Sublinear-Space Approximations of Centrality in Evolving Massive Graphs

Benjamin W. Priest

Thayer School of Engineering
Dartmouth College

benjamin.w.priest.th@dartmouth

December 12, 2018

Overview

1. Introduction & Background
2. Summary of Results
3. Pseudo-Asynchronous Communication for Distributed Algorithms
4. DegreeSketch and Local Triangle Count Heavy Hitters
5. Sublinear κ-Path Centrality

Overview

1. Introduction & Background
   1. Problem Overview
   2. Graph Primitives
   3. Streaming Data Model Background
   4. Sketching Definitions
2. Summary of Results
3. Pseudo-Asynchronous Communication for Distributed Algorithms
4. DegreeSketch and Local Triangle Count Heavy Hitters
5. Sublinear κ-Path Centrality

Motivation

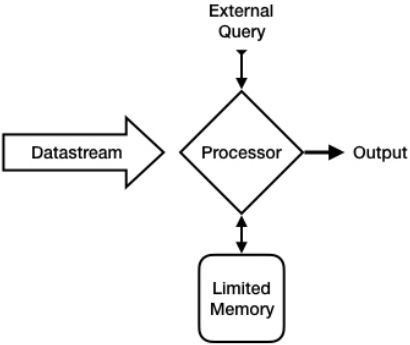
- Many modern computing problem focus on complex relational data
- Data are phrased as large graphs
  - e.g. the Internet, communication networks, transportation systems, protein networks, epidemiological models, social networks
- Often want to identify which vertices are "important"

Approach

Use data stream and distributed memory models

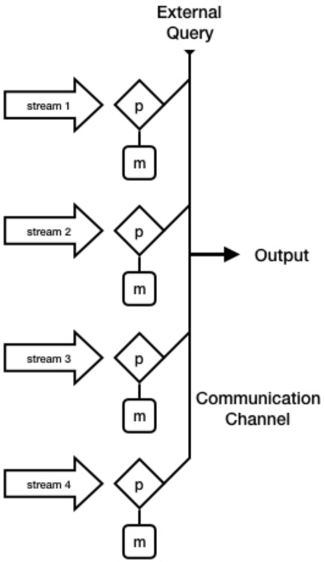
Overcoming Data Scale: Data Streaming

- 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
  - Nearly linear amortized time
  - Constrained number of passes
  - Monte Carlo Approximations



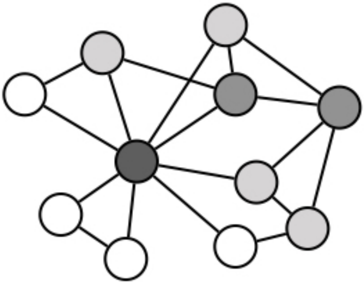
Overcoming Data Scale: Distributed Data Streaming

- Distributed memory model a staple of HPC
  - Divide computation
  - Immense scaling
- Why not distributed data streams!?!
  - **Sketches** - composable summaries
  - 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



Massive-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

Streaming Background

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 $xy \in \mathcal{E}$

$\mathcal{G}$ is given by a *stream* σ

- A list of edge insertions
- If deletions exist, say *turnstile stream*

An algorithm accumulating a data structure $\mathcal{S}$ and is said to be...

- *streaming* if $\mathcal{S}$ uses $\widetilde{O}(1) = O(\log n)$ [1] memory
- *semi-streaming* if $\mathcal{S}$ uses $\widetilde{O}(n) = O(n \operatorname{polylog} n)$ [2] memory

Want to minimize the number of passes over σ

- 1 pass ideal
- Constant or logarithmic passes sometimes acceptable

[1] $\widetilde{O}()$ notation suppresses polylogarithmic factors

[2] sometimes $\widetilde{O}\left(n^{1+\alpha}\right)$ 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 σ_1 and σ_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 $\mathbf{f}$ 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

Overview

1. Introduction & Background
2. Summary of Results
   1. Streaming Degree Centrality
   2. $O(1)$-Pass Semi-Streaming Closeness Centrality
   3. 2-Pass Semi-Streaming Triangle Count Heavy Hitters
   4. Distributed Sublinear κ-Path Centrality
3. Pseudo-Asynchronous Communication for Distributed Algorithms
4. DegreeSketch and Local Triangle Count Heavy Hitters
5. Sublinear κ-Path Centrality

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

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

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, $O\left(m^{\frac{3}{2}}\right)$-time serial and distributed algorithms [AKM13, Pea17] [a]
- Streaming sampling sublinear-space algorithms [LK15, SERU17]
    - Including distributed generalizations [SHL+18, SLO+18]

[a] Roger Pearce, "Triangle counting for scale-free graphs at scale in distributed memory," HPEC 2017

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, $O\left(m^{\frac{3}{2}}\right)$-time serial and distributed algorithms [AKM13, Pea17] [a]
- Streaming sampling sublinear-space algorithms [LK15, SERU17]
  - Including distributed generalizations [SHL+18, SLO+18]

[a] Roger Pearce, "Triangle counting for scale-free graphs at scale in distributed memory," HPEC 2017

We show 2-pass, semi-streaming, distributed sketch-based query algorithms for estimating heavy hitters [PPS18, PPS19] [a] [b]

[a] Benjamin W. Priest, Roger Pearce and Geoffrey Sanders, "Estimating Edge-Local Triangle Count Heavy Hitters in Edge-Linear Time and Almost-Vertex-Linear Space," HPEC 2018

[b] Benjamin W. Priest, Roger Pearce and Geoffrey Sanders, "DegreeSketch: Distributed Cardinality Sketches on Graphs, with Applications to Counting Triangles," In preparation for SigKDD 2019

Summary of Results: Distributed Streaming Algorithms

κ-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

κ-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]

- We show distributed sublinear vertex-centric random walk and simple path sampling algorithms
- Yields sublinear distributed κ-path centrality approximation algorithm

Overview

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
  - High degree vertices cause computation "hotspots"
  - Moves at the speed of the slowest processor

Solutions:

- Asynchronous Communication
  - Processors communicate point-to-point as needed
  - Increased implementation complexity
- Vertex delegation [PGA14] [3]
  - Cut hubs between processors

[3] Roger Pearce, Maya Gokhale and Nancy M. Amato, "Faster parallel traversal of scale free graphs at extreme scale with vertex delegates," SC 2014

Approach: Pseudo-Asynchronous Communication Protocol

The Idea

Aggregate and route messages "asynchronously", allowing 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
- Mailbox abstraction
  - Aggregate messages at intermediaries
  - Route destination node traffic traffic through same remote channel
- Three protocols:
  - Node Local
  - Node Remote
  - Node Local Node Remote (NLNR)

Node Local and Node Remote

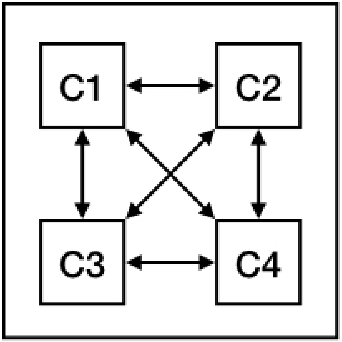
Node Local

1. Exchange locally
2. Forward remotely

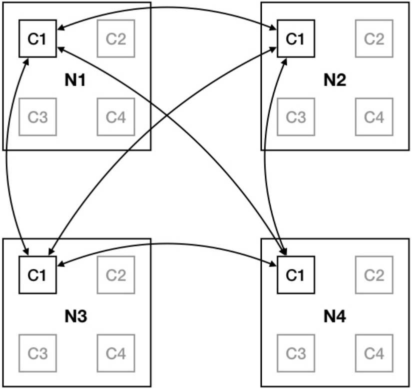
Node Remote ✓

1. Exchange remotely
2. Forward locally

Local Route	**Remote Route**



Node



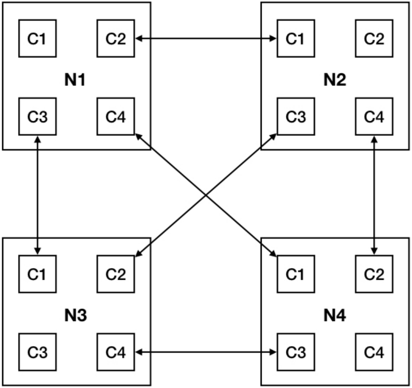
Node Local Node Remote

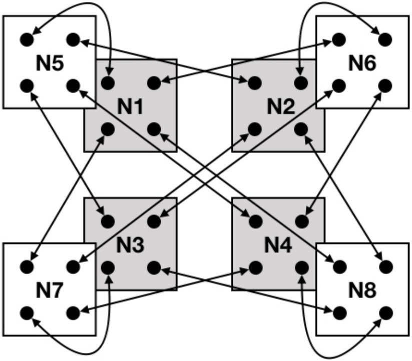
Node Local Node Remote

- Further partition $\mathcal{P}$ by *layers*
    - Set of (# cores) nodes

1. Exchange locally
2. Forward remotely
3. Forward locally

Intra-Layer Remote Route	Inter-Layer Remote Route

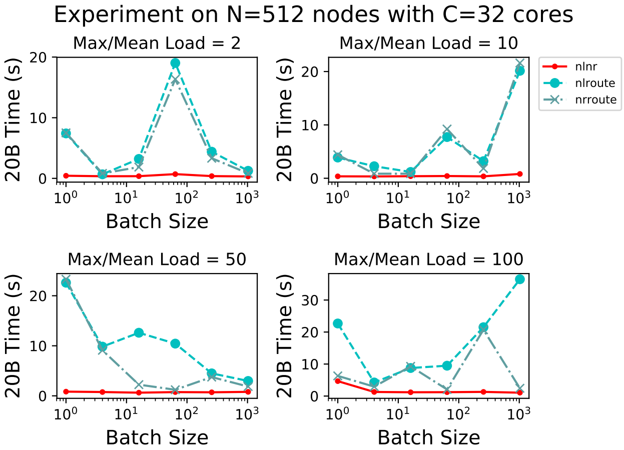




Validation of Claims

Experiment

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



NLNR exhibits best scaling, NR best 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 LLNL Projects
  - HavoqGT (graph challenge & pattern matching)
  - Sierra 42 - largest scale to date graph 500 ($\sim$ 70T edges)
  - Possibly more in the future
    - Eccentricity
- Improvements over legacy HavoqGT routing
  - Flow control via pseudo-asynchronicity
  - NLNR more scalable
  - Variable length messages

YGM to be open sourced, published in IPDPS 2019 workshop

Overview

1. Introduction & Background
2. Summary of Results
3. Pseudo-Asynchronous Communication for Distributed Algorithms
4. DegreeSketch and Local Triangle Count Heavy Hitters
   1. Local Triangle Counting
   2. HyperLogLog Cardinality Sketches
   3. DegreeSketch and Triangle Counting
   4. Verification & Implementation Details
5. Sublinear κ-Path Centrality

Motivation: Local Triangle Counting

The Problem:

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

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

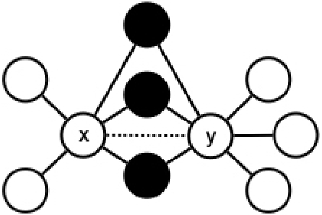
Existing Solutions:

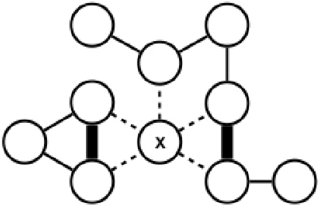
- Many exact distributed algorithms [AKM13, Pea17]
- Many approximate streaming algorithms via sampling [LK15, SERU17]
- ... and some utilizing both models [SHL+18, SLO+18]

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:
- $\rho(w) =$ in
- $M_i = \max\{M_i, \rho(w)\}$
- 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]

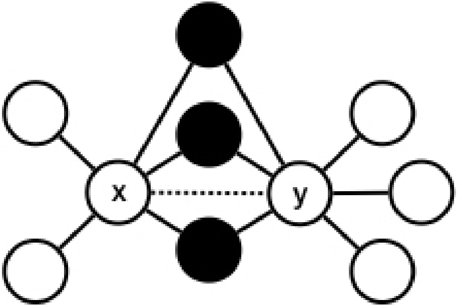
Useful results

- 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]

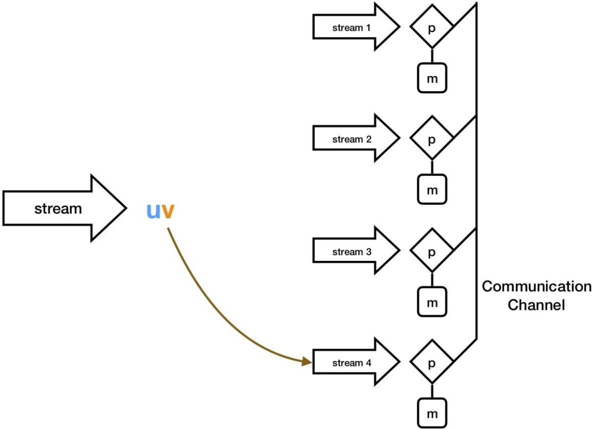
Set operation estimation with HLLs

Streaming sets X and Y with HLLs S_X and S_Y

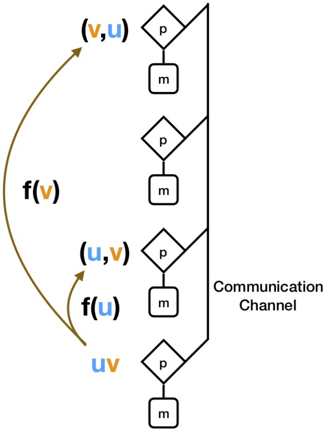
- $S_X \approx |X|$
- $S_Y \approx |Y|$
- $S_X \widetilde{\cup} S_Y \approx |X \cup Y|$
  - Same error guarantees
- $S_X \widetilde{\cap} S_Y \approx |X \cap Y|$
  - High variance if $|X \cap Y|$ small



DegreeSketch Accumulation

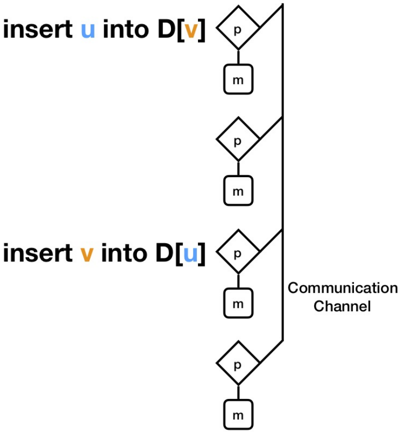


Partition stream across $\mathcal{P}$



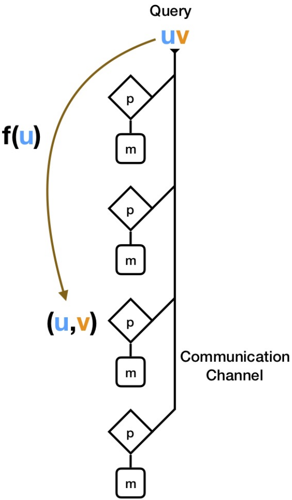
Distribute edges to endpoint owners

DegreeSketch Accumulation



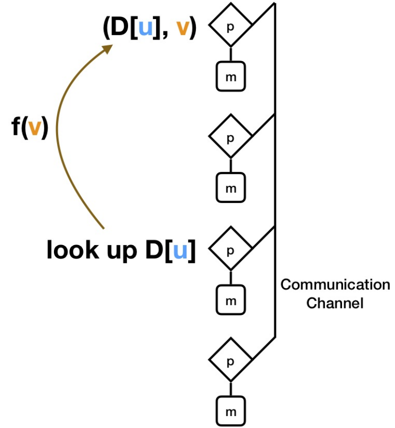
Insert into $\mathcal{D}$ for each vertex

DegreeSketch Query



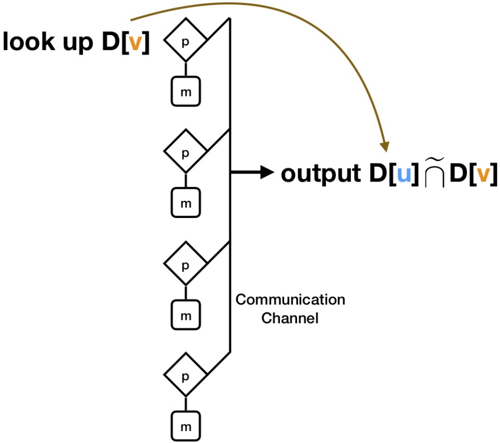
Query routes to one endpoint owner

DegreeSketch Query



Owner sends cardinality sketch to other endpoint owner

DegreeSketch Query



Final owner outputs intersection estimation

$\textsc{DegreeSketch}$ 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 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 σ
  - Assume $P \in \mathcal{P}$ gets substream σ_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!

Algorithm 1 Edge-Local Triangle Count Heavy Hitters

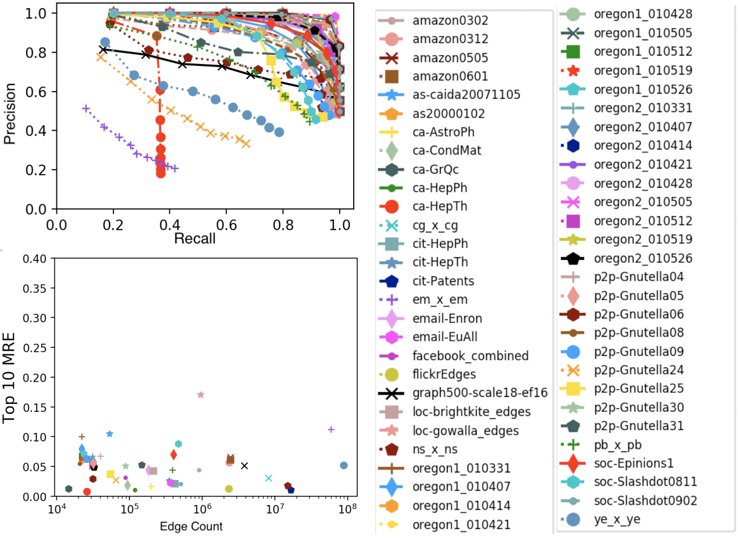
```
Accumulate $\mathcal{D}$ in distributed pass over $\sigma$
$H_k \leftarrow$ empty $k$-heap
$T \leftarrow 0$
parallel for $xy \in \sigma_P$ do        // second pass
    Send $(\mathrm{E}, xy)$ to $f(x)$
    for $(\mathrm{E}, xy) \in \mathcal{R}[P]$ do
        Send $(\mathrm{S}, xy, \mathcal{D}[x])$ to $f(y)$
    for $(\mathrm{S}, xy, \mathcal{D}[x]) \in \mathcal{R}[P]$ 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

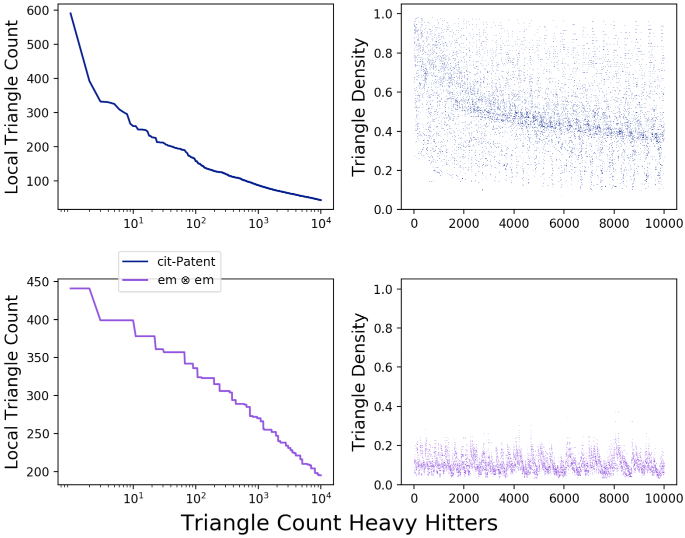
HPEC graph challenge results

Good performance on top 10 heavy hitters of *most* graphs

Why do some graphs perform worse?



High vs Low Triangle Density



Low triangle density → high variance

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
  - e.g. 2nd Neighborhood size

DegreeSketch to be open sourced

Overview

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 κ-Path Centrality

Idea: "Come at the problem sideways"

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

κ-path centrality

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

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

ℓ_p Sampling Sketches

ℓ_p sampling sketches

Sample from frequency vector $\mathbf{f}$ with probability relative to ℓ_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 ℓ_p norm sketches
- Use sketches to output $\textsc{CountSketch}$ argmax or FAIL

ℓ_p Sampling Sketches

ℓ_p sampling sketches

Sample from frequency vector **f** with probability relative to ℓ_p norm

- Sample $t_i \sim_R (0,$
- Rescale updates
- Accumulate TUG ... nd ℓ_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]

ℓ_p Sampling Sketches

ℓ_p sampling sketches

Sample from frequency vector **f** with probability relative to ℓ_p norm

- Sample $t_i \sim_R (0,$
- Rescale updates
- Accumulate TUG ... nd ℓ_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]

- ℓ_0 sketch requires $\widetilde{O}(\log(1/\delta))$ memory and update time
  - Useful for unweighted random hops
- ℓ_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 ℓ_p sketches can be accumulated in time independent of s

Distributed Accumulation ℓ_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 ℓ_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 FAIL, repeat
  - Once $\mathcal{A}[v]$ sketches are exhausted, P takes another pass over v's substream in disk memory

Avoids vertex cuts, exchanging communication for I/O

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 κ-Path Centrality

κ-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 κ-Path Centrality

Algorithm 2 Sublinear κ-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 κ-path centrality estimation algorithm
- Future Work
  - Applications for DegreeSketch
  - Sublinear random walk and random simple path implementation

Questions?

Shaikh Arifuzzaman, Maleq Khan, and Madhav Marathe.
Patric: A parallel algorithm for counting triangles in massive networks.
In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 529–538. ACM, 2013.

Elisabetta Bergamini, Henning Meyerhenke, and Christian L Staudt.
Approximating betweenness centrality in large evolving networks.
In *2015 Proceedings of the Seventeenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 133–146. SIAM, 2014.

Edith Cohen, Daniel Delling, Thomas Pajor, and Renato F Werneck.
Computing classic closeness centrality, at scale.
In *Proceedings of the second ACM conference on Online social networks*, pages 37–50. ACM, 2014.

Reuven Cohen, Liran Katzir, and Aviv Yehezkel.
A minimal variance estimator for the cardinality of big data set intersection.
In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 95–103. ACM, 2017.

Otmar Ertl.
New cardinality estimation algorithms for hyperloglog sketches.
arXiv preprint arXiv:1702.01284, 2017.

Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier.
Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm.
In *Discrete Mathematics and Theoretical Computer Science*, pages 137–156. Discrete Mathematics and Theoretical Computer Science, 2007.

Oded Green, Robert McColl, and David A Bader.
A fast algorithm for streaming betweenness centrality.
In *Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012 International Confernece on Social Computing (SocialCom)*, pages 11–20. IEEE, 2012.

Stefan Heule, Marc Nunkesser, and Alexander Hall.
Hyperloglog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm.

In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 683–692. ACM, 2013.

Hossein Jowhari, Mert Sağlam, and Gábor Tardos.
Tight bounds for lp samplers, finding duplicates in streams, and related problems.
In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 49–58. ACM, 2011.

Nicolas Kourtellis, Tharaka Alahakoon, Ramanuja Simha, Adriana Iamnitchi, and Rahul Tripathi.
Identifying high betweenness centrality nodes in large social networks.
Social Network Analysis and Mining, 3(4):899–914, 2013.

Nicolas Kourtellis, Gianmarco De Francisci Morales, and Francesco Bonchi.
Scalable online betweenness centrality in evolving graphs.
IEEE Transactions on Knowledge and Data Engineering, 27(9):2494–2506, 2015.

Kevin J Lang.
Back to the future: an even more nearly optimal cardinality estimation algorithm.
arXiv preprint arXiv:1708.06839, 2017.

Yongsub Lim and U Kang.
Mascot: Memory-efficient and accurate sampling for counting local triangles in graph streams.
In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 685–694. ACM, 2015.

Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski.
Pregel: a system for large-scale graph processing.
In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010.

Morteza Monemizadeh and David P Woodruff.
1-pass relative-error lp-sampling with applications.

In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1143–1160. SIAM, 2010.

Roger Pearce.
Triangle counting for scale-free graphs at scale in distributed memory.
In *High Performance Extreme Computing Conference (HPEC), 2017 IEEE*, pages 1–4. IEEE, 2017.

Roger Pearce, Maya Gokhale, and Nancy M Amato.
Faster parallel traversal of scale free graphs at extreme scale with vertex delegates.
In *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*, pages 549–559. IEEE, 2014.

Benjamin W. Priest, Roger Pearce, and Geoffrey Sanders.
Estimating edge-local triangle count heavy hitters in edge-linear time and almost-vertex-linear space.
In *High Performance Extreme Computing Conference (HPEC), 2018 IEEE*. IEEE, 2018.

Benjamin W. Priest, Roger Pearce, and Geoffrey Sanders.

DegreeSketch: Distributed cardinality sketches on graphs, with applications to counting triangles.
In *In Preparation for ACM SigKDD 2019*. ACM, 2019.

Jason Qin, Denys Kim, and Yumei Tung.
Loglog-beta and more: A new algorithm for cardinality estimation based on loglog counting.
arXiv preprint arXiv:1612.02284, 2016.

Matteo Riondato and Evgenios M Kornaropoulos.
Fast approximation of betweenness centrality through sampling.
Data Mining and Knowledge Discovery, 30(2):438–475, 2016.

Lorenzo De Stefani, Alessandro Epasto, Matteo Riondato, and Eli Upfal.
Trièst: Counting local and global triangles in fully dynamic streams with fixed memory size.
ACM Transactions on Knowledge Discovery from Data (TKDD), 11(4):43, 2017.

Kijung Shin, Mohammad Hammoud, Euiwoong Lee, Jinoh Oh, and Christos Faloutsos.
Tri-Fly: Distributed estimation of global and local triangle counts in graph streams.
In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 651–663. Springer, 2018.

Kijung Shin, Euiwoong Lee, Jinoh Oh, Mohammad Hammoud, and Christos Faloutsos.
Dislr: Distributed sampling with limited redundancy for triangle counting in graph streams.
arXiv preprint arXiv:1802.04249, 2018.

Daniel Ting.
Towards optimal cardinality estimation of unions and intersections with sketches.
In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1195–1204. ACM, 2016.

Hoa Vu.

Data stream algorithms for large graphs and high dimensional data.
2018.

Wei Wei and Kathleen Carley.
Real time closeness and betweenness centrality calculations on streaming network data.
In *Proceedings of the 2014 ASE Big-Data/SocialCom/Cybersecurity Conference, Stanford University*, 2014.

Qingjun Xiao, You Zhou, and Shigang Chen.
Better with fewer bits: Improving the performance of cardinality estimation of large data streams.
In *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*, pages 1–9. IEEE, 2017.

Yuichi Yoshida.
Almost linear-time algorithms for adaptive betweenness centrality using hypergraph sketches.
In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1416–1425. ACM, 2014.