DegreeSketch: Distributed Cardinality Sketches on Graphs with Applications

Benjamin W. Priest

May 1, 2019

Abstract

The identification of important vertices or edges is a ubiquitous problem in the analysis of graphs. There are many application-dependent measures of importance, such as centrality indices (e.g. degree centrality, closeness centrality, betweenness centrality, and eigencentrality) and local triangle counts, among others. Traditional computational models assume that the entire input fits into working memory, which is impractical for very large graphs. The distributed memory model and streaming model are popular solutions to this problem of scale. In the distributed memory model a collection of processors partition the graph and must optimize communication in addition to execution time. The data stream model assumes only sequential access to the input, which is handled in small chunks. Data stream algorithms use sublinear memory and a small number of passes and seek to optimize update time, query time, and post processing time.

In this dissertation, we consider the application of distributed data stream algorithms to the sublinear approximation of several centrality indices, local triangle counts, and the simulation of random walks. We pay special attention to the recovery of *heavy hitters* - the largest elements relative to the given index.

The first part of this dissertation focuses on serial graph stream algorithms. We present new algorithms providing streaming approximations of degree centrality and a semi-streaming constant-pass approximation of closeness centrality. We achieve our results by way of counting sketches and sampling sketches.

The second part of this dissertation considers vertex-centric distributed graph stream algorithms. We develop hybrid pseudo-asynchronous communication protocols tailored to managing communication on distributed graph algorithms with asymmetric computational loads. We use this protocol as a framework to develop distributed streaming algorithms utilizing cardinality sketches. We present new algorithms for estimating local neighborhood sizes, as well as vertex- and edge-local triangle counts, with special attention paid to heavy hitter recovery. We also utilize reservoir sampling and ℓ_p sampling sketches to optimize the semi-streaming simulation of many random walks in parallel in distributed memory. We use these algorithms to approximate K-path centrality as a proxy for recovery the top-k betweenness centrality elements.

1 Introduction

Many modern computing problems focus on complex relationship networks arising from real-world data. Many of these complex systems such as the Internet, communication networks, logistics and transportation systems, biological systems, epidemiological models, and social relationship networks map naturally onto graphs [**?**]. A natural question that arises in the study of such networks is how to go about identifying "important" vertices and edges. How one might interpret importance within a graph is contingent upon its domain. Accordingly, investigators have devised a large number of importance measures that account for different structural properties. These measures implicitly define an ordering on graphs, and typically only the top elements vis-á-vis the ordering are of analytic interest.

However, most traditional RAM algorithms scale poorly to large datasets. This means that very large graphs tend to confound standard algorithms for computing various important orderings. Newer computational models such as the data stream model and the distributed memory model were introduced to address these scalability concerns. The data steam model assumes only sequential access to the data, and permits a sublinear amount of additional working memory. The time to update, query, and post process this data

structure, as well as the number of passes and amount of additional memory are the important resources to optimize in the data stream model. The data stream model is a popular computational model for handling scalability in sequential algorithms. The distributed memory model partitions the data input across several processors, which may need to subsequently communicate with each other. The amount of communication is an important optimization resource. In practical terms, minimizing the amount of time processors spend waiting on their communication partners is also important.

Although both models have been applied to very large graphs independently, there is relatively little literature focusing on the union of the two models of computation. In this work we devise distributed data stream algorithms to approximate orderings of vertices and edges of large graphs. We focus in particular on recovering the heavy hitters of these orderings. We consider the sublinear approximation of classic centrality scores, as well as local and global triangle counts. We also describe space-efficient methods for sampling random walks and subtrees in scale-free vertex-centric distributed graphs, and their application to estimating some centrality indices.

2 Data Stream Models

The data stream model: A stream $\sigma = \langle a_1, a_2, \ldots, a_m \rangle$ is a sequence of elements in the universe $\mathcal{U}$, $|\mathcal{U}| = n$. We assume throughout that that the hardware has working memory storage capabilities $o(\min\{m, n\})$. We will use the notation $[p] = \{1, 2, \ldots, p-1, p\}$ for $p \in \mathbb{Z}_{>0}$ throughout for compactness. For $t \in [m]$, we will sometimes refer to the state of σ after reading t updates as $\sigma(t)$. A streaming algorithm $\mathcal{A}$ accumulates a data structure $\mathcal{S}$ while reading over σ. We will sometimes use the notation $\mathcal{D}(\sigma)$ to indicate the data structure state after $\mathcal{A}$ has accumulated σ. Authors generally assume $|\mathcal{D}| = \widetilde{O}(1) = O(\log m + \log n)$, where here the tilde suppresses logarithmic factors. We will also adopt the convention that, except where noted otherwise, we present space complexities in terms of machine words rather than bits. Except where noted otherwise, we will assume the base 2 logarithm in our presentation.

The semi-streaming model: Unfortunately, logarithmic memory constraints are not always possible. In particular, it is known that many fundamental properties of complex structured data such matrices and graphs require memory linear in some dimension of the data [**?**, **?**]. In such cases, the logarithmic requirements of streaming algorithms are sometimes relaxed to $O(n \operatorname{polylog} n)$ memory, where $\operatorname{polylog} n = \Theta(\log^c n)$ for some constant c. In the case of matrices, here n refers to one of the matrix's dimensions, whereas for graphs n refers to the number of vertices. This is usually known as the *semi-streaming model*, although some authors also use the term to refer to $O(n^{1+\gamma})$ for small γ [**?**, **?**].

The frequency vector and dynamic streams: A stream σ is often thought of as updates to a hypothetical frequency vector $\mathbf{f}(\sigma)$, which holds a counter for each element in $\mathcal{U}$. We will drop the parameterization of $\mathbf{f}$ where it is clear. We will sometimes parameterize $\mathbf{f}(t)$ to refer to $\mathbf{f}$ after reading $t \in [m]$ updates from σ. $\mathcal{D}$ can be thought of as a lossy compression of $\mathbf{f}$ that approximately preserves some statistic thereof. The stream σ's elements are of the form (i, c), where i is an index of $\mathbf{f}$ (an element of $\mathcal{U}$), $c \in [-L, \ldots, L]$ for some integer L, and (i, c) indicates that $\mathbf{f}_i \leftarrow \mathbf{f}_i + c$. Such a stream σ is called a *turnstile* or sometimes *dynamic* stream. In the *cash register* model only positive updates are permitted, whereas in the *strict* turnstile model all elements of $\mathbf{f}$ are guaranteed to retain nonnegativity.

Data sketching: Let $\circ$ be the concatenation operator on streams. For $\mathcal{A}$ a streaming algorithm, we call its data structure $\mathcal{S}$ a *sketch transform* if there is an operator $\oplus$ such that, for any streams σ_1 and σ_2,

$$\mathcal{S}(\sigma_1) \oplus \mathcal{S}(\sigma_2) = \mathcal{S}(\sigma_1 \circ \sigma_2). \tag{1}$$

$\mathcal{S}$ is usually determined by sampling hash functions from a suitable hash family. We will call the distribution over such samples Π a *sketch distribution*. The accumulated object $\mathcal{S}(\sigma)$ is a sketch data structure. We will sometimes abuse terms and refer to Π, $\mathcal{S}$, and $\mathcal{S}(\sigma)$ as *sketches*. The space of all possible sketch data structures given a sketch transform $\mathcal{S}$ is the sketch space $\mathbb{S}_{\mathcal{S}}$. $(\mathbb{S}_{\mathcal{S}}, \oplus)$ forms a commutative monoid, where the operator identity is the result of applying $\mathbb{S}$ to an empty stream.

Linear sketching: A sketch distribution Π is a *linear sketch* distribution if $\mathcal{S} \sim \Pi$ is a linear function of frequency vectors $\mathbf{f}(\cdot)$ of fixed dimension. For streams σ_1 and σ_2 with frequency vectors $\mathbf{f}_1$ and $\mathbf{f}_2$, scalars a and b, and linear sketch transform $\mathcal{S}$,

$$a\mathcal{S}(\mathbf{f}_1) + b\mathcal{S}(\mathbf{f}_2) = \mathcal{S}(a\mathbf{f}_1 + b\mathbf{f}_2). \tag{2}$$

The sketch space and the + operator form a commutative group, where the operator identity is the sketch transform applied to a zero vector.

The graph stream model: In this model, the stream σ consists of m edge insertions on n vertices. This is sometimes termed the *insert only* model in the literature. The *dynamic graph stream model* also allows edge deletions. If the graph is weighted, the stream updates the weight of the corresponding edge, possibly bringing it into existence or, in the case of dynamic streams, deleting it.

DegreeSketch and local triangle count heavy hitters: Counting global and local triangles is a canonical problem in both the random access and data stream models. In the data stream model it is known that $\Omega(n^2)$ space is required to even decide whether a graph has any triangles [**?**]. Vertex-local triangle counts are similarly fraught in the data stream model, as they require $\Omega(n)$ space to even write them down. Recent advances in the graph stream literature achieve low variance via clever sampling from edge streams [**?**, **?**, **?**], and have been distributed to achieve better variance [**?**, **?**].

We examine a different line of analysis altogether: utilizing *cardinality sketches* to estimate local triangle counts via a sublinearization of the set intersection method. We discuss trade-offs between different estimators of the intersection of cardinality sketches, as well as different cardinality sketches themselves. We develop DegreeSketch, a distributed sketch data structure composed of cardinality sketches that can answer point queries about unions and intersections of vertex neighborhoods in a manner reminiscent of CountSketch.

We implement DegreeSketch using the famous HyperLogLog sketch [**?**]. We explore various algorithmic optimizations to HyperLogLog, including improved estimators [**?**, **?**, **?**, **?**], sparse register implementation [**?**], and compressed registers [**?**]. We introduce novel improvements to the compressed register implementation described in [**?**], affording lossless merging of compressed sketches. We also discuss the tradeoffs of using different cardinality sketches in implementations, such as MinCount [**?**].

We demonstrate DegreeSketch as a tool for estimating local neighborhood sizes, as well as the edge- and vertex-local triangle count heavy hitters of large graphs.

1. **Local neighborhood size and neighborhood function**. We estimate the local neighborhood sizes and global neighborhood function of a graph using linear time and $O(n(\varepsilon^{-2}\log\log n+\log n))$ worst-case distributed memory

2. **Edge-local triangle count heavy hitters**. We estimate the edge-local triangle count heavy hitters of a graph using the same asymptotic time and memory requirements.

3. **Vertex-local triangle counts**. We estimate the vertex-local triangle counts (or, alternately, the heavy hitters) of a graph using the same asymptotic time and memory requirements.

We analyze the performance of these algorithms on numerous large graphs, and discuss many practical optimizations to the underlying algorithms. We also discuss the limitations of the approach relating to high variance on small intersections and intersections of sets of differing sizes.

3 Background and Notation

This chapter introduces the basic concepts and notation that we will use throughout this document. We will lay out the basic graph, probability, and linear algebraic definitions and notation conventions that will be referenced later We also describe centrality indices as a class of functions on graphs. Finally, we describe k-universal hash families, an important concept for many sketches, as well as some approximation definitions.

4 Graph Definitions and Notation

Throughout this document we will consider the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{w})$. We assume that $\mathcal{G}$ has no self loops, and that where $|\mathcal{V}| = n$ and $|\mathcal{E}| = m$. For convenience of reference and indexing, we will often assume that $\mathcal{V} = [n]$ and $E = [m]$. We denote an edge connecting $x, y \in \mathcal{V}$ as $xy \in \mathcal{E}$. In general we will assume that $\mathcal{G}$ is an undirected graph, except where noted otherwise. When we want to specify a direction on an edge, we will use the tuple notation (x, y) for $x, y \in \mathcal{V}$. If $\mathcal{G}$ is a weighted graph, then $\mathbf{w} \in \mathbb{R}^{\binom{n}{2}}$ is the vector of edge

weights. For $x, y \in \mathcal{V}$, $\mathbf{w}_{xy} \in \mathbb{R}_{\geq 0}$ is the weight associated with the edge xy if $xy \in \mathcal{E}$, and is zero otherwise. If $\mathcal{G}$ is unweighted, then $\mathbf{w}_e = 1$ for every $e \in \mathcal{E}$.

Let $A \in \mathbb{R}^{n \times n}$ be the *adjacency matrix* of $\mathcal{G}$, where $A_{x,y} = \mathbf{w}_{xy}$. We will adopt the convention that, if $\mathcal{G}$ is unweighted, then the columns of A correspond to the out edges whereas the rows of A correspond to the in edges. Hence, $A_{:,x}$ is the out adjacency vector of vertex x, and $A_{x,:}$ is its in adjacency vector.

For $\mathcal{G}$ an unweighted graph, let $D \in \mathbb{R}^{n \times n}$ be a diagonal matrix, where $D_{x,x}$ is the *degree* or *valency* of vertex $x \in \mathcal{V}$, which can be computed as the row sum of the xth row of A. We define $L = D - A$ as the *Laplace Matrix* or *Laplacian* of $\mathcal{G}$.

Consider the signed vertex-edge incidence matrix, $B \in \mathbb{R}^{\binom{n}{2} \times n}$, given by

$$B_{xy,z} = \begin{cases} 1 & \text{if } xy \in \mathcal{E} \text{ and } x = z \\ -1 & \text{if } xy \in \mathcal{E} \text{ and } y = z \\ 0 & \text{else.} \end{cases} \tag{3}$$

Here we let x, y, and z range over $\mathcal{V}$. Let $W \in \mathbb{R}^{n \times n}$ be a diagonal matrix such that $W_{x,y} = \sqrt{w_{xy}}$. Then if G is undirected, we can alternatively write the Laplacian as

$$L = BWW^T B^T. \tag{4}$$

If $\mathcal{G}$ is unweighted, then we can simply write $L = BB^T$.

A *path* in $\mathcal{G}$ is a series of edges $(x_1x_2, x_2x_3, \ldots, x_{\ell-1}x_\ell)$ where the tail of each edge is the head of the following edge in the path. The length, alternatively weight, of a path is the sum of the weights of all of its edges. If G is unweighted, this is simply the number of edges in the path. We can equivalently identify the path with the series of vertices $(x_1, x_2, \ldots, x_\ell)$, where an edge links each x_i, x_{i+1} pair. For vertices $x, y \in \mathcal{V}$, the *distance* $d_{\mathcal{G}}(x, y)$ between x and y in $\mathcal{G}$ is the length of the shortest path that begins at x and ends with y. There may be more than one such path. If there is no path connecting x to y in $\mathcal{G}$, then we say that $d_{\mathcal{G}}(x, y) = \infty$. If the graph is clear from context, we may omit the subscripts and write $d(x, y)$. We call a path *simple* if it visits every vertex no more than once.

5 Centrality Indices

A centrality index is any map $\mathcal{C}$ that assigns to every $x \in \mathcal{V}$ a nonnegative score. The particulars of $\mathcal{C}$ are usually assumed to be conditioned only on the structure of $\mathcal{G}$. Consequently, we can identify the centrality index on $\mathcal{G}$ as a function $\mathcal{C}_{\mathcal{G}} : \mathcal{V} \to \mathbb{R}_{\geq 0}$. For $x \in \mathcal{V}$, we will call $\mathcal{C}_{\mathcal{G}}(x)$ the centrality score of x in $\mathcal{G}$. Typically, for $x, y \in \mathcal{V}$, $\mathcal{C}_{\mathcal{G}}(x) > \mathcal{C}_{\mathcal{G}}(y)$ implies that x is more important than y in $\mathcal{G}$ with respect to the property that $\mathcal{C}$ measures. We will generally drop the subscript from $\mathcal{C}$ when it is clear from context. It is important to note that if $\mathcal{G}$ changes, so may the mapping $\mathcal{C}$. At times, we will write $\mathcal{C}(\mathcal{G})$ or $\mathcal{C}(\mathcal{V})$ to denote the set of all centrality scores of the vertices in $\mathcal{G}$.

Researchers have considered more exotic centrality indices that rely on metadata, such as vertex and edge colorings [**?**]. Such notions of centrality are most likely out of scope for the research proposed by this document.

6 Vector and Matrix definitions and notation

For a vector $v \in \mathbb{R}^n$, we denote the ℓ_p norm as follows:

$$\|v\|_p = \left(\sum_{i=1}^{n} |v_i|^p \right)^{1/p}. \tag{5}$$

As $p \to 0$, this quantity converges to the special case of the ℓ_0 norm:

$$\|v\|_0 = \sum_{i=1}^{n} v_i^0 = |\{i \in [n] \mid v_i \neq 0\}|. \tag{6}$$

Here we define $0^0 = 0$. Throughout this document we will mostly be concerned with the ℓ_0 and ℓ_1 norms of matrix rows and columns. The p-th frequency moment F_p of a vector v is related to its ℓ_p norm in the following way:

$$F_p(v) = \|v\|_p^p = \sum_{i=1}^{n} v_i^p. \tag{7}$$

We will also sometimes be interested in matrix norms. For $i \in [n]$ and $j \in [m]$, we will write the i,jth element of M as $M_{i,j}$. We will also write the ith row and jth column of M as $M_{i,:}$ and $M_{:,j}$, respectively. For a matrix $M \in \mathbb{R}^{n \times m}$, we define the Fröbenius norm as follows:

$$\|M\|_F = \left(\sum_{i=1}^{n} \sum_{j=1}^{m} M_{i,j}^2 \right)^{1/2}. \tag{8}$$

Given $A \in \mathbb{R}^{n \times d}$, let $A = U \Sigma V^T$ be its singular value decomposition (SVD), where $\Sigma \in \mathbb{R}^{n \times n}$ is a diagonal matrix and U and V are orthonormal. Set $A_k = U_k \Sigma_k V_k^T$, where U_k and V_k are the leading k columns of U and V, respectively, and $\Sigma_k \in \mathbb{R}^{k \times k}$ is a diagonal matrix whose entries are the first k entries of Σ. A_k is known to solve the optimization problem

$$\min_{\widetilde{A} \in \mathbb{R}^{n \times d} : \mathrm{rank}(\widetilde{A}) \leq k} \|A - \widetilde{A}\|_F.$$

That is, A_k is the rank-k matrix which has the smallest Fröbenius residual with A. This is also true of the spectral norm. We will sometimes denote the rank-k truncated SVD of a product of matrices $A_1 \cdots A_n$ as $[A_1 \cdots A_n]_k$. We use the notation $A^+ = V \Sigma^{-1} U^T$ to denote the Moore-Penrose Pseudoinverse of A.

7 DegreeSketch with Applications to Neighborhood Approximation and Local Triangle Count Heavy Hitter Estimation

In this chapter we present DegreeSketch, a semi-streaming distributed sketch datastructure and demonstrate its utility for estimating local triangle count heavy hitters. DegreeSketch consists of vertex-centric cardinality sketches distributed across a set of processors. While other semi-streaming approaches to estimating local triangle counts depend on sampling, DegreeSketch is a persistent queryable data structure. We discuss the advantages and limitations of this approach, and present empirical results.

8 Introduction and Related Work

Counting the number of triangles in simple graphs is a canonical problem in both the RAM and streaming models. A "triangle" is a trio of co-adjacent vertices, and is the smallest nontrivial community structure. Consequently, triangles and triangle counting arise often in applications. Both the global count of triangles and the vertex-local counts, i.e. the number of triangles incident upon each vertex, are key to network analysis and graph theory topics such as cohesiveness [**?**], global and local clustering coefficients [**?**], and trusses [**?**]. These counts are also directly useful in many applications, such as spam detection [**?**], community discovery [**?**, **?**], and protein interaction analysis [**?**].

Although not traditionally considered a centrality index, one can think of the number of triangles incident upon a vertex as a generalization of its degree centrality. To wit, we define the *triangle count centrality* of a vertex $x \in \mathcal{V}$ as

$$\mathcal{C}^{\mathrm{TRI}}(x) = |\{yz \in \mathcal{E} \mid xy, zx \in \mathcal{E} \wedge |\{x, y, z\}| = 3\}|. \tag{9}$$

We can consider the triangle count centrality of edges as well, defined similarly as

$$\mathcal{C}^{\mathrm{TRI}}(xy) = |\{z \in \mathcal{V} \setminus \{x, y\} \mid yz, zx \in \mathcal{E}\}|. \tag{10}$$

We will also refer to the total number of triangles in a graph

$$\mathcal{T}_{\mathcal{G}} = |\{\{xy, yz, zx\} \in \mathcal{E} \mid |\{x, y, z\}| = 3\}|. \tag{11}$$

Although many exact algorithms have been proposed for the triangle counting problem [**?**, **?**, **?**, **?**, **?**], their time complexity is superlinear in the number of edges, $O(m^{\frac{3}{2}})$. Consequently, these analyses are expensive on very large graphs, particularly if they are dense. While there is a rich literature centered on the exact global and local triangle counting problem, we will not exhaustively recall it in this document.

In order to avoid the dreaded superlinear scaling of exact algorithms, many researchers have turned to approximation. Several streaming local triangle counting algorithms have been proposed in recent years. These serial streaming algorithms maintain a limited number of sampled edges from an edge stream. Streaming global triangle estimation algorithms have arisen that sample edges with equal probability [**?**], sample edges with probability relative to counted adjacent sampled edges and incident triangles [**?**], and sample edges along with paths of length two [**?**]. The first proposed semi-streaming local triangle estimation algorithm relies upon min-wise independent permutations accumulated over a logarthmic number of passes [**?**]. More recently, true single-pass algorithms have arisen such as MASCOT [**?**], which maintains local estimates that are updated whether an observed edge is sampled or not. Similarly, TRIÉST [**?**] utilizes reservoir sampling, possibly disposing of sampled edges when a new edge is observed if system memory is saturated. This affords robustness to dynamic streams, as well as reducing variance. None of these algorithms perform edge-local triangle counting.

While many distributed global and vertex-local triangle counting algorithms have been proposed, the overwhelming majority store the graph in distributed memory and return exact solutions using MAPREDUCE [**?**] or distributed memory [**?**, **?**]. Recently, the study of distributed streaming vertex-local triangle counting was intiated in earnest with the presentation of TRY-FLY [**?**], which maintains parallel instantiations of $\text{TRIÉST}_{\text{IMPR}}$. A controller node feeds partitions of the graph stream to each of these instances in order to boost estimation accuracy and speed. DISLR [**?**] improves upon TRY-FLY by introducing limited redundancy into the graph stream partitions. Although TRY-FLY and DISLR are the most similar examples in the literature to our approach, we use distinct methods.

Our approach is fundamentally different, depending upon sketching rather than sampling as its core primitive. While the sampling approaches produce estimates, we produce a leave-behind queryable data structure similar in interface to COUNTMINSKETCH.

9 DEGREESKETCH via Distributed Cardinality Sketches

We take a different approach to estimating local triangle counts, relying upon sketches instead of sampling. We introduce DEGREESKETCH, a cardinality sketch-based distributed data structure trained on graph streams. We will first describe DEGREESKETCH at a high level, describe algorithms that utilize it in Sections **??**, **??**, and **??** and then discuss implementation details using the HYPERLOGLOG cardinality sketch in Section **??**.

DEGREESKETCH maintains a data structure $\mathcal{D}$ that can be queried for an estimate of a vertex's degree, similar in interface to the celebrated COUNTSKETCH [**?**] and COUNTMINSKETCH [**?**]. This data structure is organized not unlike the sampling sketch approach to graph sparsification discussed in Section **??**, with sketches summarizing the local information for each $x \in \mathcal{V}$. For each $x \in \mathcal{V}$ we maintain a cardinality sketch $\mathcal{D}[x]$, which affords the approximation of $\mathbf{d}_x$, the degree of x.

It is known that any data structure that provides relative error guarantees for the cardinality of a multiset with n unique elements requires $O(n)$ space [**?**]. Consequently, investigators have developed many so-called *cardinality sketches* that provide such relative error guarantees while admitting a small probability of failure, such as PCSA [**?**], MinCount [**?**], LogLog [**?**], Multiresolution Bitmap [**?**], HyperLogLog [**?**], and the space-optimal solution of [**?**]. While all these cardinality sketches have a natural union operation that allows one to combine the sketches of two multisets into a sketch of their union, most have no closed intersection operation.

For the purposes of discussion, we will abstract the particulars of cardinality sketches until Section **??**. We assume that the sketches provide a ε-approximation of the number of unique items in a stream using $\Omega(\varepsilon^{-2})$ space. We assume that the sketches support an INSERT operation to add elements, a MERGE operation to combine sketches, an ESTIMATE operation to estimate cardinalities, and an ESTIMATEINTERSECTION operation to estimate intersection cardinalities. For reasons that will be described in Section **??**, we do not assume that the ESTIMATEINTERSECTION procedure has the same error and variance properties as the guarantees for ESTIMATE.

We will describe the accumulation of a DegreeSketch instance on a universe of processors $\mathcal{P}$. We assume that the undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is given by a stream $\boldsymbol{\sigma}$. $\boldsymbol{\sigma}$ is further partitioned by some unknown means into $|\mathcal{P}|$ substreams. We assume that each processor $P \in \mathcal{P}$ has send and receive buffers $\mathcal{S}[P]$ and $\mathcal{R}[P]$, respectively. We will make no assumptions in the following algorithms how processors handle switching context between processing and sending and receiving communication. In our implementations we use the software package YGM, developed as a part of this thesis and described in Chapter **??**. YGM manages the send and receive buffers, as well as switching contexts, in a manner that is opaque to the client algoirthm. We further assume that there is some partitioning of vertices to processors $f : \mathcal{V} \to \mathcal{P}$. We will occasionally abuse notation and use $f(P)$ to describe the set of vertices that map to $P \in \mathcal{P}$. We make no assumptions about the particulars of f, noting that vertex partitionings are a subject of intense academic scrutiny as discussed in Section **??**. In effect, our algorithms are designed to work alongside any decent partitioning mechanism.

Algorithm **??** describes the distributed accumulation of a DegreeSketch instance. In a distributed pass over the partitioned stream, processors use the partition function f to send edges to the cognizant processors for each endpoint. These processors each maintain cardinality sketches for their assigned vertices. When $P \in \mathcal{P}$ receives an edge $xy \in \mathcal{E}$ where $f(x) = P$, it performs Insert$(\mathcal{D}[x], y)$. Once all processors are done reading and communicating, $\mathcal{D}$ is accumulated.

Algorithm 1 DegreeSketch Accumulation

Input: $\boldsymbol{\sigma}$ - edge stream divided into $|\mathcal{P}|$ substreams
$\mathcal{P}$ - universe of processors
$\mathcal{S}$ - distributed dictionary mapping $\mathcal{P}$ to send queues
$\mathcal{R}$ - distributed dictionary mapping $\mathcal{P}$ to receive queues
f - function mapping $\mathcal{V} \to \mathcal{P}$
Output: $\mathcal{D}$ - accumulated DegreeSketch

Send Context for $P \in \mathcal{P}$:
while $\mathcal{S}[P]$ is not empty **do**
  $(W, xy) \leftarrow \mathcal{S}[P]$.pop()
  $\mathcal{R}[W]$.push (xy)
Receive Context for $P \in \mathcal{P}$:
while $\mathcal{R}[P]$ is not empty **do**
  $xy \leftarrow \mathcal{R}[P]$.pop()
  if $!\exists\mathcal{D}[x]$ **then**
    $\mathcal{D}[x] \leftarrow$ empty sketch
  Insert$(\mathcal{D}[x], y)$
Accumulation Context for $P \in \mathcal{P}$:
$\mathcal{D} \leftarrow$ empty DegreeSketch dictionary
while σ_P has unread element uv **do**
  $U \leftarrow f(u)$
  $\mathcal{S}[P]$.push (U, uv)
  $V \leftarrow f(v)$
  $\mathcal{S}[P]$.push (V, vu)
return $\mathcal{D}$

DegreeSketch can be implemented with any cardinality sketch that admits some form of union and intersection estimation. In fact, the algorithms in Sections **??** and **??** do not even require a closed merge operation. In our experiments, we focus on the well-known HyperLogLog or HLL cardinality sketches. We use HLLs for implementation due to several attractive features:

1. Small register size
2. Simple merge operation

3. Sparse register representation

4. Adequate intersection estimator

We introduce HLL and discuss these features in greater detail in Section **??**. FIrst, however, we describe algorithms utilizing DegreeSketch for neighborhood size estimation in Section **??**, recovering edge-local triangle count heavy hitters in Section **??**, and finally recovering vertex-local triangle count heavy hitters in Section **??**.

10 Neighborhood Size Estimation

Let the neighborhood function $\mathcal{N}_{\mathcal{G}}(t)$ be defined by

$$\mathcal{N}_{\mathcal{G}}(t) = |\{(x, y) \in \mathcal{V} \times \mathcal{V} | d_{\mathcal{G}}(x, y) < t\}|. \tag{12}$$

This function provides data about how fast the "average ball" around each $x \in \mathcal{V}$ expands, permitting the estimation of G's properties such as the effective diameter [**?**]. We will drop the subscripts where they are clear. For $t \in \mathbf{N}$, let $\mathcal{C}_t^{\text{NBHD}}(x)$ be the *local* neighborhood function defined as

$$\mathcal{C}_t^{\text{NBHD}}(x) = |\{y \in \mathcal{V} | d_{\mathcal{G}}(x, y) < t\}|. \tag{13}$$

While not usually treated as a centrality index, we can think of the tth neighborhood size of x as a generalization of its degree centrality. In particular, we have that

$$\mathcal{N}(t) = \sum_{x \in \mathcal{V}} \mathcal{C}_t^{\text{NBHD}}(x). \tag{14}$$

The ANF [**?**] and HyperANF [**?**] algorithms estimate $\mathcal{C}_t^{\text{NBHD}}(x)$ for each $x \in \mathcal{V}$ using Flajolet-Martin and HyperLogLog cardinality sketches, respectively, and produce estimates of $\mathcal{N}(t)$ of the form Eq. (**??**).

Let $\mathcal{D}$ be an instance of DegreeSketch as described, so that for $x \in \mathcal{V}$, $\mathcal{D}[x]$ is a cardinality sketch of the adjacency set of x. Assume that there is an approximate union operator $\widetilde{\cup}$. If we have $A_{:,x}$, then we can compute an estimate of $\mathcal{C}_2^{\text{NBHD}}(x)$ by computing

$$\widetilde{\mathcal{C}}_2^{\text{NBHD}}(x) = \textsc{Estimate}\left(\widetilde{\bigcup}_{y:xy \in \mathcal{E}} \mathcal{D}[y]\right). \tag{15}$$

Higher-order merge operations described by Eq. (**??**) form the core of the ANF [**?**] and HyperANF [**?**] algorithms. We will restrict further analysis to HyperANF. HyperANF uses HyperLogLog sketches similar to DegreeSketch to estimate the t-hop neighborhood sizes of all vertices by way of a iterative sketch merging procedure, which is useful for applications such as edge prediction in social networks [**?**] and probabilistic distance calculations [**?**, **?**]. However, HyperANF also requires storing the whole graph in memory in addition to the DegreeSketch data structure, and is optimized for shared but not distributed memory. It also does not take advantage of recent advances in HyperLogLog joint estimation that permit reasonable estimation of sketch intersections as well as unions.

Algorithm **??** uses DegreeSketch to recreate the behavior of HyperANF. After accumulating $\mathcal{D}^1$, an instance of DegreeSketch, the algorithm takes a number of additional passes over $\boldsymbol{\sigma}$. For t starting at 2, we accumulate

$$\mathcal{D}^t[x] = \widetilde{\bigcup}_{y:xy \in \mathcal{E}} \mathcal{D}^{t-1}[y] \tag{16}$$

by way of a message-passing scheme similar to Algorithm **??**. When $P \in \mathcal{P}$ receives an edge $xy \in \mathcal{E}$ where $f(x) = P$, it forwards $\mathcal{D}^{t-1}[x]$ to $Y = f(y)$. When Y receives this sketch, it merges it into its next layer local sketch for y, $\mathcal{D}^t[y]$, computing Eq. (**??**) once all messages are processed. By construction, we have that

$$\mathcal{D}^t[x] = \widetilde{\bigcup}_{y:d(x,y)=s<t-1} \mathcal{D}^s[y]. \tag{17}$$

Algorithm 2 DegreeSketch Neighborhood Approximation

Input: $\boldsymbol{\sigma}$ - edge stream divided into $|\mathcal{P}|$ substreams
$\mathcal{P}$ - universe of processors
$\mathcal{D}^1$ - accumulated DegreeSketch
$\mathcal{S}$ - distributed dictionary mapping $\mathcal{P}$ to send queues
$\mathcal{R}$ - distributed dictionary mapping $\mathcal{P}$ to receive queues
f - function mapping $\mathcal{V} \to \mathcal{P}$
Output: $\mathcal{N}(t)$ for all $t > 1$

Send Context for $P \in \mathcal{P}$:
while $\mathcal{S}[P]$ is not empty **do**
  if next message is an Edge **then**
    $(W, xy, t) \leftarrow \mathcal{S}[P]$.pop()
    $\mathcal{R}[W]$.push (Edge, xy, t)
  else if next message is a Sketch **then**
    $(W, \mathcal{D}[x], y, t) \leftarrow \mathcal{S}[P].pop()$
    $\mathcal{R}[W]$.push (Sketch, y, t)
Receive Context for $P \in \mathcal{P}$:
while $\mathcal{R}[P]$ is not empty **do**
  if next message is an Edge **then**
    $(xy, t) \leftarrow \mathcal{R}[P]$.pop()
    $Y \leftarrow f(y)$
    $\mathcal{S}[P]$.push $\big(\text{Sketch}, (Y, \mathcal{D}^{t-1}[x], y, t)\big)$
  else if next message is a Sketch **then**
    $(\mathcal{D}^{t-1}[x], y, t) \leftarrow \mathcal{R}[P]$.pop()
    $\mathcal{D}^t[y] \leftarrow \text{Merge}(\mathcal{D}^t[y], \mathcal{D}^{t-1}[x])$
Execution Context for $P \in \mathcal{P}$:
$t \leftarrow 1$
$\mathcal{N}(0) \leftarrow |\mathcal{V}|$
$\mathcal{N}_P(1) \leftarrow \sum_{x \in f(P)} \text{Estimate}(\mathcal{D}^1[x])$
$\mathcal{N}(1) \leftarrow \text{Reduce}(\text{Sum}, \mathcal{N}_P(1))$
while $\mathcal{N}(t) \neq \mathcal{N}(t-1)$ **do**
  $t \leftarrow t + 1$
  $\mathcal{D}^t \leftarrow$ empty DegreeSketch dictionary
  while σ_P has unread element uv **do**
    $U \leftarrow f(u)$
    $\mathcal{S}[P]$.push $(\text{Edge}, (U, uv, t))$
    $V \leftarrow f(v)$
    $\mathcal{S}[P]$.push $(\text{Edge}, (V, vu, t))$
  $\mathcal{N}_P(t) \leftarrow \sum_{x \in f(P)} \text{Estimate}(\mathcal{D}^t[x])$
  $\mathcal{N}(t) \leftarrow \text{Reduce}(\text{Sum}, \mathcal{N}_P(t))$
  replace $\mathcal{D}^{t-1}$ with $\mathcal{D}^t$

Ergo, the set of elements inserted into $\mathcal{D}^t[x]$ consists of all $y \in \mathcal{V}$ such that $d(x,y) < t$, which is to say that $\mathcal{D}^t[x]$ directly approximates $\mathcal{C}_t^{\text{NBHD}}(x)$ (Eq. (**??**)). Ergo, the summations over all sketches in lines **??** and **??** of Algorithm **??** estimate $\mathcal{N}(t)$ in the same way as does the equivalent procedure in HyperANF. Note that these summations are performed as distributed Reduce operations. In addition to returning estimates of the neighborhood function, this procedure can be used to return estimates of the local t-degree neighborhoods of vertices $\mathcal{C}_t^{\text{NBHD}}(\cdot)$, which have their own uses. Furthermore, as the actual estimates produced by Algorithm **??** as the same as those produced by HyperANF, all of the statistical results of [**?**] apply. We reproduce the following theorem, although the other theorems and corollaries of [**?**] also apply.

Theorem 10.1. *Let $\mu_{r,n}$ and $\eta_{r,n}$ be the multiplicative bias and standard deviation for* HLL*s given in Theorem 1 of [**?**]. The output $\widetilde{\mathcal{N}}(t)$ and $\widetilde{\mathcal{C}}_t^{\text{NBHD}}(x)$ for $x \in \mathcal{V}$ at the t-th iteration satisfies*

$$\frac{\mathbb{E}\left[\widetilde{\mathcal{N}}(t)\right]}{\mathcal{N}(t)} = \frac{\mathbb{E}\left[\widetilde{\mathcal{C}}_t^{\text{NBHD}}(x)\right]}{\mathcal{C}_t^{\text{NBHD}}(x)} = \mu_{r,n} \text{ for } n \to \infty,$$

i.e. they are nearly unbiased.

Furthermore, both also have standard deviation bounded by $\eta_{r,n}$. That is,

$$\frac{\sqrt{\text{Var}\left[\widetilde{\mathcal{N}}(t)\right]}}{\mathcal{N}(t)} \leq \eta_{r,n} \text{ and } \frac{\sqrt{\text{Var}\left[\widetilde{\mathcal{C}}_t^{\text{NBHD}}(x)\right]}}{\mathcal{C}_t^{\text{NBHD}}(x)} \leq \eta_{r,n}$$

Proof. For each x, $\widetilde{\mathcal{C}}_t^{\text{NBHD}}(x) = |\mathcal{D}^k[x]|$, where $\mathcal{D}^k[x]$ is a union of HLLs, into which every y such that $d(x,y) < t$ is inserted, as we noted from Eq. (**??**). Thus by Theorem 1 of [**?**],

$$\mathbb{E}\left[\widetilde{\mathcal{C}}_t^{\text{NBHD}}(x)\right] = \mu_{r,n}\mathcal{C}_t^{\text{NBHD}}(x)$$
$$\sqrt{\text{Var}\left[\widetilde{\mathcal{C}}_t^{\text{NBHD}}(x)\right]} = \eta_{r,n}\mathcal{C}_t^{\text{NBHD}}(x).$$

Thus, by the linearity of expectation and the subadditivity of variance,

$$\mathbb{E}\left[\widetilde{N}(t)\right] = \sum_{x\in\mathcal{V}} \mathbb{E}\left[\widetilde{\mathcal{C}}_t^{\text{NBHD}}(x)\right] = \mu_{r,n}\sum_{x\in\mathcal{V}} \mathcal{C}_t^{\text{NBHD}}(x) = \mu_{r,n}\mathcal{N}(t), \text{ and}$$
$$\sqrt{\text{Var}\left[\widetilde{N}(t)\right]} \leq \sum_{x\in\mathcal{V}} \sqrt{\text{Var}\left[\widetilde{\mathcal{C}}_t^{\text{NBHD}}(x)\right]} \leq \eta_{r,n}\sum_{x\in\mathcal{V}} \mathcal{C}_t^{\text{NBHD}}(x) = \eta_{r,n}\mathcal{N}(t).$$

□

Unlike HyperANF, Algorithm **??** is distributed and streaming, and does not require the storage of $\mathcal{G}$ in memory. However, it is also not optimized to take advantage of shared memory task decomposition or multicore optimizations using *broadword programming* like HyperANF. It is possible to design an algorithm that supports these features using a hybrid shared-distributed memory architecture. We will not produce it in detail. We will instead describe it at a high level. Rather than each core on each node operating independently, each node would act as an instance of HyperANF on a partition of the graph, communicating sketches to other nodes as needed like Algorithm **??**. Merges and estimates would make use of all of the shared memory cores, utilizing the broadword programming approach given in Section 3 of [**?**].

11 Edge-Local Triangle Count Heavy Hitters

In addition to estimating local neighborhood sizes, DegreeSketch affords an analysis of local triangle counts using intersection estimation. Furthermore, while sampling-based streaming algorithms are limited to vertex-local triangle counts, DegreeSketch affords the analysis of edge-local triangle counts. Edge-local triangle counts, i.e. the number of triangles in which each edge participate, can be thought of as a

generalization of vertex-local triangle counts. Given the edge-local triangle counts for each edge incident upon a vertex, we can easily compute its vertex-local triangle count. Specifically,

$$\mathcal{C}^{\text{TRI}}(x) = \frac{1}{2} \sum_{xy \in \mathcal{E}} \mathcal{C}^{\text{TRI}}(xy) \tag{18}$$

The reverse is not true.

Edge-local triangle counts have understandably not received much attention in the streaming literature, considering that even enumerating them requires $\Omega(m)$ space. Given an accumulated DegreeSketch $\mathcal{D}$ and some approximate intersection operator $\widetilde{\cap}$, for $xy \in \mathcal{E}$ we can estimate $\mathcal{C}^{\text{TRI}}(xy)$ using

$$\widetilde{\mathcal{C}}^{\text{TRI}}(xy) = \mathcal{D}[x] \widetilde{\cap} \mathcal{D}[y]. \tag{19}$$

This procedure is similar to the well-known intersection method for local triangle counting. Indeed, we can estimate the total number of triangles $\mathcal{T}$ in the graph by computing

$$\widetilde{\mathcal{T}} = \frac{1}{3} \sum_{xy \in \mathcal{E}} \widetilde{\mathcal{C}}^{\text{TRI}}(xy) = \frac{1}{3} \sum_{xy \in \mathcal{E}} \mathcal{D}[x] \widetilde{\cap} \mathcal{D}[y]. \tag{20}$$

Unfortunately, while most cardinality sketches have a native and closed $\widetilde{\cup}$ operation, they all lack a satisfactory intersection operation. This is not surprising, as sketches are in effect lossy compressions. Indeed, it is known that the detection of a trivial intersection is impossible in sublinear memory. Hence, we must instead make use of unsatisfactory intersection operations in practice, which has been a focus of recent research [**?**, **?**, **?**]. We will discuss these in more detail in Section **??**, and their shortcomings in Section **??**. For our purposes, we will suppose that $\widetilde{\cap}$ is reliable only where intersections are large. Consequently, we will attempt only to recovery the heavy hitters of $\mathcal{C}^{Tri}$.

Algorithm **??** provides a chassis for Algorithms **??** and **??**, which differ only in their communication behavior. In Algorithm **??**, all processors read over their edge streams and forward edges to one of their endpoints, similar to the behavior in the **Accumulation Context** of Algorithm **??**. They also initialize a counter $\widetilde{\mathcal{T}}$ and a min heap with a maximum size of k, $\widetilde{\mathcal{H}}_k$. These values are modified in the send and receive contexts of Algorithms **??** and **??**.

Algorithm 3 Local Triangle Count Heavy Hitters Chassis

Input: $\boldsymbol{\sigma}$ - edge stream divided into $|\mathcal{P}|$ substreams
k - integral heavy hitter count
$\mathcal{P}$ - universe of processors
$\mathcal{D}$ - accumulated DegreeSketch
$\mathcal{S}$ - distributed dictionary mapping $\mathcal{P}$ to send queues
$\mathcal{R}$ - distributed dictionary mapping $\mathcal{P}$ to receive queues
f - function mapping $\mathcal{V} \to \mathcal{P}$

Accumulation Context for $P \in \mathcal{P}$:
1: $\widetilde{\mathcal{H}}_k \leftarrow$ empty k-heap
2: $\widetilde{\mathcal{T}} \leftarrow 0$
3: **while** σ_P has unread element uv **do**
4: $\quad U \leftarrow f(u)$
5: $\quad \mathcal{S}[P]$.push $(\text{Edge}, (U, uv))$
6: Reduce $\widetilde{\mathcal{T}}$
7: $\widetilde{\mathcal{T}} \leftarrow \frac{1}{3}\widetilde{\mathcal{T}}$

Algorithm **??** issues a chain of messages for each read edge, not unlike the procedure in Algorithm **??**. P reads uv, and issues a message of type Edge containing uv to $U = f(u)$. Upon receipt, U issues a message of type Sketch containing $(\mathcal{D}[u], uv)$ to $V = f(v)$. When V receives this message, it computes $\widetilde{\mathcal{C}}^{\text{TRI}}(uv)$

via Eq. (**??**) and updates $\widetilde{\mathcal{T}}$ and $\widetilde{\mathcal{H}}_k$. Once computation is complete and all receive queues are flushed, the algorithm computes a global REDUCE sum to find $\widetilde{\mathcal{T}}$ and similarly finds the global top k estimates via a reduce on $\widetilde{\mathcal{H}}_k$. The algorithm returns $\widetilde{\mathcal{T}}/3$ (each triangle is counted 3 times) and $\widetilde{\mathcal{H}}_k$.

Algorithm 4 DEGREESKETCH Edge-Local Triangle Count Heavy Hitters

Output: $\widetilde{\mathcal{T}}$, $\widetilde{\mathcal{C}}^{\text{TRI}}(xy)$ for top k edges xy

Send Context for $P \in \mathcal{P}$:
while $\mathcal{S}[P]$ is not empty **do**
  if next message is an EDGE **then**
    $(W, xy) \leftarrow \mathcal{S}[P]$.pop()
    $\mathcal{R}[W]$.push (EDGE, xy)
  else if next message is a SKETCH **then**
    $(W, \mathcal{D}[x], y) \leftarrow \mathcal{S}[P].pop()$
    $\mathcal{R}[W]$.push (SKETCH, xy)
Receive Context for $P \in \mathcal{P}$:
while $\mathcal{R}[P]$ is not empty **do**
  if next message is an EDGE **then**
    $xy \leftarrow \mathcal{R}[P]$.pop()
    $Y \leftarrow f(y)$
    $\mathcal{S}[P]$.push (SKETCH, $(Y, \mathcal{D}[x], xy)$)
  else if next message is a SKETCH **then**
    $(\mathcal{D}[x], xy) \leftarrow \mathcal{R}[P]$.pop()
    $\widetilde{\mathcal{C}}^{\text{TRI}}(xy) \leftarrow$ ESTIMATEINTERSECTION$(\mathcal{D}[y], \mathcal{D}[x])$
    $\widetilde{\mathcal{T}} \leftarrow \widetilde{\mathcal{T}} + \widetilde{\mathcal{C}}^{\text{TRI}}(xy)$
    if $\widetilde{\mathcal{C}}^{\text{TRI}}(xy) > \min \mathcal{H}_k$ **then**
      insert $\left(xy, \widetilde{\mathcal{C}}^{\text{TRI}}(xy)\right)$ into $\mathcal{H}_k$
      if $|\mathcal{H}_k| > k$ **then**
        remove $\min \mathcal{H}_k$
Execution for $P \in \mathcal{P}$:
Run Algorithm **??** using these communication contexts
REDUCE $\widetilde{\mathcal{H}}_k$
return $\widetilde{\mathcal{T}}, \widetilde{\mathcal{H}}_k$

Algorithm **??** addresses edge–local triangle count heavy hitter recovery using memory sublinear in the size of $\mathcal{G}$. It requires $\widetilde{O}(\varepsilon^{-2}m)$ time and communication, given our assumptions, and a total of $O(\varepsilon^{-2}|\mathcal{V}| \log\log|\mathcal{V}| + \log|\mathcal{V}|)$ space, where DegreeSketch is implemented using HYPERLOGLOG sketches with accuracy parameter ε. Unfortunately, we are unable to provide an analytic bound on the error of this algorithm, due to the nature of sublinear intersection estimation. We will explore this problem in Section **??** and provide experimental analysis of Algorithm **??**.

12 Vertex-Local Triangle Count Heavy Hitters

Given access to a trained DEGREESKETCH $\mathcal{D}$ and $A_{:,x}$, we can compute an estimate of $\mathcal{C}^{\text{TRI}}(x)$ using

$$\widetilde{\mathcal{C}}^{\text{TRI}}(x) = \frac{1}{2} \sum_{y : A_{y,x} \neq 0} \widetilde{\mathcal{C}}^{\text{TRI}}(xy) = \frac{1}{2} \sum_{y : A_{y,x} \neq 0} \mathcal{D}[x] \widetilde{\cap} \mathcal{D}[y]. \tag{21}$$

While we are not faced with the space constraint present in Section **??** when contending with simply writing down vertex-local triangle counts, we instead must contend with the dreaded small intersection

problem discussed in Section **??**. Consequently, we limit our scope to the recovery of vertex-local triangle count heavy hitters.

Algorithm **??** performs vertex-local triangle count estimation in a manner similar to Algorithm **??** with some additional steps. We maintain $\widetilde{\mathcal{C}}^{\text{TRI}}(x)$ for each $x \in \mathcal{V}$, which are of course distributed so that $X = f(x)$ computes $\widetilde{\mathcal{C}}^{\text{TRI}}(x)$. It performs similar work for $xy \in \mathcal{E}$ up to the point processor $Y = f(y)$ estimates $\widetilde{\mathcal{C}}^{\text{TRI}}(xy)$. Instead of inserting this estimate into a local max heap, we add it to $\widetilde{\mathcal{C}}^{\text{TRI}}(y)$, and forward $(\widetilde{\mathcal{C}}^{\text{TRI}}(xy), x)$ to $X = f(x)$ so that it can add it to $\widetilde{\mathcal{C}}^{\text{TRI}}(x)$. This message has the EST type, to distinguish it from EDGE and SKETCH messages.

Algorithm **??** addresses vertex–local triangle count heavy hitter recovery using the same asymptotic computation, memory and communication costs as Algorithm **??**. Unfortunately, we are similarly unable to provide an a priori analytic bound on the error of this algorithm. We do, however, have the following theorem using the subadditivity of the standard deviation (i.e. If A and B have finite variance, $\sqrt{\text{Var}\left[A+B\right]} \leq \sqrt{\text{Var}\left[A\right]} + \sqrt{\text{Var}\left[B\right]}$).

Theorem 12.1. *Let $\widetilde{\mathcal{C}}^{\text{TRI}}(x)$ be the estimated output of Algorithm **??** for $x \in \mathcal{V}$, and that $\widetilde{\mathcal{C}}^{\text{TRI}}(xy)$ is the estimated edge triangle count for each $xy \in \mathcal{E}$. Assume further that for each xy, we know a standard deviation bound η_{xy} so that*

$$\frac{\sqrt{\text{Var}\left[\widetilde{\mathcal{C}}^{\text{TRI}}(xy)\right]}}{\mathcal{C}^{\text{TRI}}(xy)} \leq \eta_{xy}. \tag{22}$$

Furhermore, let $\eta_* = \max_{xy\in\mathcal{E}} \eta_{xy}$. Then, $\widetilde{\mathcal{C}}^{\text{TRI}}(x)$ has at most twice this maximum standard deviation. That is,

$$\frac{\sqrt{\text{Var}\left[\widetilde{\mathcal{C}}^{\text{TRI}}(x)\right]}}{\mathcal{C}^{\text{TRI}}(x)} \leq 2\eta_*.$$

Proof.

$$\begin{aligned}
\frac{\sqrt{\text{Var}\left[\widetilde{\mathcal{C}}^{\text{TRI}}(x)\right]}}{\mathcal{C}^{\text{TRI}}(x)} &= \frac{\sqrt{\text{Var}\left[\sum_{xy\in\mathcal{E}} \widetilde{\mathcal{C}}^{\text{TRI}}(xy)\right]}}{\mathcal{C}^{\text{TRI}}(x)} \\
&\leq \frac{\sum_{xy\in\mathcal{E}} \sqrt{\text{Var}\left[\widetilde{\mathcal{C}}^{\text{TRI}}(xy)\right]}}{\mathcal{C}^{\text{TRI}}(x)} && \text{subadditivity} \\
&\leq \frac{\sum_{xy\in\mathcal{E}} \eta_{xy}\mathcal{C}^{\text{TRI}}(xy)}{\mathcal{C}^{\text{TRI}}(x)} && \text{Eq. (??)} \\
&\leq \frac{\eta_* \sum_{xy\in\mathcal{E}} \mathcal{C}^{\text{TRI}}(xy)}{\mathcal{C}^{\text{TRI}}(x)} \\
&= 2\eta_* && \text{Eq. (??)}
\end{aligned}$$

□

Theorem **??** shows that if we can bound the standard deviation of the edge-local triangle count estimates produced using DEGREESKETCH, we can also bound the standard deviation of the vertex-local triangle count estimates produced by Algorithm **??**. Unfortunately, we are unable to provide these bounds a priori, as they depend upon the sizes of all of the the sets and their intersections, which are unknown. It does, however, show how the variance of the vertex-local estimates depends upon those of the edge-local estimates.

Algorithm 5 DegreeSketch Vertex-Local Triangle Count Heavy Hitters

Output: $\widetilde{\mathcal{T}}$, $\widetilde{\mathcal{C}}^{\text{Tri}}(x)$ for top k vertices x

Send Context for $P \in \mathcal{P}$:
while $\mathcal{S}[P]$ is not empty **do**
  if next message is an Edge **then**
    $(W, xy) \leftarrow \mathcal{S}[P]$.pop()
    $\mathcal{R}[W]$.push (Edge, xy)
  else if next message is a Sketch **then**
    $(W, \mathcal{D}[x], xy) \leftarrow \mathcal{S}[P].pop()$
    $\mathcal{R}[W]$.push (Sketch, xy)
  else if next message is an Est **then**
    $(Y, \widetilde{\mathcal{C}}^{\text{Tri}}(xy), y) \leftarrow \mathcal{S}[P].pop()$
    $\mathcal{R}[T]$.push $\left(\text{Est}, (\widetilde{\mathcal{C}}^{\text{Tri}}(xy), y)\right)$

Receive Context for $P \in \mathcal{P}$:
while $\mathcal{R}[P]$ is not empty **do**
  if next message is an Edge **then**
    $xy \leftarrow \mathcal{R}[P]$.pop()
    $Y \leftarrow f(y)$
    $\mathcal{S}[P]$.push $(\text{Sketch}, (Y, \mathcal{D}[x], xy))$
  else if next message is a Sketch **then**
    $(\mathcal{D}[x], xy) \leftarrow \mathcal{R}[P]$.pop()
    $Y \leftarrow f(y)$
    $\widetilde{\mathcal{C}}^{\text{Tri}}(xy) \leftarrow \text{EstimateIntersection}(\mathcal{D}[y], \mathcal{D}[x])$
    $\widetilde{\mathcal{C}}^{\text{Tri}}(x) \leftarrow \widetilde{\mathcal{C}}^{\text{Tri}}(x) + \widetilde{\mathcal{C}}^{\text{Tri}}(xy)$
    $\widetilde{\mathcal{T}} \leftarrow \widetilde{\mathcal{T}} + \widetilde{\mathcal{C}}^{\text{Tri}}(xy)$
    $\mathcal{S}[P]$.push $\left(Y, (\widetilde{\mathcal{C}}^{\text{Tri}}(xy), y)\right)$
  else if next message is an Est **then**
    $(\widetilde{\mathcal{C}}^{\text{Tri}}(xy), y) \leftarrow \mathcal{R}[P]$
    $\widetilde{\mathcal{C}}^{\text{Tri}}(y) \leftarrow \widetilde{\mathcal{C}}^{\text{Tri}}(y) + \widetilde{\mathcal{C}}^{\text{Tri}}(xy)$

Execution for $P \in \mathcal{P}$:
$\widetilde{\mathcal{C}}^{\text{Tri}}(x) \leftarrow 0$ for each $x \in f(P)$
Run Algorithm **??** using these communication contexts
for $x \in f(P)$ **do**
  if $\widetilde{\mathcal{C}}^{\text{Tri}}(x) > \min \widetilde{\mathcal{H}}_k$ **then**
    insert $\left(x, \widetilde{\mathcal{C}}^{\text{Tri}}(x)\right)$ into $\widetilde{\mathcal{H}}_k$
    if $|\widetilde{\mathcal{H}}_k| > k$ **then**
      remove $\min \widetilde{\mathcal{H}}_k$
Reduce $\widetilde{\mathcal{H}}_k$
return $\widetilde{\mathcal{T}}, \widetilde{\mathcal{H}}_k$

13 HyperLogLog Cardinality Sketches

While many different cardinality sketches have been proposed, the HyperLogLog sketch is undoubtedly the most popular of these data structures in practice, and has attained widespread adoption [**?**]. The sketch relies on the key insight that the binary representation of a random machine word starts with $0^{j-1}1$ with probability 2^{-j}. Thus, if the maximum number of leading zeros in a set of random words is $j-1$, then 2^j is a good estimate of the cardinality of the set [**?**]. However, this estimator clearly has high variance. The variance is traditionally minimized using stochastic averaging to simulate parallel random trials [**?**].

Assume we have a stream σ of random machine words of a fixed size W. For a $W = (p+q)$-bit word w, let $\xi(w)$ be the first p bits of w, and let $\rho(w)$ be the number of leading zeros plus one of its remaining q bits. We pseudorandomly partition elements e of σ into $r = 2^p$ substreams of the form $\sigma_i = \{e \in \sigma | \xi(e) = i\}$. For each of these approximately equally-sized streams, we maintain an independent estimator of the above form. Each register $\mathbf{r}_i$, $i \in [m]$, accumulates the value

$$\mathbf{r}_i = \max_{x \in \sigma_i} \rho(x). \tag{23}$$

After accumulation, $\mathbf{r}_i$ stores the maximum number of leading zeroes in the substream σ_i, plus one. The authors of HyperLogLog show in [**?**] that the normalized bias corrected harmonic mean of these registers,

$$\widetilde{D} = \alpha_r r^2 \left(\sum_{i=0}^{r-1} 2^{-\mathbf{r}_i} \right)^{-1}, \tag{24}$$

where the bias correction term α_r is given by

$$\alpha_r := \left(r \int_0^\infty \left(\log_2 \left(\frac{2+u}{1+u} \right) \right)^r du \right)^{-1}, \tag{25}$$

is a good estimator of the number of unique elements in σ. If the true cardinality of the streamed multiset is D, the error of estimate $\widetilde{D}$, $|D - \widetilde{D}|$, has standard error $\approx 1.04/\sqrt{r}$. $\widetilde{D}$ satisfies

$$|D - \widetilde{D}| \leq (1.04/\sqrt{r})D. \tag{26}$$

with high probability. A comprehensive analysis of the properties of the HLL sketch, the quality of (**??**), and a proof of (**??**) can be found in [**?**]. In particular, if N is the number of possible machine words, a HyperLogLog sketch satisfied Eq. (**??**) using space $O(\varepsilon^{-2} \log \log N + \log N)$, where $r = \Theta(\varepsilon^2)$.

Of course, practical streams do not consist of random 64-bit numbers. In practice, we simulate this randomness by way of hash functions, of which there are many alternatives. The fast, non-cryptographic hash functions Murmurhash3 [**?**] and xxhash [**?**] are often utilized in implementations. We will assume throughout that algorithms have access to such a hash function $h : 2^{64} \to 2^{64}$.

A particular HyperLogLog sketch, S, consists of such a hash function h, a prefix size p (typically between 4 and 16), a maximum register value q, and an array of $r = 2^p$ registers, $\mathbf{r}$, all of which are initialized to zero. We summarize references to such a sketch as HLL(p, q, h). Algorithm **??** describes the accumulation and functions supported by the vanilla HyperLogLog sketch. We will add features in the next few sections.

Note that HLLs support a natural merge operation: taking the element-wise maximum of each index of a pair register vectors. This requires that the two sketches were generated using the same hash function. We will assume that all sketches share a hash function throughout the rest of the chapter.

13.1 Sparse Register Format

Heule et al. suggest a sparse representation for HyperLogLog sketches consisting of a list of the set index-value pairs of a HLL's register list [**?**]. Mathematically, the sparsification procedure is tantamount to maintaining the set $R = \{(i, \mathbf{r}_i) | \mathbf{r}_i \neq 0\}$. R requires less memory than $\mathbf{r}$ when the cardinality of the underlying multiset is small. Moreover, it is straightforward to saturate a sparse sketch into a dense one once it is no longer cost effective to maintain it by instantiating $\mathbf{r}$ while assuming all registers not set in R are zero. We will assume that R is implemented as a map, where an element $R[j] = z$ if $(j, z) \in R$ and

Algorithm 6 HLL(p, q, h) Operations

State Variables for HLL(p, q, h) S :

- p integral prefix size
- q integral maximum register value
- h hash function mapping universe $U \to [0, 2^{64} - 1]$
- $r \coloneqq 2^p$
- $\mathbf{r}$ array of m registers, initially all zeroes

Accumulation:

1: $S \leftarrow$ empty HLL(p, q, h)
2: **for** $e \in \sigma$ **do**
3: $\quad x \leftarrow h(e)$
4: $\quad$ INSERT$(S, \xi(x), \rho(x))$

Functions:

5: **function** INSERT(S, j, z)
6: $\quad \mathbf{r}_j \leftarrow \max(\mathbf{r}_j, z)$
7: **function** MERGE$(S^{(0)}, S^{(1)}, \ldots, S^{(\ell)})$
8: $\quad S^* \leftarrow$ empty HLL(p, q, h)
9: $\quad$ **for** $j \in [0, r)$ **do**
10: $\quad\quad \mathbf{r}^*_j \leftarrow \max_{i \in [0,\ell]} \mathbf{r}^{(i)}_j$
11: $\quad$ **return** S^*
12: **function** ESTIMATE(S)
13: $\quad$ **return** $\alpha_r r^2 \left(\sum_{j=0}^{r-1} 2^{-\mathbf{r}_j} \right)^{-1}$

Algorithm 7 HLL(p, q, h) Operations Update - sparsification

State Variables for HLL(p, q, h) S :
- ν mode $\in \{\text{SPARSE}, \text{DENSE}\}$, initially SPARSE
- R sparse register set, initially $\emptyset$

1: **function** INSERT(S, j, z)
2: **if** $\nu =$ DENSE **then**
3: $\mathbf{r}_j \leftarrow \max(\mathbf{r}_j, z)$
4: **else if** $\nu =$ SPARSE **then**
5: $R[j] \leftarrow \max\{z, R[j]\}$ (see Figures 6 & 7 of [**?**])
6: **if** $|R| > 6 * r$ **then**
7: SATURATE(S)
8: **function** SATURATE(S)
9: $\nu \leftarrow$ DENSE
10: **for** $(j, z) \in R$ **do**
11: INSERT(S, j, z)
12: $R \leftarrow \emptyset$
13: **function** MERGE$(S^{(0)}, S^{(1)}, \ldots, S^{(\ell-1)})$
14: $S^* \leftarrow$ empty HLL(p, q, h)
15: **for** $j \in [0, r)$ **do**
16: $z \leftarrow \max_{i \in [0,\ell)} \left(\max\left\{\mathbf{r}_j^{(i)}, R^{(i)}[j]\right\}\right)$
17: **if** $z \neq 0$ **then**
18: INSERT(S^*, j, z)
19: **return** S^*
20: **function** ESTIMATE(S)
21: **if** $\nu =$ DENSE **then**
22: **return** $\alpha_r r^2 \left(\sum_{j=0}^{r-1} 2^{-\mathbf{r}_j}\right)^{-1}$
23: **else**
24: **return** $\alpha_r r^2 \left(\sum_{(j,z)\in R} 2^{-z}\right)^{-1}$

is zero otherwise. Algorithm **??** describes the changes and additions to Algorithm **??** needed to implement sparse registers.

We use sparse registers in our algorithms, although we will not go into the gory details of their maintenance in the interest of parsimony. We instead describe the procedure at a high level. The sparse sketch representation R can be implemented efficiently by maintaining a list sorted by register index, containing at most one entry per register, and a set of unsorted pairs that is periodically folded into the sorted list. In a practical implementation, these pairs are encoded into a single value that must be decoded when read. We obfuscate the details of the efficient implementation in our algorithms, and invite the interested reader to investigate Figures 6 & 7 of [**?**].

The authors of [**?**] also describe a procedure by which the sparse elements of R might be stored at higher precision than the elements of $\mathbf{r}$. Although this method may be practical for applications we will avoid it in our work to avoid unnecessary confusion.

13.2 Reduced Register Size

In practice, $N = 2^{64}$, and so the registers require 6 bits apiece. This small register size is one of the advantages of HLLs over other cardinality sketches. For example, MinCount requires the storage of the same number of 64-bit hashes for the same universe size. Cardinality sketches are known to require $\Omega(\varepsilon^{-2} + \log N)$ space, where $2^p = m = O(\varepsilon^{-2})$, although the known optimal algorithm is not considered practical [**?**]. This reliance upon $\Omega(\varepsilon^{-2})$ registers implies that HLLs are about as optimal as we can get in implementations without sacrificing performance.

The authors of HyperLogLog-TailCut reduced the footprint of the HyperLogLog algorithm by reducing the registers to 4 bits [**?**]. In order to avoid overflow, HyperLogLog-TailCut adds a base register b, initialized to zero, and changes the update rule (**??**) and estimator (**??**) so that each register $\mathbf{r}_j$ stores notional value $\mathbf{r}_j + b$. When $\mathbf{r}_j$ would experience an overflow event, HyperLogLog-TailCut instead increases b to the minimum set register value and decreases all registers by the same quantity. This yields an insert rule given by Algorithm **??**.

Algorithm 8 HyperLogLog-TailCut Insert

```
e ← next element of σ
x ← h(e)
if ρ(x) − b > 15 then
    Δb ← min_{i∈[0,r)} r_i
    if Δb > 0 then
        b ← b + Δb
        for j ∈ [0, r) do
            r_j ← r_j − Δb
r_{ξ(x)} ← max(r_{ξ(x)}, min(ρ(x) − b, 15))
```

Note that this modification to the register update procedure is lossy. If $\rho(x) - b > 15$ even after updating b, then the algorithm notionally stores $b + 15$ in $\mathbf{r}_{\xi(x)}$. We henceforward refer to this event as a "tail cut". The authors show that tail cuts occur infrequently enough that it does not impact the Monte Carlo $1.04/\sqrt{m}$ error bound using the following modification to (**??**):

$$\tilde{D} = \alpha_r r^2 \left(\sum_{i=0}^{r-1} 2^{-(b+\mathbf{r}_i)} \right)^{-1}. \tag{27}$$

However, this tail-cutting introduces bias when performing merge operations on sketches. Concatenations of two streams may not yield the same answer as merging their individual sketches, which violates Eq. (**??**). Furthermore, tail-cutting is order dependent, so sketches accumulated over the same reordered stream may not agree, which violates a core assumption one usually makes about sketches.

The validity of the estimator based upon the union of sketches depends upon the property that the element-wise maximum of a set of sketches over streams is identical to the sketch accumulated from their

concatentation. The tail-cutting procedure introduced in Algorithm **??** violates this property, as some hashes can be cut in the operands that would not be cut in the sketch over the concatenated stream. For a small number of sketches, the resulting error is small enough that it might go without notice. However, when merging many sketches, the additional error introduced by the tail cuts results in an estimator that does not maintain the desired error bound property (**??**).

We solve this problem by maintaining a set E of these cut elements. This set is of the same (index, value) form as the set of sparse registers R discussed above, and in practice is small as tail cutting events are uncommon. Where b is set, the probability of generating a hash x that causes an overflow is given by

$$\Pr[\rho(x) - b > 15] = 2^{-15-b} \tag{28}$$

using an idealized hash function. The probability that this hash gets cut is more difficult to characterize, and depends on the elements read thus far. We found approximately 4 tail cut events per 10^9 distinct element insertions in our experiments.

We add a subroutine that attempts to reinsert the cut elements into the registers. This subroutine gets called whenever the base register increases, the estimate procedure is called, the sketch is merged with another, or the set reaches a given size bound. These changes result in the updated INSERT, MERGE, and ESTIMATE procedures given in Algorithm **??**. Note that these procedures are able to coexist with the sparse register format, allowing us to combine the two approaches.

Algorithm **??** guarantees order-invariance in the produced sketches, as well as maintaining a true sketch merge. These results come at the cost of some additional space, as the cut set E must be stored. An adversarial stream could result in E could hold as many as $r - 1$ elements, i.e. there is one holdout register index that prevents b from growing in spite of cut insertions. Fortunately, such an event is vanishingly unlikely when using a reasonable hash function. Furthermore, even this worst case at most doubles the size of the sketch, as so maintains the same asymptotic bounds as the vanilla HLL.

13.3 Maximum Likelihood Estimation

The estimator (**??**) is known to have several practical problems, many of which are discussed at length in [**?**] and [**?**]. Subsequent work has refined HyperLogLog by modifying the estimator (**??**) to reduce bias on high and low values [**?**, **?**, **?**], reducing the register size by a constant [**?**], and replacing the estimator (**??**) entirely with a maximum likelihood estimator [**?**, **?**, **?**]. We adopt the latter of these approaches, which yields the added benefit of a maximum likelihood estimator for the intersection of two sketches. We will sketch the ideas for the estimators here, although the full treatment is somewhat involved. We direct the interested reader to [**?**] for details.

The maximum likelihood estimator in [**?**] uses a Poisson model, assuming that the cardinality itself is drawn from a Poisson distribution with parameter λ, and that the observed register values after accumulation are independent. This yields the following loglikelihood function for λ given the observed register list $\mathbf{r}$:

$$\mathcal{L}(\lambda \mid \mathbf{r}) = -\frac{\lambda}{r}\sum_{k=0}^{q}\frac{\mathbf{c}_k}{2^k} + \sum_{k=1}^{q}\mathbf{c}_k \log\left(1 - e^{-\frac{\lambda}{r2^k}}\right) + \mathbf{c}_{q+1}\log\left(1 - e^{-\frac{\lambda}{r2^q}}\right). \tag{29}$$

Here

$$\mathbf{c}_k = |\{\mathbf{r}_i = k \mid i \in \{0, \ldots, p-1\}\}| \tag{30}$$

is the count of occurrences of the value k in the register list $\mathbf{r}$ for $k \in \{0, 1, \ldots, q+1\}$. The author shows in [**?**] that given an unbiased estimator $\hat{\lambda}$ for λ, we can leverage depoissonization [**?**] to yield an estimator for the fixed-size set. That is, $\mathbb{E}[\hat{\lambda} \mid D] = D$, where D is the cardinality of the input set. The count statistic $\mathbf{c}$ suffices to iteratively find the optimum of (**??**), yielding a maximum likelihood estimator. See Algorithm 8 of [**?**] for the full algorithm description.

13.4 Intersection Estimation

A naïve approach to estimating an intersection of two sets A and B using cardinality sketches might involve computing the intersection via the inclusion-exclusion principle:

$$|A \cap B| = |A \cup B| - |A| - |B|. \tag{31}$$

Algorithm 9 HLL(p, q, h) Operations - sparsification + tail cut

State Variables for HLL(p, q, h) S :
- b base register, initially 0
- E cut set, initially $\emptyset$

Functions:

function INSERT(S, j, z)
  if $\nu =$ DENSE **then**
    $\mathbf{r}_j \leftarrow \max(\mathbf{r}_j, z)$
    if $z - b > 15$ **then**
      $\Delta b \leftarrow \min_{i \in [0,r)} \mathbf{r}_i$
      if $\Delta b > 0$ **then**
        $b \leftarrow b + \Delta b$
        for $i \in [0, r)$ **do** $\mathbf{r}_i \leftarrow \mathbf{r}_i - \Delta b$
        FLUSHCUTS(S)
    $\mathbf{r}_j \leftarrow \max(\mathbf{r}_j, \min(z - b, 15))$
    if $z - b > 15$ **then**
      $E[j] \leftarrow \max\{z, E[j]\}$
  else if $\nu =$ SPARSE **then**
    $R[j] \leftarrow \max\{z, R[j]\}$ (see Figures 6 & 7 of [**?**])
    if $|R| > 4 * r$ **then**
      SATURATE(S)
function FLUSHCUTS(S)
  for $(j, z) \in E$ **do**
    $E \leftarrow E \setminus \{(j, z)\}$
    INSERT(j, z)
function MERGE$(S^{(0)}, S^{(1)}, \ldots, S^{(\ell)})$
  $S^* \leftarrow$ empty HLL(p, q, h)
  for $j \in [0, r)$ **do**
    $b^* \leftarrow \max_{i \in [0,\ell]} b^{(i)}$
    $z \leftarrow \max_{i \in [0,\ell]} \left(\max \left\{ \mathbf{r}_j^{(i)} + b^{(i)} - b^*, E^{(i)}[j], R^{(i)}[j] \right\} \right)$
    if $z \neq 0$ **then**
      INSERT(S^*, j, z)
  return S^*
function ESTIMATE(S)
  if $\nu =$ DENSE **then**
    return $\alpha_r r^2 \left(\sum_{j=0}^{r-1} 2^{-\max\{\mathbf{r}_j + b, E[j]\}} \right)^{-1}$
  else
    return $\alpha_r r^2 \left(\sum_{(j,z) \in R} 2^{-z} \right)^{-1}$

Given sketches $S^{(A)}$ and $S^{(B)}$ for A and B, we might be tempted to estimate the intersection via

$$\widetilde{|A \cap B|} = \textsc{Estimate}\left(S^{(A)}\right) + \textsc{Estimate}\left(S^{(B)}\right) - \textsc{Estimate}\left(\textsc{Merge}\left(S^{(A)}, S^{(B)}\right)\right). \tag{32}$$

However, the approach in Eq. (**??**) suffers from several failings. In particular, it might be negative! Furthermore, due to the error noise in each estimate, if the true intersection is small relative to the set sizes, or if one set is much larger than the other, the variance of Eq. (**??**) will be quite high.

We describe a better intersection estimator due to Ertl [**?**]. This estimator is similar to the maximum likelihood estimator Eq. (**??**), instead focusing on the joint distribution of a pair of sketched sets A and B. Accordingly, the estimator yields estimates of $|A \setminus B|$, $|B \setminus A|$, and $|A \cap B|$. The algorithm depends on a similar optimization of a Poisson model application, where it is assumed that $|A \setminus B|$ is drawn from a Poisson distribution with parameter λ_a, and similarly $|B \setminus A|$ and $|A \cap B|$ use Poisson parameters λ_b and λ_x. These parameters can be related to the observed HyperLogLog register lists corresonding to A and B, $\mathbf{r}^{(A)}$ and $\mathbf{r}^{(B)}$, via a loglikelihood function $\mathcal{L}(\lambda_a, \lambda_b, \lambda_x \mid \mathbf{r}^{(A)}, \mathbf{r}^{(B)})$. This is Eq. (70) in [**?**], which we reproduce in Eq. (**??**) in this work.

$$\begin{aligned}
&\mathcal{L}(\lambda_a,\lambda_b,\lambda_x \mid \mathbf{r}^{(A)}, \mathbf{r}^{(B)}) = \\
&\quad \sum_{k=1}^{q} \log\left(1 - e^{-\frac{\lambda_a+\lambda_x}{r2^k}}\right)\mathbf{c}_k^{(A),<} + \log\left(1 - e^{-\frac{\lambda_b+\lambda_x}{r2^k}}\right)\mathbf{c}_k^{(B),<} \\
&\quad + \sum_{k=1}^{q+1} \log\left(1 - e^{-\frac{\lambda_a}{r2^{\min\{k,q\}}}}\right)\mathbf{c}_k^{(A),>} + \log\left(1 - e^{-\frac{\lambda_b}{r2^{\min\{k,q\}}}}\right)\mathbf{c}_k^{(B),>} \\
&\quad + \sum_{k=1}^{q+1} \log\left(1 - e^{-\frac{\lambda_a+\lambda_x}{r2^{\min\{k,q\}}}} - e^{-\frac{\lambda_b+\lambda_x}{r2^{\min\{k,q\}}}} + e^{-\frac{\lambda_a+\lambda_b+\lambda_x}{r2^{\min\{k,q\}}}}\right)\mathbf{c}_k^{=} \\
&\quad - \frac{\lambda_a}{r}\sum_{k=0}^{q}\frac{\mathbf{c}_k^{(A),<} + \mathbf{c}_k^{=} + \mathbf{c}_k^{(A),>}}{2^k} - \frac{\lambda_b}{r}\sum_{k=0}^{q}\frac{\mathbf{c}_k^{(B),<} + \mathbf{c}_k^{=} + \mathbf{c}_k^{(B),>}}{2^k} - \frac{\lambda_x}{r}\sum_{k=0}^{q}\frac{\mathbf{c}_k^{(A),<} + \mathbf{c}_k^{=} + \mathbf{c}_k^{(B),<}}{2^k}.
\end{aligned} \tag{33}$$

Like Eq. (**??**), this function depends on the statistics:

$$\begin{aligned}
\mathbf{c}_k^{(A),<} &= |\{i \mid k = \mathbf{r}_i^{(A)} < \mathbf{r}_i^{(B)}\}|, \\
\mathbf{c}_k^{(A),>} &= |\{i \mid k = \mathbf{r}_i^{(A)} > \mathbf{r}_i^{(B)}\}|, \\
\mathbf{c}_k^{(B),<} &= |\{i \mid k = \mathbf{r}_i^{(B)} < \mathbf{r}_i^{(A)}\}|, \\
\mathbf{c}_k^{(B),>} &= |\{i \mid k = \mathbf{r}_i^{(B)} > \mathbf{r}_i^{(A)}\}|, \\
\mathbf{c}_k^{=} &= |\{i \mid k = \mathbf{r}_i^{(A)} = \mathbf{r}_i^{(B)}\}|,
\end{aligned} \tag{34}$$

which capture the differences in register list distribution. The Naïve inclusion-exclusion estimator implemented using the maximum likelihood optimization of Eq. (**??**) depends on the count statistics

$$\begin{aligned}
\mathbf{c}^{(A)} &= \mathbf{c}^{(A),<} + \mathbf{c}^{=} + \mathbf{c}^{(A),>}\mathbf{c}, \\
\mathbf{c}^{(B)} &= \mathbf{c}^{(B),<} + \mathbf{c}^{=} + \mathbf{c}^{(B),>}\mathbf{c}, \\
\mathbf{c}^{(A\cup B)} &= \mathbf{c}^{(A),>} + \mathbf{c}^{=} + \mathbf{c}^{(B),>}\mathbf{c},
\end{aligned} \tag{35}$$

which lose information present in the more detailed count statistics statistics in Eq. (**??**). Algorithm 9 of [**?**] describes the estimation of $|A \setminus B|$, $|B \setminus A|$, and $|A \cap B|$ by accumulating the sufficient statistic (**??**) and using it to find the maximum of Eq. (**??**) via maximum likelihood estimation. The author shows extensive simulation evidence indicating that this method significantly improves upon the estimation error of a naïve estimator. They also show evidence suggesting that this estimator can be used to obtain a better estimate of $|A \cup B|$ than the naïve method, which involves composing a new sketch by computing the elementwise maximum of each register and taking its estimate. In the parlance of Algorithm **??** this is tantamount to

computing ESTIMATE $\left(\text{MERGE}\left(S^{(A)}, S^{(B)}\right)\right)$. We reaffirm these conclusion with our findings in Section **??**, where we use this algorithm to estimate the intersection of sets underlying pairs of sketches.

Algorithm **??** summarizes the various improved estimation procedures that we have described. In practice we use the machinery of Algorithm **??** along with the estimators of Algorithm **??**.

Algorithm 10 HYPERLOGLOG Maximum Likelihood Estimators

```
1: function EstimateMLE(S)
2:     Compute a statistic c of form Eq. (??)
3:     return MLE of (??) (e.g. Algorithm 8 of [?])
4: function NaïveIntersection(S^(A), S^(B))
5:     return EstimateMLE(S^(A)) + EstimateMLE(S^(B))
           −EstimateMLE(Merge(S^(A), S^(B)))
6: function EstimateUnion(S^(A), S^(B))
7:     Compute statistics c^(A),<, c^(A),>, c^(B),<, c^(>),> and c^= of form (??)
8:     return sum of MLEs of (??) (e.g. Algorithm 9 of [?])
9: function EstimateIntersection(S^(A), S^(B))
10:    Compute statistics c^(A),<, c^(A),>, c^(B),<, c^(>),> and c^= of form (??)
11:    return Intersection MLE of (??) (e.g. Algorithm 9 of [?])
```

14 Intersection Estimation Limitations: Dominations and Small Intersections

We have noted that there are limitations to the sketch intersection estimation in Section **??**. There appear to be two main sources of large estimation error in practice.

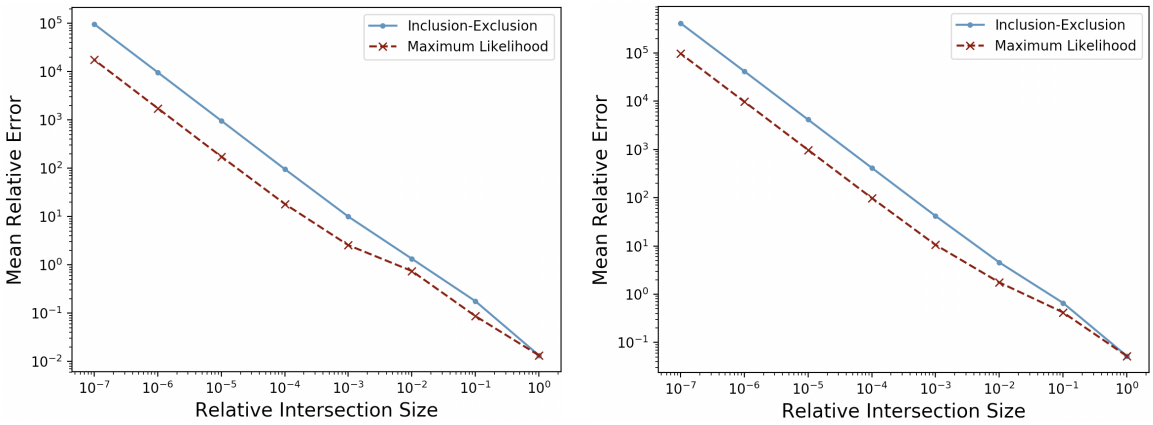
The first is the phenomenon where $\mathbf{r}_i^{(A)} > \mathbf{r}_i^{(B)}$ for all i where $\mathbf{r}_i^{(B)} > 0$, resulting in $\mathbf{c}_k^{(A),<} = \mathbf{c}_k^{(B),>} = 0$ for all k and $\mathbf{c}_k^{=} = 0$ for all $k > 0$. We say that such an A *strictly dominates* B. In this case, Eq. (**??**) can be rewritten as the sum of functions depending upon λ_a and $\lambda_b + \lambda_x$. This means that the optimization relative to λ_a does not depend upon λ_x or λ_b, and is given by $\widetilde{\lambda}_{(A)} = \text{ESTIMATEMLE}(S^{(A)})$. The optimization relative to $\lambda_b + \lambda_x$ is similarly independent of λ_a, and thus is given by $\widetilde{\lambda}_{(B)} = \text{ESTIMATEMLE}(S^{(B)})$. Consequently, Eq. (**??**) does not specify an estimator for λ_x in this case, as it could be anything between 0 and $\widetilde{\lambda}_{(B)}$ without affecting the optimum.

We also consider the phenomenon where $\mathbf{r}_i^{(A)} \geq \mathbf{r}_i^{(B)}$ for all i, resulting in $\mathbf{c}_k^{(A),<} = \mathbf{c}_k^{(B),>} = 0$ for all k. We say that such an A *dominates* B. We are unable to make the same analytic statements about Eq. (**??**), as the terms dependent upon $\mathbf{c}^{=}$ are not eliminated. Consequently, the optimum estimate for λ_a depends upon λ_b and λ_x. If A dominates B, the count statistics given by Eq. **??** are unable to distinguish whether B is subset of A. Given the construction of Eq. (**??**), many and large nonzero values for $\mathbf{c}_k^{=}$ for large k will bias the optimization towards larger intersections, whereas the converse is true if $\mathbf{c}_k^{=}$ is nonzero for only a few small values of k. If $|A| \gg |B|$, then the latter might occur whether $|A \cap B|$ is large or small. Furthermore, note that if B is a subset of A, then A will (possibly strictly) dominate B. Our experiments in Section **??** indicate that dominations are just as bad as strict dominations in terms of the resulting estimation error.

If A dominates B, then $S^{(A \cup B)} = \text{MERGE}(S^{(A)}, S^{(B)}) = S^{(A)}$. Ergo, the inclusion-exclusion naïve intersection estimator produces the estimate NAÏVEINTERSECTION = $\text{ESTIMATEMLE}(S^{(B)}) = \lambda_{(B)}$. This estimate is dubious, given that we have no evidence that the sets A and B hold any elements in common. This is especially true if $|A| \gg |B|$. Hence, both the naïve and maximum likelihood estimators may suffer from bias when a domination event occurs.

Consequently, it might be safest to disregard dominations in practice, as doing so can greatly reduce mean relative error. However, this poses a problem for our applications, as we will frequently have to compare the sketches of high degree vertices with those of comparatively low degree.

We have also noted the problem of small intersections. As discussed above, the intersection estimate optimized by Eq. (**??**) is proportional to the number (and size) of the nonzero $\mathbf{c}_k^{=}$ for $k > 0$, where larger k



(a) $p = 8$.

(b) $p = 12$.

Figure 1: HLL inclusion-exclusion and maximum likelihood intersection estimator performance where $|A| = |B| = 10^7$ and $|A \cap B|$ varies from 1 up to $|B|$. Increasing p increases intersection estimator performance.

biases the estimate toward larger intersections. If the ground truth intersection is small relative to $|A|$ and $|B|$, however, Eq. (**??**) will exhibit high variance. We explore this phenomenon empirically in Section **??**.

One immediate conclusion that we can draw from this analysis is that the intersection estimator is biased – it will tend to overestimate small intersections and intersections where one sketch dominates the other.

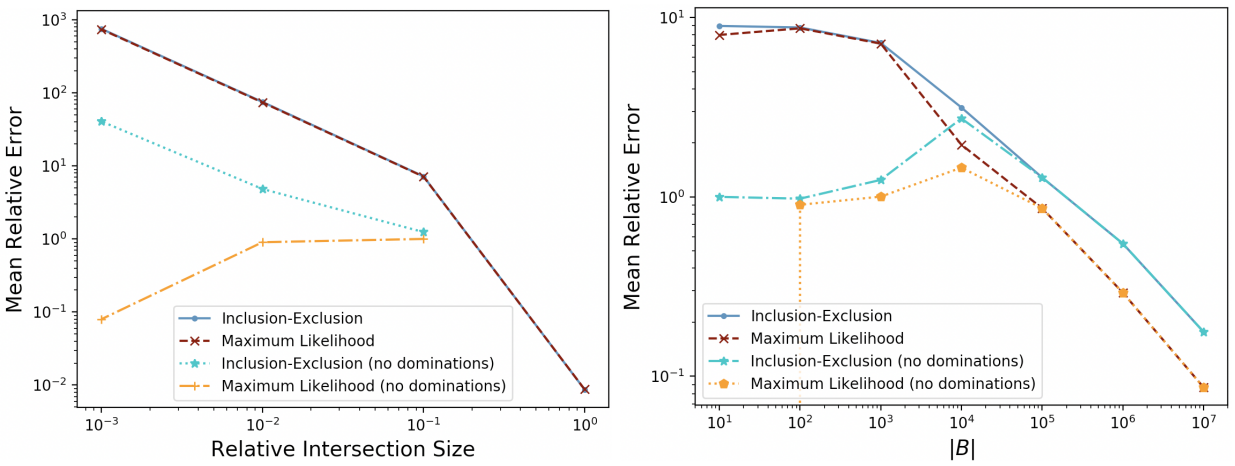
15 Experiments

15.1 Intersection Estimation: Small Intersections and Dominations

We will begin with an analysis of the claims in Section **??** having to do with the error of intersection estimation where the intersection is small or where one of the two sets dominates the other. We analyzed both the naïve and maximum likelihood intersection estimators for 1000 iterations for different settings of two sets, A and B. In all experiments, $|A| = 10^7$. We considered all settings of $|B| = 10^j$ for $j \in [7]$. For $|B| = 10^j$, we considered all settings of $|A \cap B| = 10^i$ for $i \in [0, j]$, varying from a single element intersection up to $B \subseteq A$. For each setting of $|B|$ and $|A \cap B|$, we ran 1000 experiments over sets of random numbers satisfying the specified relationship between A and B, using a fresh hash function and fresh random numbers in each iteration. Our implementation uses murmurhash3 [**?**] as the HLL hash function. We used prefix size $p = 12$. Figure **??** plots the mean relative error as a function of the size of $|A \cap B|$ for the case where $|A| = |B|$.

Figure **??** plots the error of the naïve inclusion-exclusion estimator and the maximum likelihood estimator of $|A| = |B| = 10^7$ where $|A \cap B|$ varies from 1 up to 10^7 and $p = 8$. The sketches in question then have $r = 2^8 = 256$ registers. While the maximum likelihood estimator generally outperforms the naïve estimator, both exhibit unacceptably poor performance when the true intersection is very small. Meanwhile, Figure **??** plots the same parameters where $p = 12$ and so $r = 2^{12} = 4096$. In particular, the error of the maximum likelihood estimator remains at reasonable levels for much smaller intersections. The mean relative error remains below 1 up until about the point where $|A \cap B| = \frac{|A|}{10^2}$.

Clearly, performance can clearly be boosted by increasing the precision parameter p, thereby increasing the number of registers. For fixed set sizes, one can choose p to achieve arbitrarily small precision. However, as the sizes of the sets increase a fixed p will cease to suffice, meaning that the size of the sketches is no longer independent of the streaming data size up to logarithmic factors, which is highly undesirable. Thus, we can expect the triangle counting algorithms such as those described in Sections **??** and **??** to exhibit poor performance when estimating the incident triangles on an edge linking two hubs where the ground truth triangle count is small. In our implementation on commodity hardware, the maximum likelihood estimations took approximately 2.7732824 times as much time as the naïve estimations.



(a) Mean relative error as a function of relative intersection size where $|A| = 10^7$ and $|B| = 10^3$.

(b) Mean realtive error as a function of $|B|$, where $|A \cap B| = \frac{|B|}{10}$.

Figure 2: More comparisons between HLL inclusion-exclusion and maximum likelihood intersection estimator performance where $|A| = 10^7$ and $p = 12$.

Meanwhile, Figure **??** plots the same relationship where $|B| = 10^3 = \frac{|A|}{10^4}$. This plot shows us a somewhat different picture. While both estimators perform poorly when the intersection is small, both appear to consistently estimate the intersection to be $|B| = 10^3$. This is due to the observation of the effects of dominations from Section **??**. Firgure **??** also plots the error of just the comparisons where A does not dominate B at all (about 25% of the experiments). Eliminating the strict dominations does not dramatically improve mean performance, which appears asymptotically consistent with the aggregate. However, removing all estimates that involve a domination results in *improved* performance of both estimators. In fact, the maximum likelihood estimator actually appears to improve as the intersection size decreases. On the other hand, the non-domination inclusion-exclusion estimates appear to exhibit relative error that is better, but not dramatically better, than the domination estimates.

Figure **??** plots the mean relative error as a function of $|B|$, where $|B|$ takes values in 10^j for $j \in [7]$ and $|A \cap B|$ is locked to $\frac{|B|}{10}$. As $|B|$ gets smaller, the likelihood of a domination increases. At $|B| = 10^4$ dominations occur in 6.6% of cases, at $|B| = 10^3$ dominations occur in 76.9% of cases, at $|B| = 10^2$ dominations occur in 97.5% of cases, and at $|B| = 10$ dominations occur in 99.8% of cases. In particular in the two cases where $|B| = 10$ and $|A \cap B| = 1$ and a domination does not occur, the maximum likelihood estimator returns exactly 1. So for a fixed intersection size relative to $|B|$, both the inclusion-exclusion and maximum likelihood estimators return more reasonable estimates when dominations do not occur. Figure **??** indicates that the maximum likelihood estimator benefits more as the intersection decreases.

These results suggest that, had we a means of avoiding dominations, then the maximum likelihood estimator would be at least coarsely reliable when comparing sets of very different size. Unfortunately this is a difficult guarantee to provide. We can bound the probability that A dominates B for disjoint A and B building from Eqs. (1) and (2) of [**?**]. However, the resulting expression is very messy and rather uninformative to look at, so we do not reproduce it here. Suffice to say that the probability is bounded using the union bound by the normalized sum over all hashings of elements of A and B to their length r register arrays $\mathbf{r}^{(A)}$ and $\mathbf{r}^{(B)}$ of the products of the probabilities that $\mathbf{r}_i^{(A)} \geq \mathbf{r}_i^{(B)}$. By increasing r, we can make this bound arbitrarily small for a given $|A|$ and $|B|$ by doubling the number of registers for each incremental increase in p. A similar expression suffices for the case where $|A \cap B| \neq 0$, but the expression grows even more complex. However, as $|A \cup B|$ increases the bound also increases. Ergo, if we want to bound the probability that a domination occurs among disjoint sets of a given sizes, it requires setting r scaling proportional to $|A \cup B|$. This is clearly an infeasible condition for a streaming algorithm, as the size of the sketch is no longer independent of the size of the streaming data.

If $|A| \gg |B|$, then for a fixed precision p A will asymptotically dominate B. Increasing p may suffice given

a ceiling upon the size of the sets, but we have sketched how this is not a valid general purpose solution. Unfortunately, this means that unless the size of possible sets has a reasonable bound, it it not feasible to reliably recover intersections of very differently sized sets.

These negative results set a dour tone for our subsequent analysis. We have shown that the intersection estimator exhibits poor performance on very small intersections of equally sized sets and the intersections of sets of wildly different sizes, and we have explored why these gaps in performance occur. Thus, the performance of the algorithms of Sections **??** and **??** are dependent upon how prevalent these cases are in the data. This is the reason that we limit ourselves to empirical heavy hitter recovery.

15.2 Local Triangle Counting

We performed experiments on real graph datasets for the purpose of establishing the following of our Algorithms.

1. **Heavy Hitter recovery** Do the heavy hitters returned by Algorithms **??** and **??** correspond to the ground truth heavy hitters?
2. **Estimation Quality** Do the algorithms yield good global and edge- and vertex-local estimates? How does the maximum likelihood estimator compare to the naïve estimator?
3. **Speed & Scalability** How fast is accumulation? Estimation? How does wall time relate to $|\mathcal{P}|$?

Graphs: We considered the graphs listed in Table **??**. We simulated graph streams by randomly ordering and partitioning the edge lists into $|\mathcal{P}|$ separate files, one of which is read by each member of $\mathcal{P}$.

Many of these graphs are provided by Stanford's widely used SNAP dataset [**?**]. These graphs are collected from natural sources, such as email records, transportation networks, peer-to-peer communications, social media, and citation corpora, among others. We casted each graph as unweighted, ignoring directionality, self-loops, and repeated edges.

We also used 5 graphs derived from nonstochastic Kronecker products of smaller graphs. Nonstochastic Kronecker graphs [**?**] have adjacency matrices C that are Kronecker products $C = C_1 \otimes C_2$, where the factors are also adjacency matrices. This type of synthetic graph is attractive for testing graph analytics at massive scale [**?**, **?**], as ground truth solution is often cheaply computable. For such graphs, global triangle count and triangle counts at edges are computed via Kronecker formulas [**?**]: for a graph with m edges, the worst-case cost of computing global triangle counts is sublinear, $O\left(m^{\frac{3}{4}}\right)$, whereas the cost of computing the full set of edge-local counts is $O(m)$.

Here, we build $C = C_1 \otimes C_2$ from identical factors, $C_1 = C_2$, that come from a small set of graphs with m up to 10^5 from the University of Florida sparse matrix collection (`polbooks`, `celegans`, `geom`, `yeast` [**?**]). All graphs were forced to be undirected, unweighted, and without self loops. We compute the number of triangles at each edge for C_1 and use the Kronecker formula in [**?**] to get the respective quantities for C. Summing over the edges and dividing by 3 gives the global triangle count for C.

Hardware: All of the experiments were performed on a cluster of compute nodes with thirty-six 2.1 GHz Intel Xeon E5-2695 v4 cores and 128GB memory per node. We varied the number of nodes per experiment depending on scalability requirements and the size of the graph. Each core on each node is responsible for an equally sized partition of the vertices in the graph. We consider graph partitioning to be a separate problem, and accordingly use simple round-robin assignment for our experiments.

Implementation: We implemented all of our algorithms in C++ and MVAPICH2 2.3. Inter- and intra-node communication is managed using the pseudo-asynchronous MPI-enabled communication software package YGM described in Chapter **??**. We used xxhash as our hash function implementation [**?**].

Evaluation: We ran each experiment for a total of 100 iterations using different random seeds, using prefix size $p = 12$. We found in experiments that this size gave the best tradeoff between performance and accuracy.

graph	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{T}\rvert$	**graph**	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{T}\rvert$
as20000102	6,474	12,572	6,584	facebookcombined	4,039	88,234	1,612,010
ca-GrQc	5,242	14,484	48,260	p2p-Gnutella30	36,682	88,328	1,590
p2p-Gnutella08	6,301	20,777	2,383	ca-CondMat	23,133	93,439	173,361
oregon1010407	10,729	21,999	15,834	ca-HepPh	12,008	118,489	3,358,500
oregon1010331	10,670	22,002	17,144	p2p-Gnutella31	62,586	147,892	2,024
oregon1010414	10,790	22,469	18,237	email-Enron	36,692	183,831	727,044
oregon1010428	10,886	22,493	17,645	ca-AstroPh	18,772	198,050	1,351,440
oregon1010505	10,943	22,607	17,597	loc-brightkiteedges	58,228	214,078	494,728
oregon1010512	11,011	22,677	17,598	cit-HepTh	9,877	352,285	1,478,740
oregon1010519	11,051	22,724	17,677	email-EuAll	265,214	364,481	267,313
oregon1010421	10,859	22,747	19,108	**pb $\otimes$ pb**	11,024	388,962	1,881,600
oregon1010526	11,174	23,409	19,894	soc-Epinions1	75,879	405,740	1,624,480
ca-HepTh	9,877	25,973	28,339	cit-HepPh	34,546	420,877	1,276,870
p2p-Gnutella09	8,114	26,013	2,354	soc-Slashdot0811	77,360	469,180	551,724
oregon2010407	10,729	30,855	78,138	soc-Slashdot0902	82,168	504,230	602,592
oregon2010505	11,157	30,943	72,182	amazon0302	262,111	899,792	717,719
oregon2010331	10,900	31,180	82,856	loc-gowallaedges	196,591	950,327	2,273,140
oregon2010512	11,260	31,303	72,866	roadNet-PA	1,088,092	1,541,898	67,150
oregon2010428	11,113	31,434	78,000	roadNet-TX	1,379,917	1,921,660	82,869
p2p-Gnutella06	8,717	31,525	1,142	flickrEdges	105,938	2,316,948	107,987,000
oregon2010421	11,080	31,538	82,129	amazon0312	400,727	2,349,869	3,686,470
oregon2010414	11,019	31,761	88,905	amazon0505	410,236	2,439,437	3,951,060
p2p-Gnutella05	8,846	31,839	1,112	amazon0601	403,394	2,443,408	3,986,510
oregon2010519	1,1375	32,287	83,709	roadNet-CA	1,965,206	2,766,607	120,676
oregon2010526	11,461	32,730	89,541	graph500-scale18-ef16	174,147	3,800,348	82,287,300
p2p-Gnutella04	10,876	39,994	934	**cg $\otimes$ cg**	205,208	8,201,250	64,707,900
as-caida20071105	26,475	53,381	36,365	**ns $\otimes$ ns**	2,524,920	15,037,128	85,006,200
p2p-Gnutella25	22,687	54,705	806	cit-Patents	3,774,768	16,518,947	7,515,020
p2p-Gnutella24	26,518	65,369	986	**em $\otimes$ em**	1,283,688	59,426,802	171,286,000
				ye $\otimes$ ye	5,574,320	88,338,632	74,765,400

Table 1: Considered moderate graphs

We evaluated the accuracy of the estimates of each algorithm in terms of

$$\text{global relative error} = \frac{\left|T - \widetilde{T}\right|}{T}, \tag{36}$$

$$\text{mean relative error (vertices)} = \frac{1}{|\mathcal{V}|}\sum_{u\in\mathcal{V}} \frac{\left|\mathcal{C}^{\text{TRI}}(u) - \widetilde{\mathcal{C}}^{\text{TRI}}(u)\right|}{1+\mathcal{C}^{\text{TRI}}(u)}, \text{ and} \tag{37}$$

$$\text{mean relative error (edges)} = \frac{1}{|\mathcal{E}|}\sum_{uv\in\mathcal{E}} \frac{\left|\mathcal{C}^{\text{TRI}}(uv) - \widetilde{\mathcal{C}}^{\text{TRI}}(uv)\right|}{1+\mathcal{C}^{\text{TRI}}(uv)} \tag{38}$$

as appropriate. We will abbreviate "mean relative error" as MRE in figures and tables.

In addition, we computed the top k ground truth edge- and vertex-local triangle count elements of each graph, and compared them with the estimated top k elements. We considered multiple performance metrics using this approach.

One can make a similar comparison to the precision vs recall tradeoff by directly comparing the orderings. Kendall's rank correlation coefficient, often colloquially referred to as Kendall's τ, is a canonical non-parametric statistic that quantifies the similarity between different indexings of the same data [**?**]. Vigna generalized Kendall's τ to a correlation coefficient that is robust to ties within indexings and permits weightings of different indices. This generalization is an additive hyperbolic weighted correlation coefficient, and we will refer to it as Vigna's $\tau_{h,\phi}$ [**?**]. $\tau_{h,\phi}$ is a correlation coefficient on two indexings $\{s_i\}_{i=1}^n$ and $\{r_i\}_{i=1}^n$ of the set of elements $\{1,2,\ldots,n\}$. Assume that $\phi : \{1,2,\ldots,n\} \to \{1,2,\ldots,n,\infty\}$ is the map of elements to their ground truth rank, where $\phi(i) = \infty$ indicates that i has null rank. Let

$$\langle r, s\rangle_{h,\phi} = \sum_{i<j} \text{SGN}(x_i - x_j)\text{SGN}(y_i - y_j)\left(\frac{1}{\phi(i)+1} + \frac{1}{\phi(j)+1}\right), \tag{39}$$

where

$$\text{SGN}(z) \coloneqq \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}. \tag{40}$$

If we further define the norm $\|r\|_{h,\phi} = \sqrt{\langle r, r\rangle_{h,\phi}}$, $\tau_{h,\phi}$ can be expressed as

$$\tau_{h,\phi} = \frac{\langle r, s\rangle_{h,\phi}}{\|r\|_{h,\phi}\|s\|_{h,\phi}}. \tag{41}$$

This coefficient yields a value $-1 \le \tau_{h,\phi} \le 1$, where $\tau_{h,phi} = 1$ indicates perfect agreement and $\tau_{h,phi} = -1$ indicates perfect disagreement between the indexings. The use of the hyperbolic function ensures that higher ground truth priority elements bear a greater impact upon the correlation. We use the ranking function

$$\phi_r(z) \coloneqq \begin{cases} i & \text{if } i \le r \text{ and } z \text{ is the } i\text{th largest element} \\ \infty & \text{else} \end{cases}, \tag{42}$$

using τ_{h,ϕ_r} to measure top r correlation. Vigna suggests this function in [**?**] as a means of performing top-r correlation comparisons. When one of the indexings is a ground truth ordering and the other is an estimate, τ_{h,ϕ_r} is affected by only the top r estimated elements, as well as the estimated indices of each of the ground truth elements. This allows us to efficiently compute Vigna's τ_{h,ϕ_r} on massive orderings in a distributed fashion in a single pass in $\widetilde{O}(n)$ time and $\widetilde{O}(1)$ space, where r is treated as a constant. We used $r = 100$ in our experiments.

15.2.1 Heavy Hitter Recovery

For an experiment consisting of a graph G, a prefix size p, a number of heavy hitters k, and a number of estimated heavy hitters k', we run an implementation of Algorithm **??** using HyperLogLog sketches with

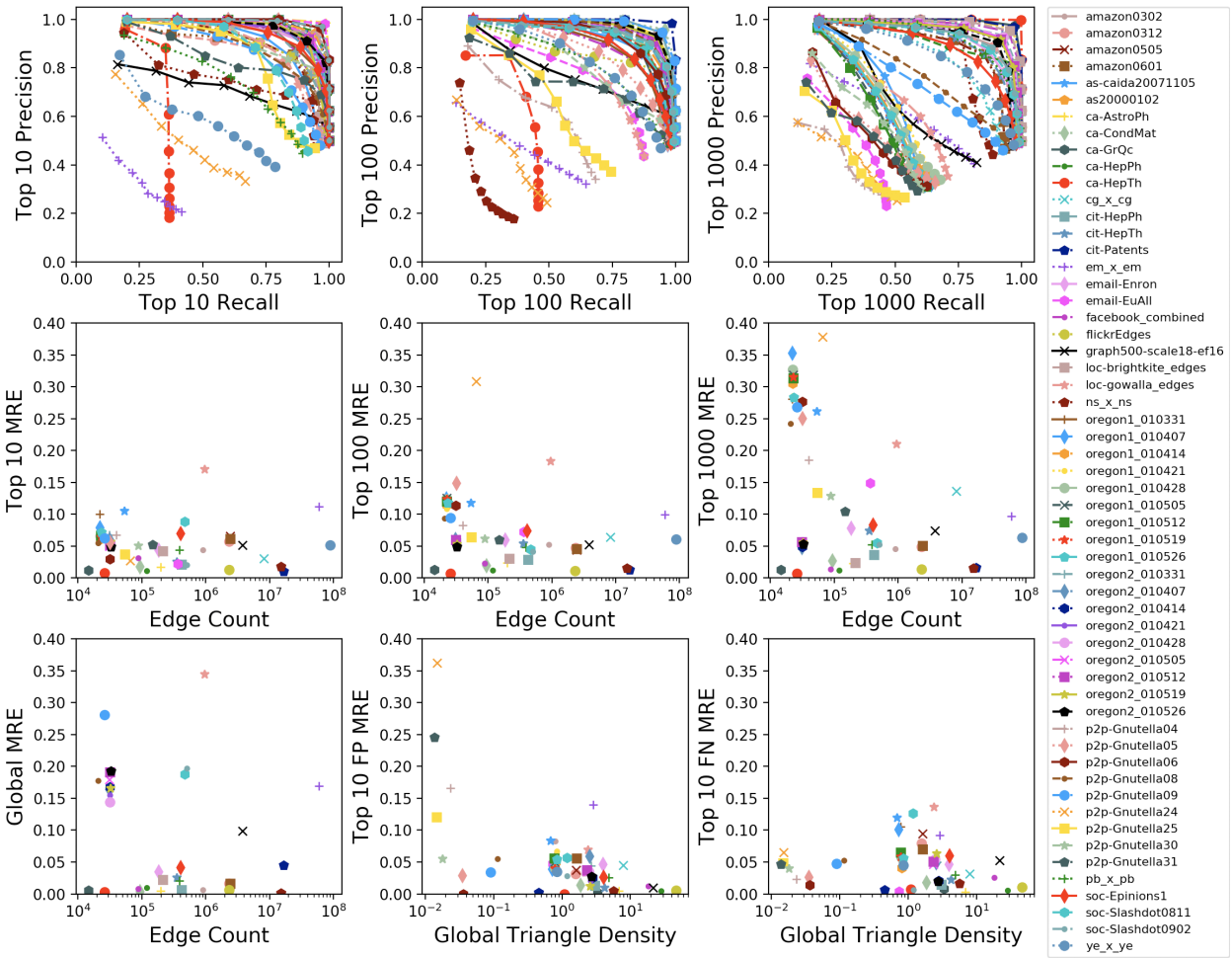


Figure 3: Precision vs. recall and mean relative error (MRE) for the top 10, top 100, and top 1000 ground truth heavy hitters of all graphs listed in Table **??** using $p = 12$ and the naïve estimator. Also plots the mean relative error for the global estimates and the top 10 false positives and false negatives. All mean relative error plots use $k' = k$.

$m = 2^p$ registers and accumulate a k'-min-heap $\widetilde{H}_{k'}$. We then compare this heap to H_k, the true set of top k triangle count edges in G. We can also compare $\widetilde{\mathcal{T}}$ to $\mathcal{T}$.

We treat $H_{k'}$ as a one-class classifier of the top k elements in H_k, implicitly labelling all other vertices as "not heavy hitters". Accordingly, an edge $e \in \mathcal{E}$ is a true positive (TP) if $e \in H_k$ and $e \in \widetilde{H}_{k'}$, a false negative (FN) if $e \in H_k$ and $e \notin \widetilde{H}_{k'}$, a false positive (FP) if $e \notin H_k$ and $e \in \widetilde{H}_{k'}$, or a true negative (TN) if $e \notin H_k$ and $e \notin \widetilde{H}_{k'}$. We can report the quality of an experiment in terms of its recall $\left(\frac{TP}{TP+FN}\right)$ versus its precision $\left(\frac{TP}{TP+FP}\right)$. The precision versus recall tradeoff is a common metric in information retrieval, where the goal is to tune model parameters so as to force both the precision and recall as close to one as possible. In our experiments, we vary k' around a stationary k to demonstrate the tradeoffs of accumulating more or less than the target number of heavy hitters. Although these measures are know to exhibit bias, they are accepted as being reasonable for heavily uneven classification problems such as ours, where the class of interest is a small proportion of the samples [**?**].

Figure **??** shows the edge local precision versus recall curves for all of the graphs in Table **??** for $k = 10, 100, 1000$, where we vary k' from $0.2k$ to $2k$. These figures are generated using the naïve estimator. The figure also plots the mean relative error of these top 10, top 100, and top 1000 edges. They also plot the global mean relative error and the mean relative errors of the false positives and the false negatives. This

false positive relative error is the mean error observed by false positives - edges that are incorrectly identified as heavy hitters. Similarly, the false negative relative error is the mean error observed by false negatives -edges that the algorithm fails to correctly identify as heavy hitters. Low false positive error indicates that the true heavy hitters are still accurately estimated, while low false negative error indicates that those edges falsely identified as top k' elements are still These results were published by Priest, Pearce and Sanders in [**?**].

Figure **??** indicates that while the edge-local algorithm returns good performance on heavy hitter recovery for many of these graphs, some exhibit poor performance. Figure **??** compares three of these poor performers (em $\otimes$ em, p2pGnutella24, and ca-HepTh) with a graph that exhibits good performance (cit-Patents). The figure plots the triangle count for the ordered top 10^4 edges in each graph, as well as the triangle density for the same edges, in the same order. The cit-Patent graph demonstrates near-optimal features of a graph given our approach. The triangle count distribution is roughly scale-free, making it easier to differentiate top k elements from others for small k. Furthermore, the triangle density of most of these edges is nontrivial, avoiding the noted problem stemming from estimating small intersections. We will discuss how the other three graphs fail to

Consider first the the em $\otimes$ em graph. This is a synthetic kronecker graph, whose formula tends to promote the jagged triangle count distribution seen in Figure **??**. This results in many triangle count ties, which is problematic for our analysis. Indeed, this is a weakness of the approach. Furthermore, all of the heavy hitter edges exhibit low triangle density, with the majority being trivial. This leads to increased estimation error as discussed in Section **??**, which can be observed in Figure **??**. This error exacerbates the problems introduced by having many ties.

Now consider the p2p-Gnutella24 graph. This graph also exhibits many ties, but this is due to extreme sparsity of triangles present in the graph. Indeed, all but 14 edges in the graph have 2 or fewer incident triangles, and nearly all edges exhibit trivial triangle density. This graph also exhibits the most estimation error of those considered in our experiments for this reason. This indicates that our approach is probably not suitable for graphs with very few triangles. It is worth noting, however, that we are reliably able to capture the top 2 triangle count elements.

Finally, consider the ca-HepTh graph, which encodes the collaboration network of the Arxiv high energy physics theory community. This graph, a union of cliques, proves to be a pathological for this problem. Note that, save the top 2 edges, the top ~ 300 edges are tied in triangle count. Although our algorithm is able to reliably recover these top 2 elements, even an algorithm with perfect accuracy would fail up to ties on this graph for $k > 2$ and $k <\sim 300$, which our results reflect. Indeed, Figure **??** indicates that this graph exhibits some of the lowest relative error in our experiments. This indicates that, even for graphs with favorable triangle density, ties can lead to poor heavy hitter recovery. This problem could be partially ameliorated via a dynamically sized heap like that utilized in Theorem **??**.

Figure **??** also tells us that as k increases, the relative estimation error of the top k elements tends to increase. This is sensible, as both the true triangle counts and triangle density tend to decrease as we begin to consider smaller elements. It is worth noting that some of these graphs would exhibit better performance for different settings of k, such as noted p2p-Gnutella and ca-HepTh perform well when $k = 2$, because the top two elements stand out above the others. Unfortunately, in practice we typically cannot know how many outliers there are ahead of time.

Furthermore, the global mean relative error for most of these graphs is reasonable. However, there are some notable outliers, which are mostly overestimates due to many small intersections. We will show that the maximum likelihood estimator dramatically increases the estimation accuracy in practice.

15.2.2 Performance comparison between MLE and naïve estimators

We have shown some results concerning the naïve estimators. It is interesting to compare the performance of the naïve estimator to the noted superior performance of the MLE estimator.

We choose to present the results in terms of the distribution of signed error for vertices and edges. The

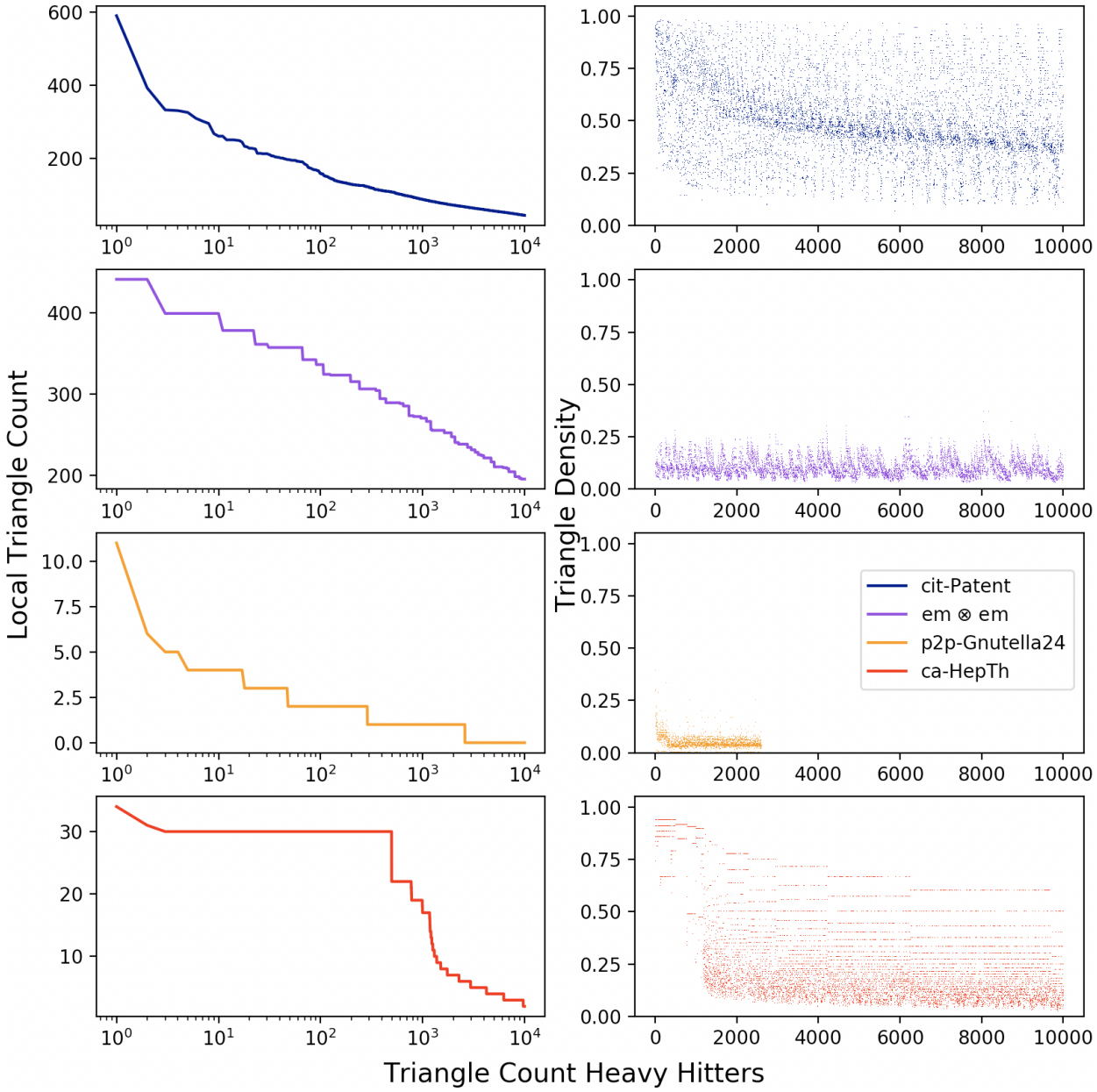


Figure 4: The triangle counts and triangle densities of the edge-local triangle count heavy hitters up to 10^4 for four graphs. The cit-Patent graph exhibits good performance in Figure **??**, and demonstrates a reasonable triangle count distribution as well as high triangle density throughout. The other three graphs demonstrate poor performance in Figure **??**. The kronecker em $\otimes$ em graph exhibits an unusual number of ties in its triangle count distribution due to its construction, in addition to low triangle density among its heavy hitters. The P2P-Gnutella24 graph has very low triangle density, and a 3 or fewer triangles for the vast majority of its edges. The ca-HepTh graph exhibits an unusual triangle distribution, where a huge portion of its edges tie at 30 triangles. Consequently, even a perfect heavy hitter extraction procedure will fail on this graph. Notably, the two edges with the largest triangle counts are reliably returned.

signed error of edges and vertices are given by

$$\text{signed relative error (vertex } x) = \frac{\widetilde{\mathcal{C}}^{\text{TRI}}(x) - \mathcal{C}^{\text{TRI}}(x)}{1 + \mathcal{C}^{\text{TRI}}(x)}, \text{ and} \tag{43}$$

$$\text{signed relative error (edge } xy) = \frac{\widetilde{\mathcal{C}}^{\text{TRI}}(xy) - \mathcal{C}^{\text{TRI}}(xy)}{1 + \mathcal{C}^{\text{TRI}}(xy)}. \tag{44}$$

The signed error of edge xy varies from -1, where $\widetilde{\mathcal{C}}^{\text{TRI}}(xy)$ is incorrectly zero, to $\min\{\mathbf{d}_x, \mathbf{d}_y\}$, where $\widetilde{\mathcal{C}}^{\text{TRI}}(x) = 0$ and a domination occurs, in which case the returned estimate is $\min\{\mathbf{d}_x, \mathbf{d}_y\}$. The signed error of a vertex x is similarly bounded above by $\frac{1}{2}\sum_{y:xy\in\mathcal{E}} \min\{\mathbf{d}_x, \mathbf{d}_y\}$.

We ran two experiments of Algorithms **??** and **??** on the graphs in Table **??**, disregarding loc-brightkite and loc-gowalla, one using the naïve estimator and another using the maximum likelihood estimator. We used $p = 12$ for each experiment. We collected the distribution of signed relative error for vertices and edges in each experiment. As there are a huge number of observations of each phenomenon in each experiment, we only ran each experiment once.

Figure **??** plots the distributions of edge local signed errors for the graphs using the maximum likelihood estimator. Note that the experimental distributions for all vertices appear roughly unbiased around 0, which is highly desirable. Compare these results with those of the naïve inclusion-exclusion estimator in Figure **??**. We have widened the horizontal axis to account for signed error larger than 1. The jaggedness of the distributions is due to the fact that we round our approximations to the nearest integer. Note that a few plots have spikes at 5. This is because 5 is the largest histogram bucket that we maintained in our experiments, and so anything larger than 5 gets counted as 5. In any case, these plots indicate that not only are the naïve results biased in most observed graphs, but also they have a propensity to severely overestimate edge local triangle counts. Thus, the MLE estimator is much more reliable for estimation in general, and can also be used for heavy hitter recovery.

Figure **??** plots the distributions of vertex local signed errors for the graphs using the maximum likelihood estimator. We should expect worse performance for vertex local triangle count estimation, as the estimates are composed of many triangle count estimates. Indeed, we find that the experimental results no longer appear unbiased, as many vertices appear to underestimate their true triangle counts in most graphs. Compare these results with those of the naïve inclusion-exclusion estimator in Figure **??**. Similar to Figure **??**, we have widened the horizontal axis and some plots exhibit spikes at 5. Figure **??** tells a similar story to Figure **??**, with large overestimates in a large proportion of the vertex population. Again, we find that the MLE estimator is much more reliable.

We should be careful about generalizing these results to truly massive graphs, however. As we noted in Section **??**, maintaining a constant bound on dominations is dependent upon graph size. That is, for fixed p, there will be a point as we increase the size of scale-free graphs where the error distribution will cease to look like the better examples in Figure **??**. For example, consider Figure **??**, which plots the edge local signed error distributions for the larger Twitter graph (see Table **??**). This plot shows that the larger twitter graph involves a large number of dominations, which result in a heavy tail in the error distribution. When dominations are discounted, however, the result is a much more well behaved distribution. While maintaining a low proportion of dominations is key, as we have established the only way to do so in the presence of increasing set size is to increase p. Thus, in order to scale to immense graphs sketch size must increase as well, which is an undesirable feature.

Consider also Figure **??**. This figure plots the distribution of observed relative errors versus triangle density for the em $\otimes$ em graph and the same Twitter graph explored in Figure **??**. Note that the twitter graph is about two orders of magnitude larger than the em $\otimes$ em graph. We see a marked improvement in the distribution of error for the em $\otimes$ em graph as p increases. While $p = 12$ is clearly sufficient for the em $\otimes$ em graph, scaling to the Twitter graph clearly requires larger p.

15.2.3 MLE performance statistics

Table **??** examines the maximum likelihood estimator performance on the graph in Table **??** in specific detail. Note that these are the data from only a single run of the algorithm on each graph, as this is sufficient information to glean the distribution of errors on edge and vertex local estimates because each

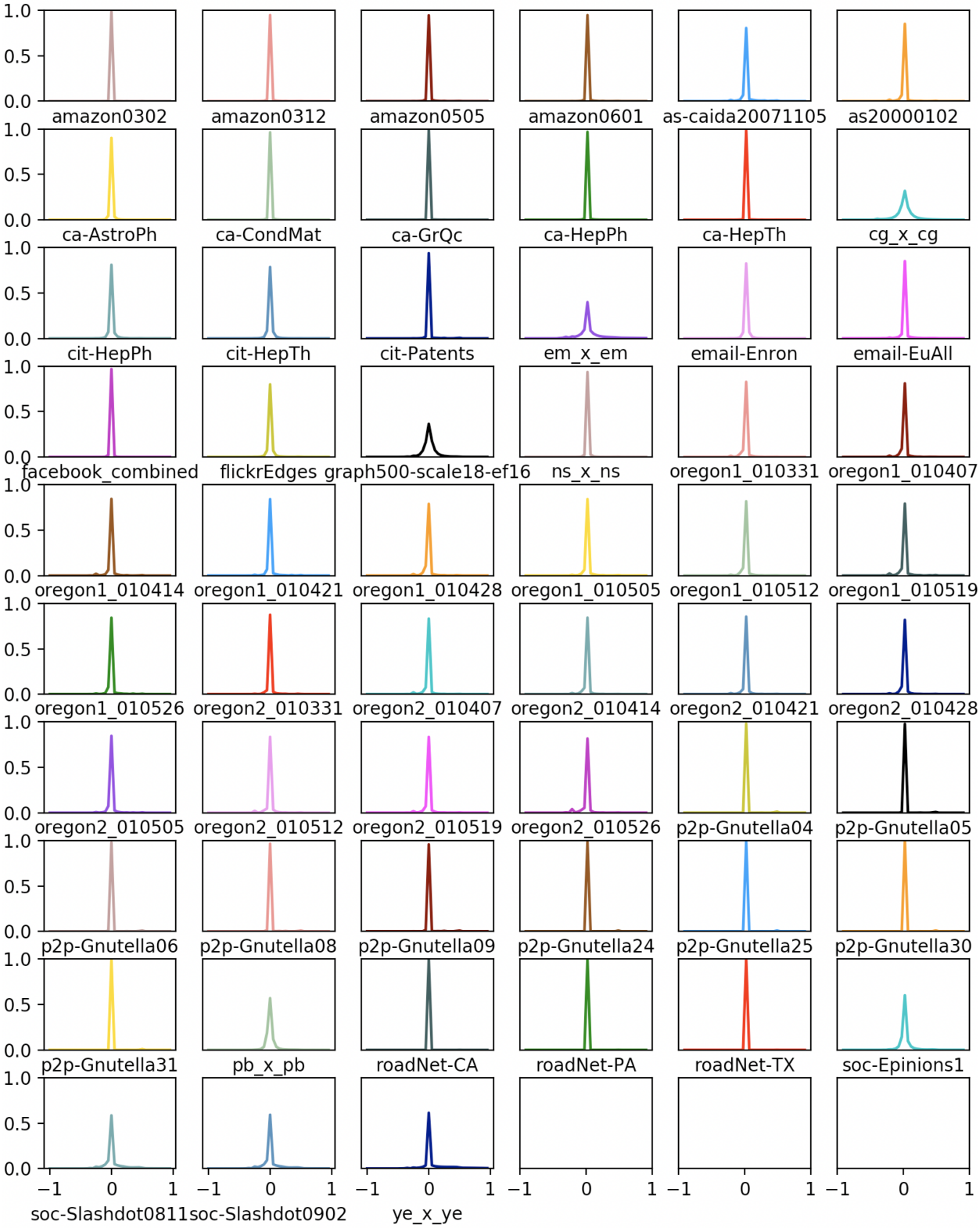


Figure 5: The edge-local relative error of most of the Table **??** using $p = 12$. Note that the experimental error appears to be largely unbiased.

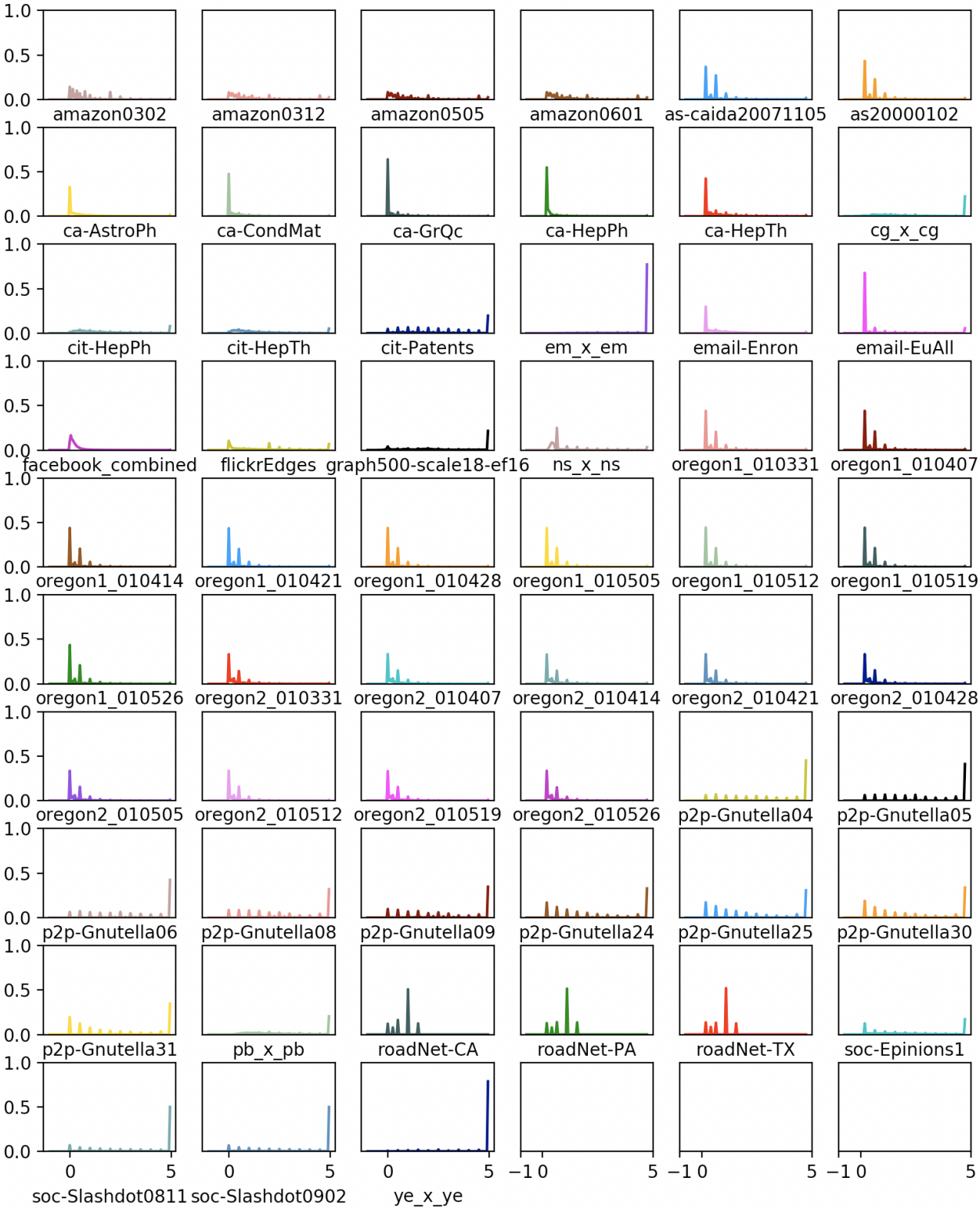


Figure 6: The edge-local relative error of most of the Table **??** using $p = 12$ and the naïve inclusion-exclusion intersection estimator. Note that we have had to widen horizontal axis to account for large overestimates in graphs. Note the degraded performance compared to Figure **??**.

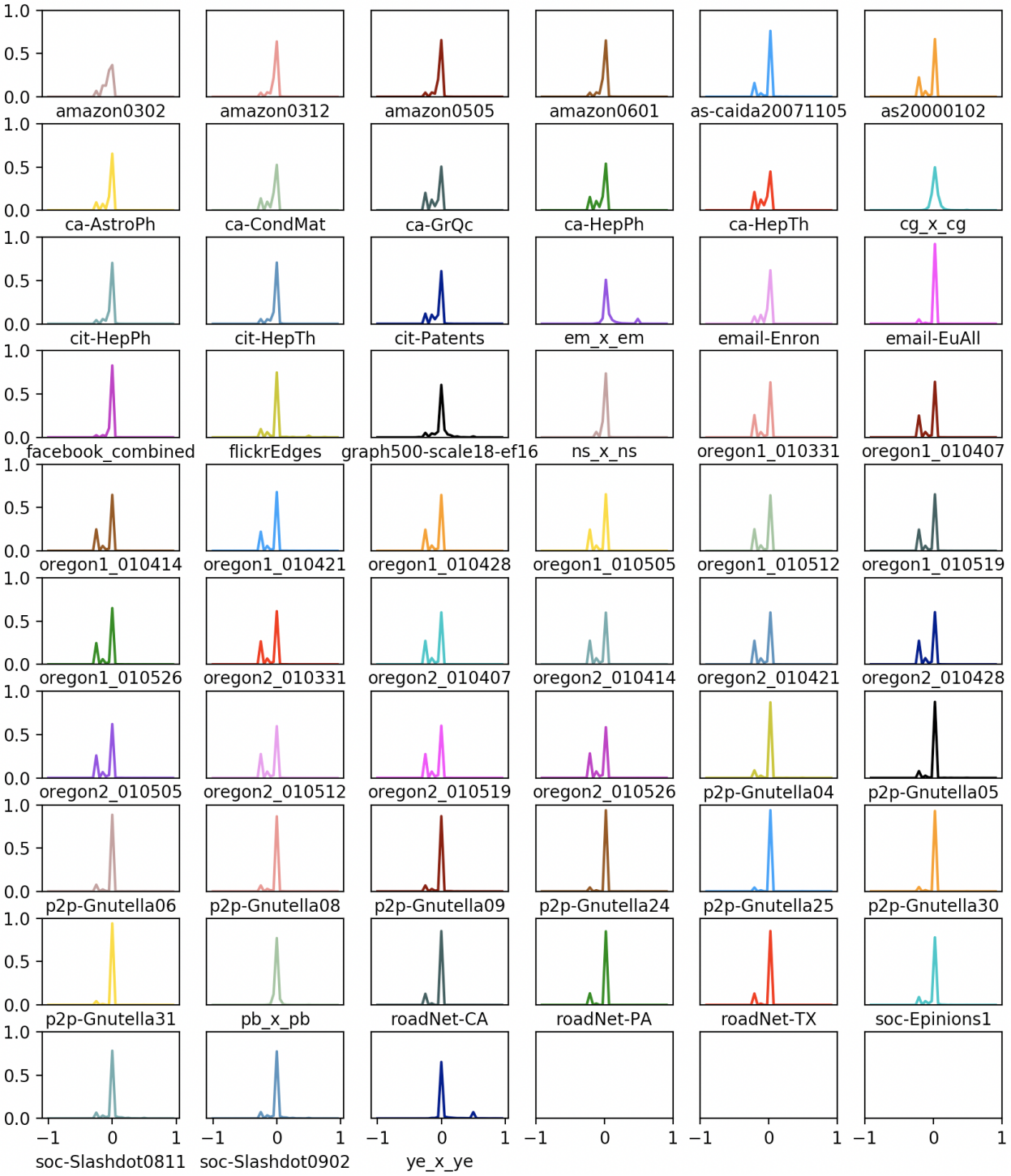


Figure 7: The vertex-local relative error of most of the Table **??** using $p = 12$.

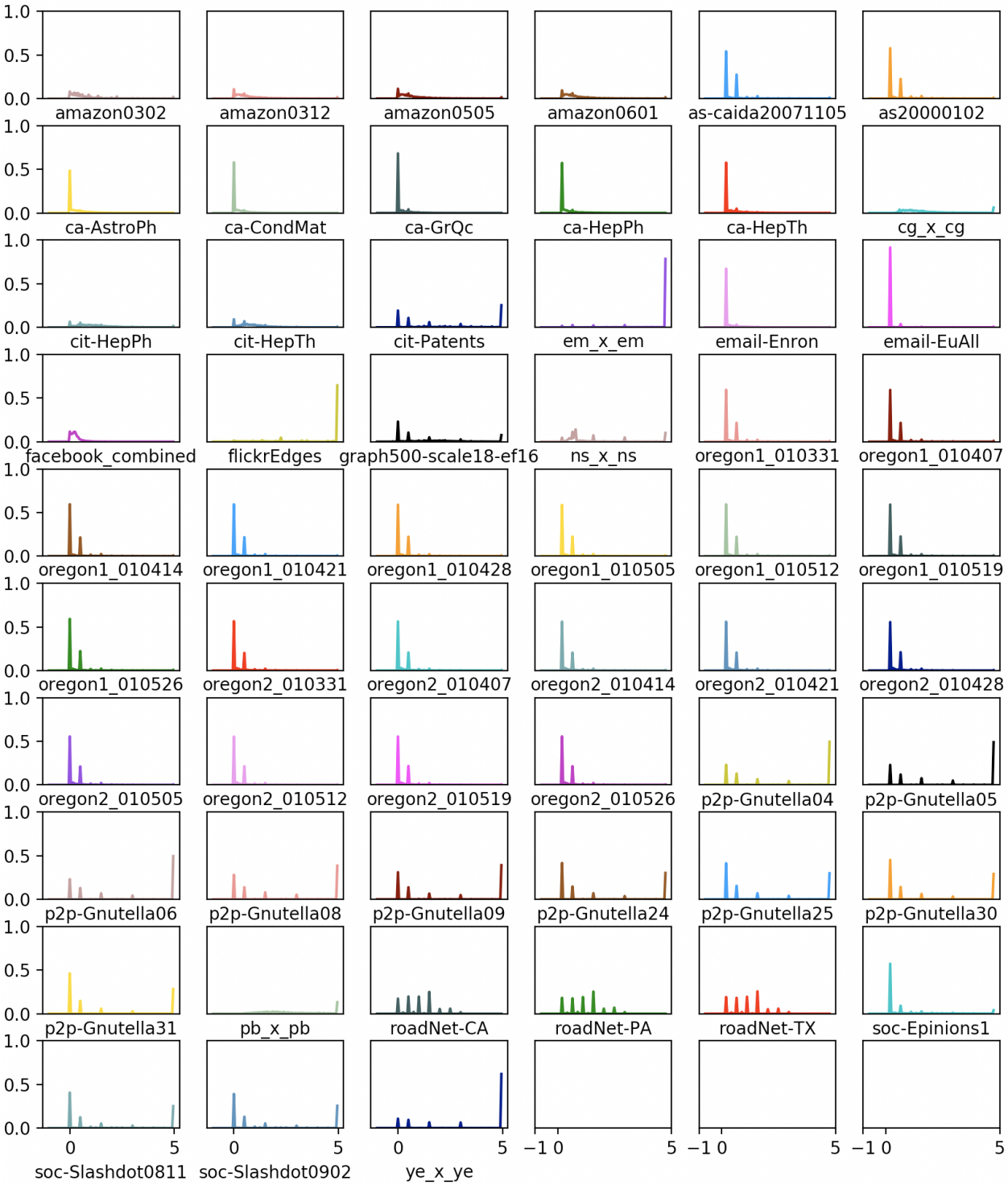


Figure 8: The edge-local relative error of most of the Table **??** using $p = 12$ and the naïve inclusion-exclusion intersection estimator. Note that we have had to widen horizontal axis to account for large overestimates in graphs. Note the degraded performance compared to Figure **??**.

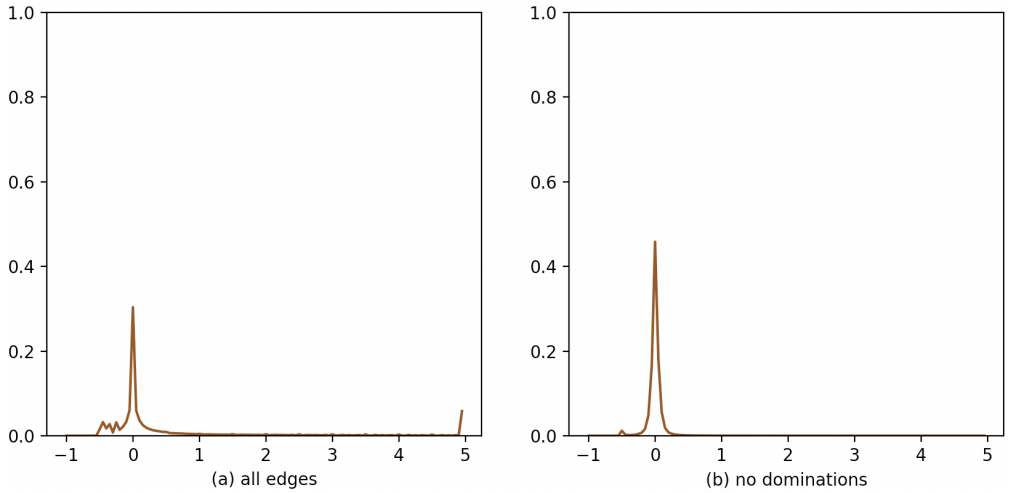
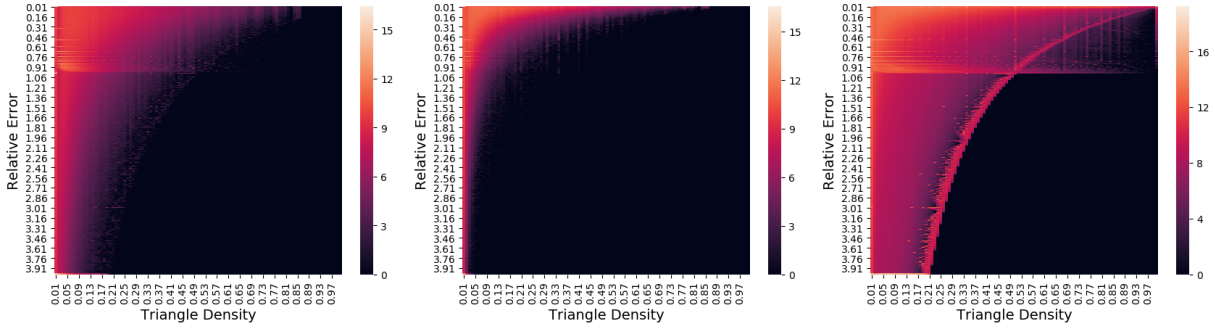


Figure 9: The edge local relative error distribution of the twitter graph (see Table **??**). The distribution over all intersections (a) displays the heavy tail of overestimates. The distribution over only intersections not involving a domination (b) does not exhibit this tail, indicating that minimizing dominations results in improved relative error performance.



(a) em $\otimes$ em, $p = 8$. (b) em $\otimes$ em, $p = 12$. (c) Twitter, $p = 12$.

Figure 10: Heat maps displaying the distribution of edge-local triangle count relative error versus triangle density for the em $\otimes$ em and Twitter graphs. Heat values asigned to cells are the logarithm of the number of edges matching that triangle density and relative error. Note that performance drastically improves for the em $\otimes$ em graph when p increases from 8 to 12.

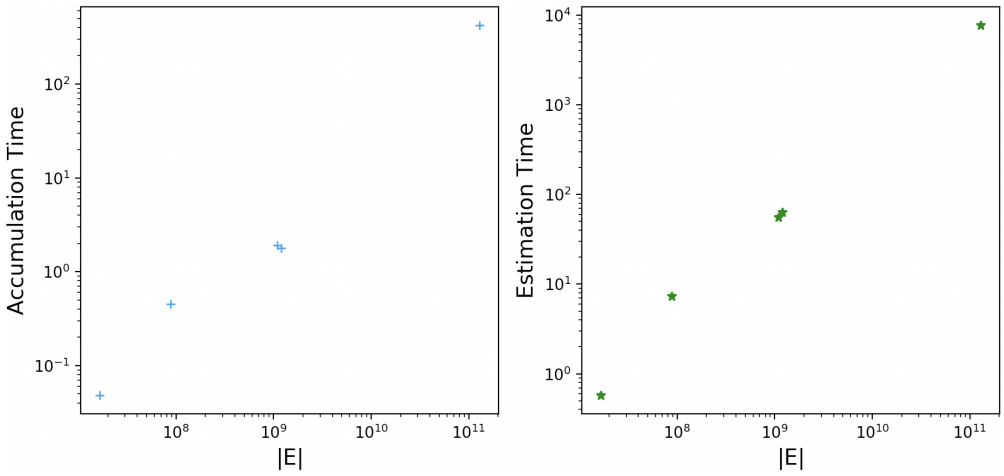


Figure 11: The time in seconds to accumulate and estimate using $N = 72$ compute nodes, each of which have 24 cores, for all graphs listed in Table **??** using $p = 8$. We have ignored the time taken reading from streams and included only the time spend on local computations and communication. This shows that linear scaling in $|\mathcal{E}|$ extends out to truly massive graphs.

operation is performed a large number of times in each experiment. The table plots the global relative error Eq. (**??**), the number of dominations, the edge mean relative error Eq. (**??**), the edge variance, the edge Vigna's $\tau_{h,\phi_{100}}$, the vertex mean relative error Eq. (**??**), the vertex variance, and the vertex Vigna's $\tau_{h,\phi_{100}}$. The Vigna's $\tau_{h,\phi_{100}}$ statistics provide a quantificaton of the heavy hitter recovery performance similar to the precision versus recall, where elements at the very top of the ground truth triangle count are weighted more heavily than those near 100. It also serves as a better general purpose statistic.

The results in Table **??** indicate that our algorithms produce surprisingly good estimates on all edges and vertices, not just heavy hitters, for sufficiently large p. Note that the worst performers in terms of mean relative error are the large synthetic Kronecker graphs $em \otimes em$ and $ye \otimes ye$, which also experience a large volume of dominations. Increasing p, of course, improves performance on these graphs at the expense of increased memory, communication, and computation overhead.

15.2.4 Scaling to Massive Graphs

We have already discussed the challenges in scaling intersections to large graphs. We will now examine the performance scaling of the algorithms as a function of data and computing resource sizes. An advantage of this analysis is that it extends to other applications of DegreeSketch, such as the distributed neighborhood estimation algorithm discussed in Section **??**.

For our scaling experiments we ran Algorithm **??** (accumulation) and Algorithm **??** (vertex local estimation) on each graph in Table **??**. Note that some of these graphs are much larger than those considered in Table **??**. We used $p = 8$ for efficacy, as we are interested in scaling and not concerned with the estimation quality in this case. We discounted the I/O time spent on reading data streams from files.

It is worth noting that a competing state-of-the-art exact triangle counting algorithm required $N = 256$ compute nodes to even load the largest WebDataCommons graph into distributed memory [**?**].

Figure **??** measures the the time in seconds spent accumulating DegreeSketch and performing the vertex-local estimation on each graph, plotted against the number of edges in each graph. We used $N = 72$ compute nodes in each case. As promised, the wall time is linear in the number edges for both accumulation and estimation. Similarly, Figure **??** measures the the time in seconds spent accumulating DegreeSketch and performing the vertex-local estimation on the cit-Patents graph, where N varies from 1 up to 72. The wall time gain of increasing N is linear in this case as well.

Table 2: MLE performance on all graphs, including global MRE, # dominations, MRE and variance and $\tau_{h,\phi_{100}}$ for edges and vertices.

			edges			vertices		
graph name	global MRE	# doms	MRE	variance	$\tau_{h,\phi_{100}}$	MRE	variance	$\tau_{h,\phi_{100}}$
amazon0302	0.000410871	37258	0.00511415	0.00236737	0.948169	0.00617304	0.00174465	0.862339
amazon0312	0.00101454	97923	0.0147273	0.0066181	0.969474	0.0109858	0.00210728	0.910493
amazon0505	0.0045002	105156	0.0157157	0.00815814	0.971034	0.011564	0.00238126	0.9086
amazon0601	0.000363835	101666	0.0147765	0.00686827	0.971244	0.0110941	0.00214676	0.90911
as-caida20071105	0.0329567	1848	0.0592927	0.0292914	0.854042	0.0394628	0.0159984	0.765845
as20000102	0.0321906	425	0.040091	0.0139573	0.694214	0.0271399	0.00551577	0.764055
ca-AstroPh	0.0020574	8326	0.0183591	0.00194955	0.915358	0.00705243	0.0014227	0.678024
ca-CondMat	0.00014373	3740	0.00790223	0.00135315	0.870845	0.00701574	0.00271152	0.720792
ca-GrQc	0.00554309	500	0.00440433	0.000970238	0.506617	0.00647719	0.00275309	0.678824
ca-HepPh	0.00248209	4927	0.0107195	0.00133045	0.899939	0.00842168	0.00227844	0.684348
ca-HepTh	0.00359288	1112	0.00499942	0.00117584	0.788572	0.00820159	0.0037099	0.771194
cg ⊗ cg	0.00450143	80556	0.180783	0.059384	0.979046	0.115359	1.10753	0.0957111
cit-HepPh	0.000491567	17143	0.0383193	0.00859173	0.930214	0.0150293	0.00176638	0.717774
cit-HepTh	3.17927e-05	16412	0.0389419	0.00744493	0.929648	0.0137159	0.00178527	0.764128
cit-Patents	0.00780909	692085	0.0238169	0.0134303	0.98958	0.0183164	0.0102654	0.97152
em ⊗ em	0.0269762	618030	0.361598	0.694572	0.970543	1.39935	29.4262	0.935464
email-Enron	0.00237792	8728	0.0308221	0.00570846	0.904239	0.0125266	0.00314965	0.791837
email-EuAll	0.0071636	10901	0.0438659	0.0250123	0.925389	0.00995284	0.00762299	0.91131
facebook_combined	0.00280804	3362	0.0115543	0.000545316	0.834189	0.00737009	0.000233799	0.536985
flickrEdges	0.00341508	102572	0.0471881	0.0218786	0.983674	0.195658	32.6787	0.730175
graph500-scale18-ef16	0.0165722	160923	0.138074	0.0367471	0.977851	0.133161	0.0554794	0.230933
ns ⊗ ns	0.000667794	155908	0.0160818	0.00657722	0.953689	0.0199951	0.0686846	0.90299
oregon1_010331	0.0120226	973	0.0374072	0.0105397	0.81614	0.0283393	0.00623208	0.761701
oregon1_010407	0.00341295	923	0.0392721	0.0110109	0.800534	0.0297942	0.00548587	0.736287
oregon1_010414	0.0394825	1577	0.0413332	0.0159812	0.82125	0.0302528	0.00780099	0.751239
oregon1_010421	0.0119283	725	0.0397194	0.0180728	0.800203	0.0284574	0.00844028	0.758715
oregon1_010428	0.0178996	500	0.0450234	0.0149815	0.810885	0.0323162	0.00681453	0.733992
oregon1_010505	0.00514401	635	0.0374775	0.0133416	0.815531	0.0262984	0.00580777	0.764276
oregon1_010512	0.00372097	1369	0.0363218	0.0100373	0.812308	0.0266561	0.00477533	0.787779
oregon1_010519	0.00167728	839	0.0544582	0.0216188	0.805998	0.040216	0.0112947	0.756786
oregon1_010526	0.0257453	779	0.0394423	0.0163412	0.790612	0.0278942	0.00708625	0.77016
oregon2_010331	0.00215726	1065	0.0313933	0.0110602	0.814389	0.0265301	0.00632335	0.732991
oregon2_010407	0.0102734	1798	0.0399965	0.0124726	0.812382	0.0374412	0.0079474	0.764744
oregon2_010414	0.00214211	1139	0.0348277	0.00999769	0.828402	0.0317464	0.00610695	0.741403
oregon2_010421	0.00696287	1697	0.0349412	0.0116954	0.807914	0.0297692	0.00620846	0.763229
oregon2_010428	0.000930958	1518	0.0378781	0.0110547	0.829194	0.0321114	0.00601949	0.775626
oregon2_010505	0.00920473	1016	0.0361535	0.0133905	0.82981	0.0288103	0.00774782	0.78395
oregon2_010512	0.000384439	1312	0.0425334	0.0146203	0.801557	0.0366068	0.00780288	0.787347
oregon2_010519	0.00208633	1194	0.0378019	0.0107673	0.839299	0.0333108	0.00662257	0.754128
oregon2_010526	0.00106532	1473	0.0514426	0.0174551	0.835291	0.0515479	0.0123812	0.769886
p2p-Gnutella04	0.148558	1555	0.0177248	0.0168244	0.695298	0.0489285	0.111457	0.762134
p2p-Gnutella05	0.121239	1310	0.0151124	0.0137138	0.808389	0.034442	0.0241619	0.728775
p2p-Gnutella06	0.063464	1191	0.0146142	0.0124697	0.802282	0.0345433	0.0267341	0.66756
p2p-Gnutella08	0.0728463	754	0.0174324	0.0119514	0.752416	0.0241876	0.0191441	0.6
p2p-Gnutella09	0.135828	1110	0.0200961	0.0146221	0.804122	0.0272512	0.0165108	0.6878
p2p-Gnutella24	0.219074	2730	0.0109942	0.0103753	0.819203	0.0219898	0.0254313	0.859488
p2p-Gnutella25	0.291334	2322	0.00901289	0.00842632	0.758603	0.0191121	0.0239306	0.820848
p2p-Gnutella30	0.173799	3829	0.0107415	0.0101481	0.851817	0.0216437	0.0322464	0.808806
p2p-Gnutella31	0.228864	6211	0.0102196	0.00964274	0.886073	0.0204559	0.020475	0.767571
pb ⊗ pb	0.0142477	4671	0.0801485	0.0248473	0.858059	0.0969235	0.6582	0.569607
roadNet-CA	0.0059839	113001	0.000806493	0.000731067	0.567993	0.00130683	0.00097556	0.985063
roadNet-PA	0.00545083	63207	0.000781687	0.000702402	0.590963	0.00126972	0.000967292	0.99904
roadNet-TX	0.0053398	78363	0.000763467	0.000690403	0.579244	0.00126706	0.000940359	1.0
soc-Epinions1	0.0041685	19872	0.0823205	0.025062	0.927086	0.0256688	0.0086639	0.841355
soc-Slashdot0811	0.0219384	19301	0.152505	0.0902361	0.937038	0.0713931	0.044451	0.855897
soc-Slashdot0902	0.0153844	20220	0.145098	0.0816233	0.949649	0.0664616	0.0370551	0.890315
ye ⊗ ye	0.128437	926066	0.381506	0.930024	0.991193	2.15683	110.979	0.943107

Table 3: Scaling Graphs

graph	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	Type
patents	3,774,768	16,518,947	Citation
ye $\otimes$ **ye**	5,574,320	88,338,632	Kronecker
or $\otimes$ **or**	131,859,288	1,095,962,562	Kronecker
Twitter	41,652,224	1,201,045,942	Soc. Net. [**?**]
WebDataCommons	3,563,602,788	128,736,914,864	Web

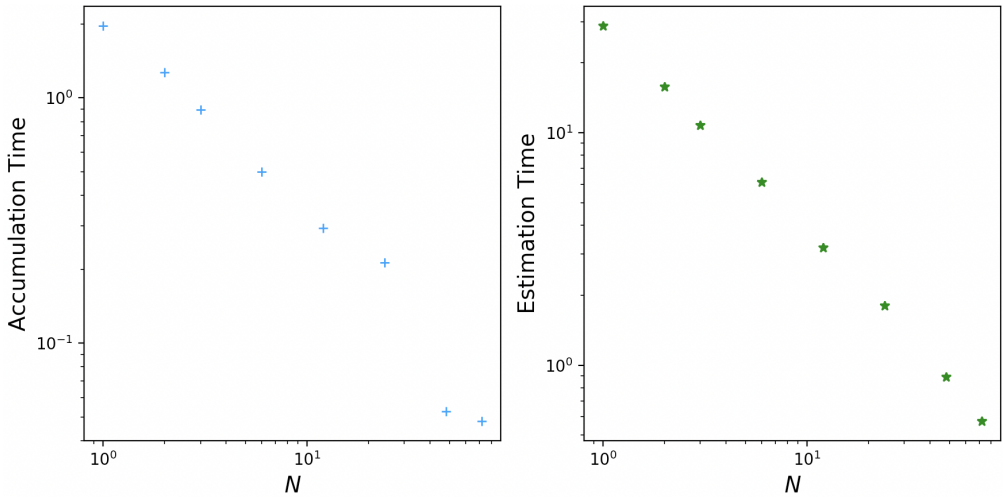


Figure 12: The time in seconds to accumulate and estimate using $N = 1$ up to 72 compute nodes, each of which have 24 cores, for the citation-Patents graph using $p = 8$. We have ignored the time taken reading from streams and included only the time spend on local computations and communication. This shows inverse linear scaling as N increases and the workload remains the same.

References

[ADWR17] Nesreen K Ahmed, Nick Duffield, Theodore L Willke, and Ryan A Rossi. On sampling from massive graph streams. *Proceedings of the VLDB Endowment*, 10(11):1430–1441, 2017.

[AKM13] 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.

[AMS99] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58:137–147, 1999.

[App] Austin Appleby. MurmurHash3. https://github.com/aappleby/smhasher/blob/master/src/MurmurHash3.cpp. Accessed: 2018-12-20.

[BBCG08] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 16–24. ACM, 2008.

[BBCG10] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient algorithms for large-scale local triangle counting. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4(3):13, 2010.

[BHLP11] Jonathan W Berry, Bruce Hendrickson, Randall A LaViolette, and Cynthia A Phillips. Tolerating the community detection resolution limit with edge weighting. *Physical Review E*, 83(5):056119, 2011.

[BJEA17] Sabri Boughorbel, Fethi Jarray, and Mohammed El-Anbari. Optimal classifier for imbalanced data using matthews correlation coefficient metric. *PloS one*, 12(6):e0177678, 2017.

[BRV11] Paolo Boldi, Marco Rosa, and Sebastiano Vigna. Hyperanf: Approximating the neighbourhood function of very large graphs on a budget. In *Proceedings of the 20th international conference on World wide web*, pages 625–634. ACM, 2011.

[BYJK+02] Ziv Bar-Yossef, TS Jayram, Ravi Kumar, D Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 1–10. Springer, 2002.

[BYKS02] Ziv Bar-Yossef, Ravi Kumar, and D Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 623–632. Society for Industrial and Applied Mathematics, 2002.

[CC11] Shumo Chu and James Cheng. Triangle listing in massive networks and its applications. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 672–680. ACM, 2011.

[CCFC02] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.

[CKY17] 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.

[CM05] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[Coh08] Jonathan Cohen. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report*, 16, 2008.

[Col] Yann Collet. xxHash. https://github.com/Cyan4973/xxHash. Accessed: 2018-12-20.

[DF03] Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities. In *European Symposium on Algorithms*, pages 605–617. Springer, 2003.

[DH11] Timothy A Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1, 2011.

[Ert17] Otmar Ertl. New cardinality estimation algorithms for hyperloglog sketches. *arXiv preprint arXiv:1702.01284*, 2017.

[EVF03] Cristian Estan, George Varghese, and Mike Fisk. Bitmap algorithms for counting active flows on high speed links. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 153–166. ACM, 2003.

[FFGM07] 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.

[FKM+05] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2-3):207–216, 2005.

[FM85] Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

[GGL+13] Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. Wtf: The who to follow service at twitter. In *Proceedings of the 22nd international conference on World Wide Web*, pages 505–514. ACM, 2013.

[Gir09] Frédéric Giroire. Order statistics and estimating cardinalities of massive data sets. *Discrete Applied Mathematics*, 157(2):406–427, 2009.

[HNH13] 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.

[JS98] Philippe Jacquet and Wojciech Szpankowski. Analytical depoissonization and its applications. *Theoretical Computer Science*, 201(1-2):1–62, 1998.

[JSP13] Madhav Jha, Comandur Seshadhri, and Ali Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 589–597. ACM, 2013.

[Ken38] Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[KKM+16] Chanhyun Kang, Sarit Kraus, Cristian Molinaro, Francesca Spezzano, and VS Subrahmanian. Diffusion centrality: A paradigm to maximize spread in social networks. *Artificial Intelligence*, 239:70–96, 2016.

[KNW10] Daniel M Kane, Jelani Nelson, and David P Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 41–52. ACM, 2010.

[KSA+18] Jeremy Kepner, Siddharth Samsi, William Arcand, David Bestor, Bill Bergeron, Tim Davis, Vijay Gadepally, Michael Houle, Matthew Hubbell, Hayden Jananthan, et al. Design, generation, and validation of extreme scale power-law graphs. *arXiv preprint arXiv:1803.01281*, 2018.

[Kun13] Jérôme Kunegis. Konect: the koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 1343–1350. ACM, 2013.

[Lan17] Kevin J Lang. Back to the future: an even more nearly optimal cardinality estimation algorithm. *arXiv preprint arXiv:1708.06839*, 2017.

[LCK+10] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. Kronecker graphs: An approach to modeling networks. *Journal of Machine Learning Research*, 11(Feb):985–1042, 2010.

[LK14] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[LK15] 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.

[M+05] Shanmugavelayutham Muthukrishnan et al. Data streams: Algorithms and applications. *Foundations and Trends® in Theoretical Computer Science*, 1(2):117–236, 2005.

[M+11] Michael W Mahoney et al. Randomized algorithms for matrices and data. *Foundations and Trends® in Machine Learning*, 3(2):123–224, 2011.

[McG09] Andrew McGregor. Graph mining on streams. In *Encyclopedia of Database Systems*, pages 1271–1275. Springer, 2009.

[MSGL14] Seth A Myers, Aneesh Sharma, Pankaj Gupta, and Jimmy Lin. Information network or social network?: the structure of the twitter follow graph. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 493–498. ACM, 2014.

[MSOI+02] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

[Pea17] 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.

[PGF02] Christopher R Palmer, Phillip B Gibbons, and Christos Faloutsos. Anf: A fast and scalable tool for data mining in massive graphs. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 81–90. ACM, 2002.

[PPS18] 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.

[QKT16] 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.

[SERU17] 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.

[SHL+18] 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.

[SLO+18] 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.

[SPLFK18] Geoffrey Sanders, Roger Pearce, Timothy La Fond, and Jeremy Kepner. On large-scale graph generation with validation of diverse triangle statistics at edges and vertices. *arXiv preprint arXiv:1803.09021*, 2018.

[SV11] Siddharth Suri and Sergei Vassilvitskii. Counting triangles and the curse of the last reducer. In *Proceedings of the 20th international conference on World wide web*, pages 607–614. ACM, 2011.

[Tin16] 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.

[TKMF09] Charalampos E Tsourakakis, U Kang, Gary L Miller, and Christos Faloutsos. Doulion: counting triangles in massive graphs with a coin. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 837–846. ACM, 2009.

[Tso08] Charalampos E Tsourakakis. Fast counting of triangles in large real networks without counting: Algorithms and laws. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 608–617. IEEE, 2008.

[Vig15] Sebastiano Vigna. A weighted correlation index for rankings with ties. In *Proceedings of the 24th international conference on World Wide Web*, pages 1166–1176. International World Wide Web Conferences Steering Committee, 2015.

[WDB+17] Michael M Wolf, Mehmet Deveci, Jonathan W Berry, Simon D Hammond, and Sivasankaran Rajamanickam. Fast linear algebra-based triangle counting with kokkoskernels. In *High Performance Extreme Computing Conference (HPEC), 2017 IEEE*, pages 1–7. IEEE, 2017.

[Wei62] Paul M Weichsel. The kronecker product of graphs. *Proceedings of the American mathematical society*, 13(1):47–52, 1962.

[WF94] Stanley Wasserman and Katherine Faust. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

[WZTT10] Nan Wang, Jingbo Zhang, Kian-Lee Tan, and Anthony KH Tung. On triangulation-based dense neighborhood graph discovery. *Proceedings of the VLDB Endowment*, 4(2):58–68, 2010.

[XZC17] 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.