases} \mathbf{w}_{xy} & \text{if } x = z \\ -\mathbf{w}_{xy} & \text{if } y = z \\ 0 & \text{else.} \end{cases}$$




$$B = \begin{array}{c} \\ ab \\ ac \\ ad \\ bc \\ bd \\ cd \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{pmatrix} \end{array}$$

# Streaming Background

- A *stream* $\sigma$ accumulating $\mathbf{f} \in \mathbb{R}^n$ is a list of rank-1 updates
  - An update $(i, c)$ means $M \leftarrow \mathbf{f} + c * e_i$
  - A *cash register* stream enforces $c > 0$ for all updates
  - A *turnstile* stream allows negative updates
  - A *strict turnstile* stream allows negatives but enforces $\mathbf{f} \in \mathbb{R}^n_{\geq 0}$
- An algorithm accumulating a data structure $\mathcal{S}$ and is said to be...
  - *streaming* if $\mathcal{S}$ uses $O(\log n)$ memory
  - *semi-streaming* if $\mathcal{S}$ uses $O(n \operatorname{polylog} n)$[1] memory
- Want to minimize the number of passes over $\sigma$
  - 1 pass ideal
  - Constant or logarithmic passes sometimes acceptable

[1] sometimes $O(n^{1+\alpha})$ for $\alpha \in (0, 1/2]$

# Sketching

## Definition (Sketch)

A *Sketch* is a streaming data structure $\mathcal{S}$ that admits a merge operator $\oplus$. If $\circ$ is the stream concatenation operator, then for any streams $\sigma_1$ and $\sigma_2$,

$$\mathcal{S}(\sigma_1) \oplus \mathcal{S}(\sigma_2) = \mathcal{S}(\sigma_1 \circ \sigma_2).$$

## Definition (Linear Sketch)

A *Linear Sketch* $\mathcal{S}$ is a linear projection of $\mho$ to a lower dimension. For any streaming frequency vectors $\mathbf{f}_1$ and $\mathbf{f}_2$ and scalars $a$ and $b$,

$$a\mathcal{S}(\mathbf{f}_1) + b\mathcal{S}(\mathbf{f}_2) = \mathcal{S}(a\mathbf{f}_1 + b\mathbf{f}_2).$$

Sketches are useful for stream summarization when *comparisons between streams* are important

# Summary of Results: Serial Algorithms

Degree Centrality

$$\mathcal{C}^{\mathrm{DEG}}(x) = |\{(u,v) \in E \mid x \in \{u,v\}\}| = \|A_{x,:}\|_1 = \|A_{:,x}\|_1$$

- Naïve online $O(n)$-space and -time algorithm exists

# Summary of Results: Serial Algorithms

Degree Centrality

We show $\widetilde{O}(1)$-space distributable streaming algorithms

- Naïve online $O(n)$-space and -time algorithm exists

# Summary of Results: Serial Algorithms

**Degree Centrality**

We show $\widetilde{O}(1)$-space distributable streaming algorithms

- Naïve online $O(n)$-space and -time algorithm exists

**Closeness Centrality**

$$\mathcal{C}^{\text{CLOSE}}(x) = \frac{1}{\sum_{y \in V} d(x, y)}$$

- Online exact $O(n^2)$-space $O(nm)$-time algorithm [WC14]
- Batch Approximate $O(n^2)$-space and almost-linear time algorithm [CDPW14]

# Summary of Results: Serial Algorithms

### Degree Centrality

We show $\widetilde{O}(1)$-space distributable streaming algorithms

- Naïve online $O(n)$-space and -time algorithm exists

### Closeness Centrality

$$C^{\text{CLOSE}}(x) = \frac{1}{\sum_{y} d(x,y)}$$

We show constant-pass semi-streaming algorithm

- Onl
- Batch Approximate $O(n^2)$-space and almost-linear time algorithm [CDPW14]

# Summary of Results: Distributed Streaming Algorithms

Triangle Count Centrality

$$\mathcal{C}^{\mathrm{TRI}}(x) = |\{yz \in \mathcal{E} \mid xy, yz, xz \in \mathcal{E}\}| \qquad \text{(vertex-local)}$$

$$\mathcal{C}^{\mathrm{TRI}}(xy) = |\{z \in \mathcal{E} \mid xy, yz, xz \in \mathcal{E}\}| \qquad \text{(edge-local)}$$

- Exact $O(m)$-space, $O(m^{\frac{3}{2}})$ serial and distributed algorithms [AKM13]
- Streaming sampling sublinear-space algorithms [LK15, SERU17]
  - Including distributed generalizations [SHL+18, SLO+18, PPS18]

# Summary of Results: Distributed Streaming Algorithms

Triangle Count Centrality

$$\mathcal{C}^{\mathrm{TRI}}(x) = |\{yz \in \mathcal{E} \mid xy, yz, xz \in \mathcal{E}\}| \qquad \text{(vertex-local)}$$

We show 2-pass, semi-streaming, distributed sketch-based query algorithms for estimating heavy hitters

- Ex KM13]
- Streaming sampling sublinear-space algorithms [LK15, SERU17]
  - Including distributed generalizations [SHL$^+$18, SLO$^+$18, PPS18]

# Summary of Results: Distributed Streaming Algorithms

Triangle Count Centrality

$$\mathcal{C}^{\mathrm{TRI}}(x) = |\{yz \in \mathcal{E} \mid xy, yz, xz \in \mathcal{E}\}| \qquad \text{(vertex-local)}$$

We show 2-pass, semi-streaming, distributed sketch-based query algorithms for estimating heavy hitters

- Ex [illegible] KM13]
- Streaming sampling sublinear-space algorithms [LK15, SERU17]
  - Including distributed generalizations [SHL$^+$18, SLO$^+$18, PPS18]

$\kappa$-Path Centrality

$$\mathcal{C}^{\kappa}(x) = \Pr_{p:|p|\leq\kappa}[x \in p \wedge p \text{ a simple path }]$$

- $O(m)$-space $O(n^{1+\alpha}\log^2 n)$-time approximation algorithm [WC14]
- Empirical proxy for betweenness centrality heavy hitters
  - Online exact and approximate $O(n^2)$- and $O(m)$-space algorithms exist [GMB12, WC14, KMB15, BMS14]

# Summary of Results: Distributed Streaming Algorithms

### Triangle Count Centrality

$$\mathcal{C}^{\mathrm{TRI}}(x) = |\{yz \in \mathcal{E} \mid xy, yz, xz \in \mathcal{E}\}| \qquad \text{(vertex-local)}$$

**We show 2-pass, semi-streaming, distributed sketch-based query algorithms for estimating heavy hitters**

- Ex... KM13]
- Streaming sampling sublinear-space algorithms [LK15, SERU17]
  - Including distributed generalizations [SHL+18, SLO+18, PPS18]

### $\kappa$-Path Centrality

$$\mathcal{C}^{\kappa}(x) = \Pr [x \in p \wedge p \text{ a simple path }]$$

**We show distributed sublinear vertex-centric sampling algorithm**

- $O(r$ ... C14]
- Emp... heavy hitters
  - Online exact and approximate $O(n^2)$- and $O(m)$-space algorithms exist [GMB12, WC14, KMB15, BMS14]

# Motivation: Vertex-Centric Algorithms

**The Problem**:

- Most distributed graph algorithms are vertex-centric
  - Partition local vertex information across processors
  - Processors communicate as in rounds [MAB+10]
- Scale-free graphs common in applications
  - Exhibit very high degree vertices
  - Cause computation, communication, and memory “hotspots”
  - Synchronous communication moves at the speed of the slowest processor

**Existing Solutions**:

- Asynchronous Communication
  - Processors communicate point-to-point as needed
  - Increased implementation complexity
- Vertex delegation [PGA14]
  - Subpartition high degree vertices between processors
  - Adds communication overhead

# Approach: Pseudo-Asynchronous Communication Protocol

## The Idea

Allow processors to drop out of communication exchanges when finished

- Partion processor set $\mathcal{P}$ into *local* and *remote* exchanges
  - Takes advantage of hybrid distributed memory
- $P \in \mathcal{P}$ maintains send buffer $\mathcal{S}[P]$ and receive buffer $\mathcal{R}[P]$
  - Begin forwarding messages when $\mathcal{S}[P]$ reaches threshold
  - Drop out of exchange when all other processors in exchanges stop updating $\mathcal{R}[P]$
- Three protocols:
  - Node Local
  - Node Remote
  - Node Local Node Remote (NLNR)

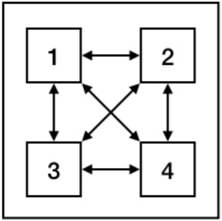
# Node Local and Node Remote

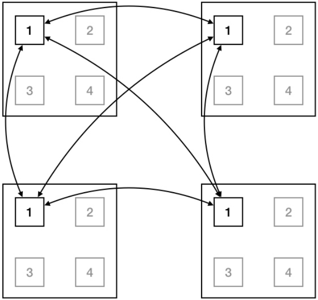
## Node Local

1. Send messages to local core matching destination local offset
2. Send messages to remote core matching destination node offset
3. Good for mostly point-to-point messages

## Node Remote

1. Send messages to remote core matching destination node offset
2. Send messages to local core matching destination local offset
3. Good for large numbers of broadcasts

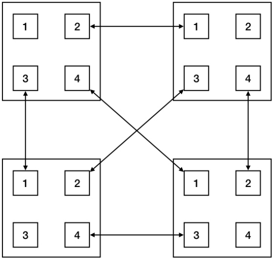


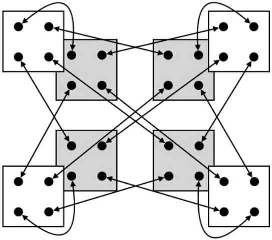



**Node Local Node Remote**

- Further partition processors by *layers*
  - A layer is a collection of nodes equal to the number of cores per node

1. Send messages to local core matching destination layer offset
2. Send messages to remote core matching destination node offset
3. Send messages to local core matching destination local offset
4. Best for extreme scale where many layers exist

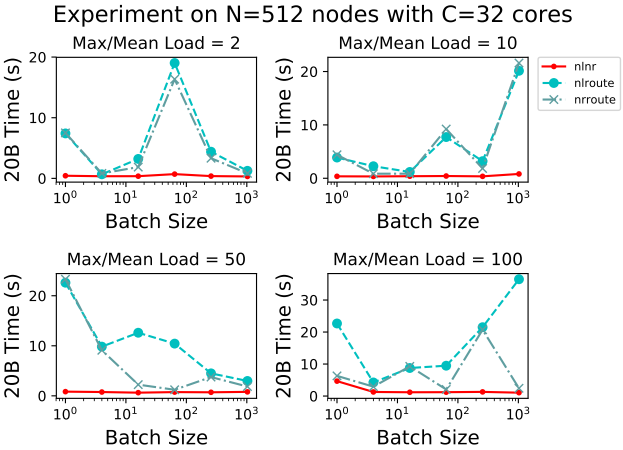


# Validation of Claims

**Experiment**

- 20B message exchange
  - Destination sampled from Pareto distribution
  - Batch size is maximum $|\mathcal{S}[P]|$




NLNR exhibits best scaling, others more useful with fewer nodes

# Implementation Details

YGM **C++/MPI Library**

- Authored by myself, Trevor Steil (UMN), and Roger Pearce (LLNL)
- Simple API for handling pseudo-asynchronous communication
  - Clients need only specify receive behavior
- Supports message serialization for arbitrary, variable-length messages
- Supports LLNL Projects
  - HAVOQgt
  - graph500 scale leader
  - others?
- Useful for not just vertex-centric algorithms, but any algorithm with asymmetric computational and communication load

YGM to be open sourced

**The Problem**:

- Local triangle counting a common big data analytic
  - Exact computation expensive $O\left(m^{\frac{3}{2}}\right)$!
- Recall

$$\mathcal{C}^{\mathrm{TRI}}(x) = |\{yz \in \mathcal{E} \mid xy, yz, xz \in \mathcal{E}\}| \qquad \text{(vertex-local)}$$
$$\mathcal{C}^{\mathrm{TRI}}(xy) = |\{z \in \mathcal{E} \mid xy, yz, xz \in \mathcal{E}\}| \qquad \text{(edge-local)}$$

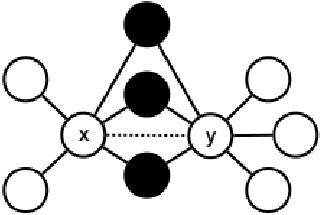
**Existing Solutions**:

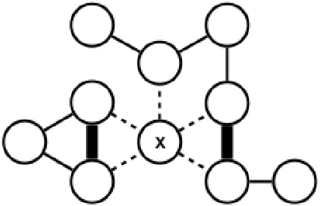
- Many exact distributed algorithms [AKM13, Pea17]
- Many approximate streaming algorithms via sampling [LK15, SERU17]
- ... and some utilizing both models [SHL$^+$18, SLO$^+$18]

# Approach: Sublinear Intersection Method

**Idea**: Intersection method, but using cardinality sketches

- Cardinality sketches summarize set size
- Support union operation, and some support limited intersection operation
  - High variance if intersection is small
  - Likely best performance on heavy hitters
- Affords edge- and vertex-local triangle count estimation
- Outputs only reliable if *triangle density* is nontrivial
  - Triangle density $= \frac{\#\text{ triangles}}{\#\text{ possible triangles}}$



# HYPERLOGLOG Cardinality Sketches

**HLL cardinality sketches**

Maintain $r = 2^p$ 6-bit registers $M$ and a 64-bit hash function $h$

- Insert $x$: let $i = \langle x_1, \ldots, x_p \rangle$ and $w = \langle x_{p+1}, \ldots, x_{64} \rangle$
- $\rho(w) =$ initial zero bits of $w$ plus 1
- $M_i = \max\{M_i, \rho(w)\}$
- Estimator derives from harmonic mean of $M$

# HyperLogLog Cardinality Sketches

**HLL cardinality sketches**

Maintain $r = 2^p$ 6-bit registers $M$ and a 64-bit hash function $h$

- Insert $x$:
- $\rho(w) =$ i
- $M_i = \max\{M_i, \rho(w)\}$
- Estimator derives from harmonic mean of $M$

Outputs $\widetilde{C}$ such that for cardinality C, w.h.p. $|C - \widetilde{C}| \leq \frac{1.04}{\sqrt{m}} C$ [FFGM07]

# HyperLogLog Cardinality Sketches

**HLL cardinality sketches**

Maintain $r = 2^p$ 6-bit registers $M$ and a 64-bit hash function $h$

- Insert $x$:
- $\rho(w) =$ i
- $M_i = \max\{M_i, \rho(w)\}$
- Estimator derives from harmonic mean of $M$

Outputs $\widetilde{C}$ such that for cardinality C, w.h.p. $|C - \widetilde{C}| \leq \frac{1.04}{\sqrt{m}} C$ [FFGM07]

**Useful results**

- Native intersection operator (elementwise maximum)
- Various improved harmonic [HNH13, QKT16] and maximum likelihood estimators [XZC17, Lan17, Ert17]
- Sparsification for low cardinality sets [HNH13]
- Compression to 4 and 3 bit registers [XZC17]
- Intersection estimators [Tin16, CKY17, Ert17]

# $\textsc{DegreeSketch}$ and Triangle Counting

Assume a partition $f : \mathcal{V} \to \mathcal{P}$, and let $\mathcal{V}_P = \{v \in \mathcal{V} \mid f(v) = P\}$

- Distribute $\textsc{DegreeSketch}$ $\mathcal{D}$ across $\mathcal{P}$
  - $\mathcal{D}[v]$ holds a $\mathrm{HLL}$ for adjacency set of $v \in \mathcal{V}$
  - $P$ holds $\mathcal{D}[v]$ for $v \in \mathcal{V}_P$
- Accumulate $\mathcal{D}$ in one pass over $\sigma$
  - Assume $P \in \mathcal{P}$ gets substream $\sigma_P$
  - $P$ sends $xy \in \sigma_P$ to $f(x)$ and $f(y)$
  - When $P$ gets $xy : x \in \mathcal{V}_P$, insert $y$ into $\mathcal{D}[x]$
    - $\mathcal{D}[x]$ starts sparse and eventually saturates
- $\mathcal{D}$ can be queried after estimation, e.g.
  - Estimate $\widetilde{\mathcal{C}}^{\mathrm{DEG}}(v) = \textsc{Estimate}(\mathcal{D}[v])$
  - Estimate $\widetilde{\mathcal{C}}^{\mathrm{TRI}}(uv) = \mathcal{D}[u]\widetilde{\cap}\mathcal{D}[v]$
    - Involves communication if $f(u) \neq f(v)$
  - Estimate $\widetilde{\mathcal{C}}^{\mathrm{TRI}}(v) = \frac{\sum_{uv \in \mathcal{E}} \widetilde{\mathcal{C}}^{\mathrm{TRI}}(uv)}{2}$
    - Requires second pass in general

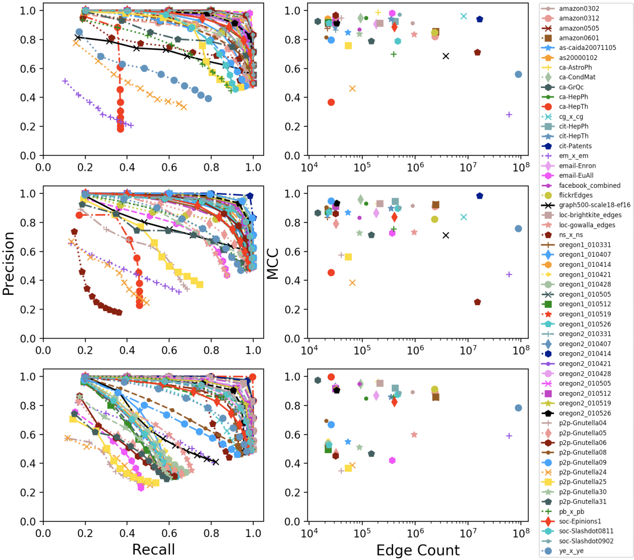
$\widetilde{O}(m)$ time and communication and $\widetilde{O}(\varepsilon^{-2}n)$ space!

# Edge-Local Triangle Count Heavy Hitters

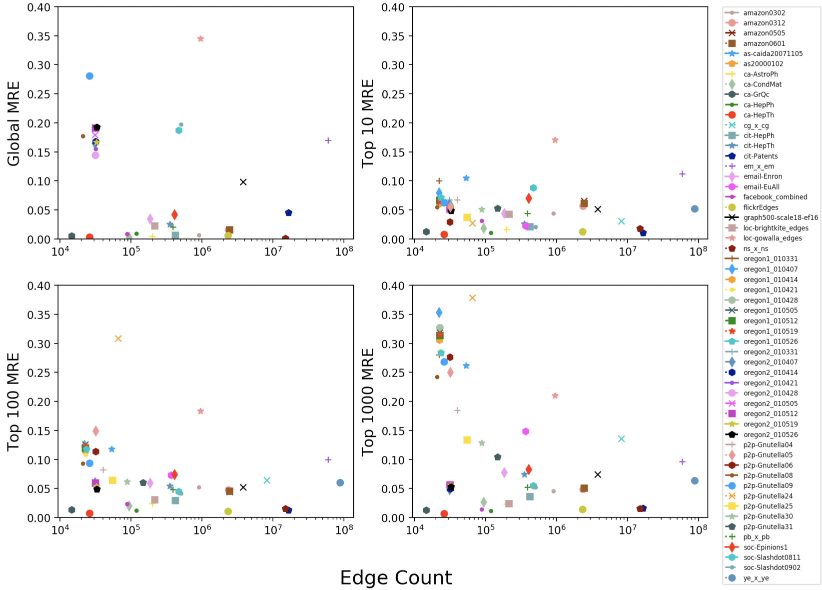
**Algorithm 1** Edge-Local Triangle Count Heavy Hitters

Accumulate $\mathcal{D}$ in distributed pass over $\sigma$
$H_k \leftarrow$ empty $k$-heap
$T \leftarrow 0$
**parallel for** $xy \in \sigma_P$ **do** // second pass
    Send $(\mathsf{E}, xy)$ to $f(x)$ and $(\mathsf{E}, yx)$ $f(y)$
    **for** $(\mathsf{E}, xy) \in \mathcal{R}[P]$ **do**
        Send $(\mathsf{S}, xy, \mathcal{D}[x])$ to $f(y)$
    **for** $(\mathsf{S}, xy, \mathcal{D}[x]) \in \mathcal{R}[P]$ **do**
        Insert $(xy, \mathcal{D}[x] \widetilde{\cap} \mathcal{D}[y])$ into $H_k$
        $T \leftarrow T + \mathcal{D}[x] \widetilde{\cap} \mathcal{D}[y]$
$T \leftarrow T/2$
Global accounting of $T$, $H_k$
**return** $H_k$

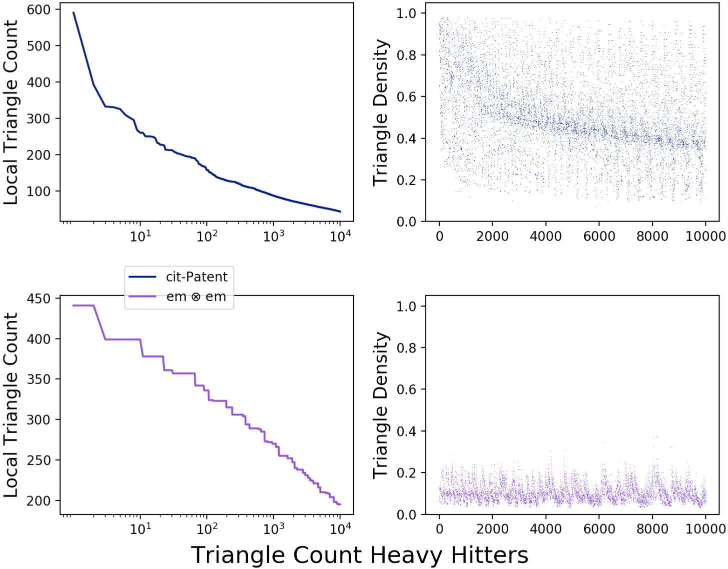
# Validation of Claims: Precision and Recall



# Validation of Claims: Relative Error



# Validation of Claims: Good vs Bad Triangle Density



# Implementation Details

DegreeSketch **C++/MPI Library**

- Authored by myself
- Utilizes YGM for communication
- Accumulation and query API for DegreeSketch
- Supports sparse and compressed registers
- Implementations for edge- and vertex-local triangle count heavy hitter estimation
- Supports more exotic queries

DegreeSketch to be open sourced

# Motivation: Betweenness Centrality Heavy Hitters

**The Problem**:

- Computing Betweenness centrality exactly amounts to computing AllSourcesAllShortestPaths
  - Expensive $O(mn)$!

**Existing Solutions**:

- Approximate via a logarithmic number of SingleSourceAllShortestPaths [GMB12, BMS14, Yos14, KMB15, RK16]
  - Difficult to distribute
  - Unclear if possible in $o(m)$ memory

# Approach: Sublinearize $\kappa$-Path Centrality

**Idea**: "Come at the problem sideways"

- High $\kappa$-path centrality empirically correlates with high betweenness centrality [KAS$^+$13]
- Algorithm amounts to sampling random simple paths
  - Use $\ell_p$ sampling sketches to sublinearize
- Sublinear approximation of $\kappa$-path centrality $\rightarrow$ emprical recovery of high betweenness centrality vertices?

$\kappa$-path centrality

$$\mathrm{PC}(x, \kappa) = \mathrm{Pr}_{p:|p|\leq\kappa}[x \in p \wedge p \text{ a simple path}]$$

"simple path" $=$ non-self-intersecting path

# $\ell_p$ Sampling Sketches

**$\ell_p$ sampling sketches**

Sample from frequency vector $\mathbf{f}$ with probability relative to $\ell_p$ norm

- Sample $t_i \sim_R (0, 1)$ $\forall i \in [n]$
- Rescale updates to $\mathbf{f}_i$ by $1/t_i^{1/p}$
- Accumulate $\textsc{Tug-of-War}$, $\textsc{CountSketch}$, and $\ell_p$ norm sketches
- Use sketches to output $\textsc{CountSketch}$ argmax or FAIL

# $\ell_p$ Sampling Sketches

**$\ell_p$ sampling sketches**

Sample from frequency vector $\mathbf{f}$ with probability relative to $\ell_p$ norm

- Sample $t_i \sim_R (0,$
- Rescale updates
- Accumulate TUG-OF-WAR, COUNTSKETCH, and $\ell_p$ norm sketches
- Use sketches to output COUNTSKETCH argmax or FAIL

Blah blah etc etc

It is complex

# $\ell_p$ Sampling Sketches

**$\ell_p$ sampling sketches**

Sample from frequency vector **f** with probability relative to $\ell_p$ norm

- Sample $t_i$
- Rescale u
- Accumulate TUG-OF-WAR, COUNTSKETCH, and $\ell_p$ norm sketches
- Use sketches to output COUNTSKETCH argmax or FAIL

Outputs $(i, P)$ w.p. $1-\delta$, where $i \in [n]$ is sampled w.p. $P = (1 \pm \varepsilon)\frac{|v_i|^p}{\|v\|_p^p}$ [MW10]

**Useful results** [JST11, Vu18]

- $\ell_0$ sketch requires $\widetilde{O}(\log(1/\delta))$ memory and update time
  - Useful for unweighted random hops
- $\ell_1$ sketch requires $\widetilde{O}(\varepsilon^{-1}\log(1/\delta))$ memory and $\widetilde{O}(\log(1/\delta))$ update time
  - Useful for weighted random hops
- $s$ parallel $\ell_p$ sketches can be accumulated in time independent of $s$

# $\ell_p$ Sampling Graph Sparsification

- Exploit sketch linearity
  - Sample $\ell_0$ sampling sketch matrices $S_1, \ldots, S_t$, each of which will sketch every column of $B$
  - $S_1(B_{:,x})$ returns a sampled neighbor of $x$, say $y$
  - $S_2(B_{:,x}) + S_2(B_{:,y}) = S_2(B_{:,x} + B_{:,y})$ returns a sampled neighbor of the supervertex $(x + y)$
  - et cetera
- This method can solve several problems [AGM12a, AGM12b]:
  - $O(n \operatorname{polylog} n)$ to decide connectivity, k-connectivity, bipartiteness, and to approximate the weight of the MST
  - Multipass $\tilde{O}(n^{1+1/\alpha})$ to compute sparsifiers, the exact MST, $\alpha$-**spanners**, and approximate the maximum weight matching

We will use similar methods to sample random walks and random simple paths in distributed algorithms

# Distributed Accumulation $\ell_p$ Sampling Sketches

- $P \in \mathcal{P}$ accumulates adjacency set $\mathcal{A}[v]$ for each $v \in \mathcal{V}_P$
  - When $\mathcal{A}[v]$ too large, replace it with $s$ $\ell_0$ sampling sketches
  - Write current state and all subsequent updates to disk memory
- Queries to $\mathcal{A}[v]$ return a sampled neighbor of $v$
  - If $\mathcal{A}[v]$ is a set of sketches, one is consumed
    - If $\textsc{Fail}$, repeat
  - Once $\mathcal{A}[v]$ sketches are exhausted, $P$ takes another pass over $v$'s substream in disk memory

Avoids need to subpartition vertices across multiple processors, effectively exchanging communication time for I/O time

# Sublinear Random Walk and Simple Path Sampling

**Random Walk Simulation**

- Sample $t$ vertices $\{v_{1,1}, \ldots, v_{t,1}\}$ and:
  - Sample $v_{i,j+1}$ from $\mathcal{A}[v_{i,j}]$
  - Communicate $(v_{i,1}, \ldots, v_{i,j+1})$ to $f(v_{i,j+1})$

**Random Simple Path Simulation**

- Similar to random walks, except:
  - Do not accumulate sketches ahead of time
  - Sample $v_{i,j+1}$ from $\mathcal{A}[v_{i,j}] \setminus \{v_{i,1}, \ldots, v_{i,j-1}\}$
    - If $\mathcal{A}[v_{i,j}]$ is not in memory, accumulate a sketch ignoring edges to any of $\{v_{i,1}, \ldots, v_{i,j-1}\}$
  - Communicate $(v_{i,1}, \ldots, v_{i,j+1})$ to $f(v_{i,j+1})$

Sublinear distributed storage of graph by sketching high degree vertices

# Sublinear $\kappa$-Path Centrality

$\kappa$-Path Centrality Approximation Algorithm ([KAS+13]):

1. Simulate $T = 2\kappa^2 n^{1-2\alpha} \ln n$ $(\leq \kappa)$-length simple paths over $\mathcal{G}$
   - maintain $count[x]$ for each $x \in \mathcal{V}$
2. $\widetilde{\mathcal{C}}^{\kappa}(x) \leftarrow \frac{count[x]}{2\kappa n^{-2\alpha} \ln n}$

- Given $\alpha \in [-1/2, 1/2]$, for each $x \in \mathcal{V}$, $\left|\widetilde{\mathcal{C}}^{\kappa}(x) - \mathcal{C}^{\kappa}(x)\right| \leq n^{1/2+\alpha}$ w.h.p.
- Easy to distribute in vertex-centric model

# Sublinear $\kappa$-Path Centrality

**Algorithm 2** Sublinear $\kappa$-Path Centrality

**for** $i \in \{1, \ldots, T\}$ **do**
    $p_i \leftarrow$ empty path
    $p_{i,1} \leftarrow$ uniform sample from $\mathcal{V}$
    $l_i \leftarrow$ uniform sample from $\{1, 2, \ldots, \kappa\}$
**for** $x \in \mathcal{V}$ **do**$c_x \leftarrow 0$
**parallel for** $j \in \{1, 2, \ldots, \kappa - 1\}$ **do**
    **parallel for** $i \in \{1, 2, \ldots, T\}$ **do**
        **if** $j < l_i$ **then**
            $p_{i,j+1} \leftarrow$ sample from $\mathcal{A}[p_{i,j}]$
            **if** $p_{i,j+1} = \emptyset$ **then** discard
        **else if** $j = l_i$ **then**
            $c_{p_{i,k}} \leftarrow c_{p_{i,k}} + 1$ for $k \in \{1, \ldots, j\}$
**return** $c_x / 2\kappa n^{-2\alpha} \ln n$ for $x \in V$

# Summary of Results

- The Goal: distributed sublinear approximations of centrality indices
- Engineering Results
  - YGM: Pseudo-Asynchronous Communication Handler
- Algorithmic Results
  - A streaming degree centrality approximation and heavy hitter recovery algorithms
  - A O(1)-pass semi-streaming closeness centrality approximation algorithm
  - 2-pass distributed semi-streaming edge- and vertex-local triangle count heavy hitter estimation algorithms using DegreeSketch
  - Distributed sublinear semi-streaming random walk and random simple path sampling algorithms
  - Distributed sublinear semi-streaming $\kappa$-path centrality estimation algorithm
- Future Work
  - Applications for DegreeSketch
  - Sublinear random walk and random simple path implementation

Questions?