Distributed Data Summarization in Well-Connected Networks

Hsin-Hao Su
Boston College, MA, USA
suhx@bc.edu

Hoa T. Vu
Boston College, MA, USA
vuhd@bc.edu

Abstract

We study distributed algorithms for some fundamental problems in data summarization. Given a communication graph G of n nodes each of which may hold a value initially, we focus on computing $\sum_{i=1}^{N} g(f_i)$, where f_i is the number of occurrences of value i and g is some fixed function. This includes important statistics such as the number of distinct elements, frequency moments, and the empirical entropy of the data.

In the CONGEST model, a simple adaptation from streaming lower bounds shows that it requires $\tilde{\Omega}(D+n)$ rounds, where D is the diameter of the graph, to compute some of these statistics exactly. However, these lower bounds do not hold for graphs that are well-connected. We give an algorithm that computes $\sum_{i=1}^{N} g(f_i)$ exactly in $\tau_G \cdot 2^{O(\sqrt{\log n})}$ rounds where τ_G is the mixing time of G. This also has applications in computing the top k most frequent elements.

We demonstrate that there is a high similarity between the GOSSIP model and the CONGEST model in well-connected graphs. In particular, we show that each round of the GOSSIP model can be simulated almost perfectly in $\tilde{O}(\tau_G)$ rounds of the CONGEST model. To this end, we develop a new algorithm for the GOSSIP model that $1 \pm \epsilon$ approximates the p-th frequency moment $F_p = \sum_{i=1}^{N} f_i^p$ in $\tilde{O}(\epsilon^{-2} n^{1-k/p})$ rounds [1], for $p \geq 2$, when the number of distinct elements F_0 is at most $O\left(n^{1/(k-1)}\right)$. This result can be translated back to the CONGEST model with a factor $\tilde{O}(\tau_G)$ blow-up in the number of rounds.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Networks → Network algorithms; Mathematics of computing → Graph algorithms

Keywords and phrases Distributed Algorithms, Network Algorithms, Data Summarization

Digital Object Identifier 10.4230/LIPIcs.DISC.2019.34

Related Version A full version of the paper is available at `https://arxiv.org/abs/1908.00236`.

1 Introduction

Motivation. Analyzing massive datasets has become an increasingly important and challenging problem. Collecting the entire data to one single machine is usually infeasible due to memory, I/O, or network bandwidth constraints. Furthermore, in many cases, data are distributed over the network and we hope to aggregate some of their properties efficiently. In this work, we consider several fundamental data summarization problems in distributed networks, specifically in the CONGEST and GOSSIP models.

In this problem, we have a graph $G = (V, E)$ of n nodes. Each node v in the graph may hold a value $\text{val}(v)$ in the range $\{1, \ldots, N\} \cup \{\text{NULL}\}$ where NULL simply means that the node does not hold a value. If $\text{val}(v) = \text{NULL}$, we call v an *empty node*.

[1] $\tilde{O}$ omits polylog(n) factors.

We often use the notation $[N] := \{1, \ldots, N\}$. Let f_i be the number of nodes that hold value i, i.e., $f_i = |\{v \in V : \mathrm{val}\,(v) = i\}|$. We want to compute $\sum_{i=1}^{N} g(f_i)$ for some fixed function g. To demonstrate some important cases, consider the following examples.

Consider $g(f_i) = 1$ if $f_i > 0$ and 0 otherwise. This corresponds to the problem of counting the number of distinct elements (or computing the 0-th frequency moment F_0). The problem may arise in the following situation: Each node stores a version of a file (e.g. the hash of a blockchain), and we want to know how many different versions there are in the network.

If $g(f_i) = f_i^p$ for some fixed $p = 2, 3, \ldots$, then this corresponds to the problem of computing the p-th frequency moment F_p. We note that F_p is a basic, yet very important statistic of a dataset. F_2 measures the variance and could be used to estimate the size of a self-join in database applications. For higher p, F_p measures the skewness of the dataset (see [2]). Note that F_1 can be computed in $O(D)$ rounds in the CONGEST model by aggregating along a breath-first-search (BFS) tree (in the GOSSIP model F_1 can be computed exactly in $O(\log n)$ rounds).

Another example is $g(f_i) = -(f_i/F_1) \cdot \log(f_i/F_1)$. In this case, the sum is the empirical entropy of the data. Computing the empirical entropy is motivated by network applications such as detecting anomalies [20, 40, 42].

Models. We now give a formal description of the CONGEST and GOSSIP models, where the running time of an algorithm is measured by the number of rounds.

▶ **Definition 1.** *In the* CONGEST *model, we are given a graph $G = (V, E)$ of n nodes, in each synchronous round, each node can talk (send and receive message) to each of its neighbors and then perform local computations. Each message is restricted to be at most $O(\log n)$ bits.*

▶ **Definition 2.** *In the* GOSSIP(λ) *model with n nodes, in each synchronous round, each node u samples a node $t(u)$ from a distribution that satisfies the following: For any node v and any subset of nodes Z where $u \notin Z$,*

$$\Pr\left(t(u) = v \,\middle|\, \bigwedge_{z \in Z} t(z)\right) \in \left[\frac{1-\lambda}{n}, \frac{1+\lambda}{n}\right].$$

In the above, "$\bigwedge_{z \in Z} t(z)$" means conditioning on any assignment of each $t(z)$ for $z \in Z$. Then, u can PUSH a message of size $O(\log n)$ to $t(u)$ or PULL a message of size $O(\log n)$ from $t(u)$. Then, after performing some local computations, it proceeds to the next round. We refer GOSSIP *model as the* GOSSIP(0) *model.*

1.1 Our results

We organize our main results into three categories: a) results in the CONGEST model, b) an emulation of the GOSSIP model in the CONGEST model, and c) results in the GOSSIP model.

Results in the CONGEST model. We briefly show how to adapt streaming algorithms to approximate F_p (for $p = 0, 2, 3, \ldots$) in the CONGEST model. We also demonstrate some lower bounds and conditional lower bounds that give evidence that such algorithms are optimal or near-optimal.

The lower bounds show that computing F_p exactly for $p = 0, 2, 3, \ldots$ requires $\tilde{\Omega}(D + n)$ rounds and approximating F_p within a constant factor requires *polynomial* rounds in n for $p \geq 3$. Roughly speaking, the hard instances in the CONGEST model are graphs with a small

balanced cut of $O(1)$ size that causes an information bottleneck. However, such bottleneck does not occur in graphs that are well-connected. Our first main result aims to answer the following question: *Could one design more efficient algorithms for well-connected graphs?* We give a positive answer to this question.

By using the permutation routing algorithms of Ghaffari et al. [15] (later improved by Ghaffari and Li [16]), we show that there exists an algorithm running in $\tau_G \cdot 2^{O(\sqrt{\log n})}$ rounds that computes $\sum_{i=1}^{N} g(f_i)$ for all fixed and computable functions g with high probability (w.h.p.) [2]. This includes all the aforementioned quantities such as the number of distinct elements, higher frequency moments, and the empirical entropy. Thus, if the graph has small mixing time such as expanders [19, 23], where $\tau_G = \text{polylog}(n)$, then we obtain a much more efficient sub-polynomial in n algorithm compared to the adaptation of the streaming counterpart.

▶ **Theorem 3** (Main result 1). *There exists an algorithm that computes $\sum_{i=1}^{N} g(f_i)$ exactly for all (fixed and computable) functions g in the* CONGEST *model in $\tau_G \cdot 2^{O(\sqrt{\log n})}$ rounds w.h.p.*

Our algorithm can also easily be extended to find the top k frequent elements in $O(k) + \tau_G \cdot 2^{O(\sqrt{\log n})}$ rounds.

From CONGEST to GOSSIP. The lower bounds do not apply directly to the GOSSIP model either. This is because for any balanced cut of the nodes, one expects $O(n)$ messages to be sent across in one round. Moreover, the expected communication degree per node in the GOSSIP model is $O(1)$. Intuitively, the graph formed by the communication pattern in the GOSSIP model is similar to an expander graph.

In fact, we show that well-connected graphs can emulate the GOSSIP model efficiently. In particular, one round of the GOSSIP $(1/\text{poly}(n))$ model can be emulated in $\tau_G \cdot \text{polylog}(n)$ rounds in the CONGEST model where the underlying graph is G. Therefore, any algorithm that works in the GOSSIP $(1/\text{poly}(n))$ model can be turned into an algorithm in the CONGEST model with an $\tilde{O}(\tau_G)$ factor blow-up.

Consider our results in the CONGEST model. The permutation routing algorithms of [15] and [16] introduce a super-logarithmic factor, $2^{O(\sqrt{\log n})}$, on top of the mixing time. It becomes the bottleneck in graphs with small mixing times (e.g., expanders). Improving the permutation routing algorithm directly yields improvements to our results in the CONGEST model (and many other problems). However, it is unclear if it can be improved. This emulation result serves as an alternative route to circumvent the $2^{O(\sqrt{\log n})}$ factor, if one develops efficient GOSSIP algorithms.

▶ **Theorem 4** (Main result 2). *For $\lambda = 1/\text{poly}(n)$, one round of the* GOSSIP(λ) *model can be emulated in $\tilde{O}(\tau_G)$ rounds in the* CONGEST *model where G is the connected graph denoting the underlying network.*

We believe that this emulation result may be of independent interest. Jelasity et al. [27] studied how to implement the gossip-based peer sampling service empirically. Our result is an additional way to implement the service with theoretical guarantees.

[2] We consider $1 - 1/\text{poly}(n)$ as high probability.

Table 1 Results summary for computing frequency moments F_p. (*) can also be used to compute $\sum_i g(f_i)$ for all fixed and computable functions g.

	Number of rounds	Assumption	Approximation
CONGEST	$\tau_G \cdot 2^{O(\sqrt{\log n})}$ (*)		Exact
	$O(\epsilon^{-2}\tau_G \cdot \text{polylog}\, n)$	$F_0 \leq O(n^{1/(p-1)})$	$1 \pm \epsilon$
GOSSIP	$O(\epsilon^{-2} \cdot n^{1-k/p} \cdot \text{polylog}\, n)$	$F_0 \leq O(n^{1/(k-1)})$	$1 \pm \epsilon$

Results in the GOSSIP model. Motivated by our emulation result, we develop algorithms for the GOSSIP model. In particular, we are interested in the following question: *Suppose the number of non-empty nodes are sublinear in n. Could we take advantage of the computational power of the empty nodes?*

Suppose that the number of non-empty nodes is at most $O(n^{1/(k-1)})$ (or more generally, $F_0 \leq O(n^{1/(k-1)})$). We show that for any $p \geq 2$, F_p can be approximated within a $1 \pm \epsilon$ factor in $O(\epsilon^{-2} n^{1-k/p} \log^2 n)$ rounds with high probability.

▶ **Theorem 5** (Main result 3). *If $F_0 = O(n^{1/(k-1)})$ for some integer $2 \leq k \leq p$, then there exists an algorithm that approximates F_p up to a $1 \pm \epsilon$ factor in $O(\epsilon^{-2} n^{1-k/p} \log^2 n)$ rounds in the* GOSSIP *$(1/n^c)$ model, for some sufficiently large constant c, w.h.p.*

The GOSSIP $(1/n^c)$ model will incur a $\pm 1/\text{poly}(n)$ additive error which we consider insignificant. Since $F_0 \leq n$, we have an algorithm that approximates F_2 in $\tilde{O}(\epsilon^{-2})$ rounds by setting $k = 2$. When $k > 2$, the empty nodes serve as the extra computation power to solve the problem. In such scenarios, we are able to obtain running time that is not known to be achievable by adapting the streaming counterpart. For example, when $k = 3$, $F_0 = O(n^{1/2})$, we may approximate F_3 within a constant factor in $\text{polylog}(n)$ rounds. Direct adaption of known streaming algorithms [2, 3, 36] requires super-logarithmic rounds, even in the case where $F_0 = O(n^{1/2})$.

Combining Theorem 5 and Theorem 4 with $k = p$, we have the following corollary.

▶ **Corollary 6.** *If $F_0 = O(n^{1/(p-1)})$, then there exists an algorithm in the* CONGEST *model that approximates F_p up to a $1 \pm \epsilon$ factor in $\tilde{O}(\epsilon^{-2} \cdot \tau_G)$ rounds w.h.p.*

1.2 Related work and preliminaries

Related work. In the distributed setting, Kuhn et al. [33] studied the problem of finding the mode, i.e., the most frequent element, in the CONGEST model. Let D is the diameter of the graph, and f^* is the largest number of occurrences among the values. They gave an algorithm that uses $O(D + F_2/f^* \cdot \log F_0)$ rounds. They also briefly explained how to implement streaming algorithms for approximating F_0 and F_2 in the CONGEST models. Also related to data summarization, Kuhn et al. [34] designed selection algorithms in the CONGEST model.

In the data stream model, each stream token (i, x) corresponds to the update $f_i \leftarrow f_i + x$. The problem of approximating the number of distinct elements F_0 and frequency moments F_p have been extensively studied. An incomplete list includes [2–4, 14, 18, 24, 25, 29, 41]. Roughly speaking, the space complexity for approximating F_p in the data stream model is $\tilde{O}(\epsilon^{-2})$ for $0 \leq p \leq 2$ and $\tilde{O}(\epsilon^{-2} n^{1-2/p})$ for $p \geq 2$. Furthermore, it is known that approximating F_∞ (or identifying the mode) is not possible in sublinear space. In the data stream model, researchers have also studied the problem of approximating the entropy [9, 10, 22].

We will briefly discuss the similarities between the data stream model and the CONGEST model. Roughly speaking, since streaming algorithms use little memory, they can be adapted to the CONGEST model by passing the memory state of the corresponding algorithm along the breadth-first-search tree. Similarly, lower bounds from streaming algorithms literature can also be translated into lower bounds in the CONGEST model. Data aggregation problems have also been studied in directed networks [35].

There is also a rich literature in the GOSSIP model started by the work of [12]. Some examples include spreading message [13, 30, 38], computing the sum and average [11, 31, 32], renaming [17], and quantile computation [21].

Preliminaries. We introduce basic notations and algorithmic building blocks in the CONGEST model.

To ease our presentation, we assume $N = O(\text{poly}(n))$. In our algorithms, we often want to learn about the sum of all the values (or hash values, indicator variables) held by the nodes; this can be done in $O(D)$ rounds. Another algorithmic primitive, based on downcasts and upcasts, is to broadcast the k smallest values in $O(D + k)$ rounds.

We define the mixing time similarly to [15]. A lazy random walk is a random walk in which at each step, we stay at the same node with probability 0.5 and move to a random neighbor with probability 0.5. Lazy random walk ensures the existence of a unique stationary distribution (i.e., the walk is aperiodic). From now on, we simply refer to a lazy random walk as a random walk.

Let $P_u^t = (P_u^t(v_1), \ldots, P_u^t(v_n)) \in [0,1]^n$ denotes the probability distribution on the nodes after t steps of a lazy random walk that starts at u. A crucial property of a random walk is that it will converge to the stationary distribution $(\deg(v_1)/2m, \ldots, \deg(v_n)/2m)$. Define the mixing time τ_G to be the minimum t such that for any starting node u and any node v_i,

$$\left| P_u^t(v_i) - \frac{\deg(v_i)}{2m} \right| \leq \frac{\deg(v_i)}{2mn} .$$

Using an $O(D)$-round pre-proscessing, we can assume that each node has a unique ID in $[n]$. Suppose we want the nodes in a graph to have unique IDs in $[n]$. We can elect a leader and build a breadth-first-search (BFS) tree that is rooted at the leader in $O(D)$ rounds [37]. Each node u can learn about the number of nodes in T_v where v is a child of u and T_v is the subtree that is rooted at v. This is done by aggregating the size from the leaves upward. It is then straightforward to assign the IDs to the nodes based on the depth-first-search (DFS) ordering. Specifically, the root notifies each of its children v the range of the IDs in T_v, based on the DFS ordering, and then recurse on T_v. From now on, we can refer to the nodes by their IDs, i.e., $\text{ID}(v) = v$.

We will also make use of hash functions. An $O(1)$-wise independent hash function $h : [a] \to [b]$ where a and b are at most $\text{poly}(n)$ can be stored in $O(\log n)$ bits. Hence, if we need to use a hash function, a leader can broadcast such hash function (using a BFS tree) in $O(D + \log n)$ rounds in the CONGEST model and $O(\log n)$ rounds in the GOSSIP model.

2 Algorithms in the CONGEST Model

2.1 Approximation algorithms

Upper bounds. We show that we can adapt the streaming algorithms given by Bar-Yossef et al. [4] (for approximating F_0) and by Alon et al. [2] (for approximating F_p, where $p \geq 2$) to the CONGEST model (see the appendix in our full version [39]). This is not of particular

novelty though we need some careful pipelining arguments to optimize the number of rounds. Kuhn et al. [33] also briefly outlined similar results. However, the exact round-complexity for a good approximation w.h.p. is not very clear from their paper.

▶ **Theorem 7.** *There exists an $O(D + \epsilon^{-2} \log n)$-round algorithm in the* CONGEST *model that computes a $1 \pm \epsilon$ approximation of F_0 and F_2 w.h.p. Furthermore, for $p > 2$, there exists an $O(D + \epsilon^{-2} \min(n, N)^{1-1/p} \log n)$-round algorithm that computes a $1 \pm \epsilon$ approximation of F_p w.h.p.*

Lower bounds. We show that the dependence on ϵ is tight via a conditional lower bound. Moreover, computing F_p exactly requires $\tilde{\Omega}(n)$ rounds. The lower bounds are obtained by adapting the existing streaming lower bounds to the CONGEST model. Due to space constraint, we refer to the appendix in our full version for the discussion.

▶ **Theorem 8.** *We have the following lower bounds in the* CONGEST *model.*

- *If the conjecture in [8] holds, then approximating F_p (for fixed $p \neq 1$) up to a $1 \pm \epsilon$ factor requires $\Omega(D + \epsilon^{-2}/\log n)$ rounds.*
- *A (1 ± 0.1)-approximation of F_p, for $p > 2$, requires $\Omega\left(D + \left(N^{1-\frac{2}{p}} + n^{\frac{1-2/p}{1+1/p}}\right)/\log n\right)$ rounds.*
- *Computing F_p exactly requires $\Omega(D + n/\log n)$ rounds.*

Hence, we cannot expect a sublinear algorithms (in terms of N, n) when $\epsilon \ll 1/\sqrt{n}$ or when we want to obtain the exact answer. The lower bounds arise in graphs with a small balanced cut which causes an information bottleneck. This observation motivates us to design an exact algorithm when the graph is well-connected.

2.2 An exact algorithm in near mixing-time

In this subsection, we show that it is possible to beat the lower bounds and achieve an exact algorithm in sublinear time if the graph has fast mixing time. For example, expander graphs are sparse and have $O(\text{polylog}\, n)$ mixing time.

Suppose each node has a set of messages (of size $\text{polylog}(n)$) each of which has a destination that is another node. In parts of our algorithms, we want to route messages in a small number of rounds. We rely on the following routing algorithm in the CONGEST model that uses $\tau_G \cdot 2^{O(\sqrt{\log n})}$ rounds. We note that $2^{O(\sqrt{\log n})}$ is more than $\text{polylog}\, n$ but smaller than any n^ϵ for $\epsilon > 0$. Also note that $D = O(\tau_G)$. Let $\deg(v)$ be the degree of v in G.

▶ **Theorem 9** ([16], [15]). *If each node of G is the source and the destination of at most $\deg(v) \cdot 2^{O(\sqrt{\log n})}$ messages, then there is a randomized algorithm in the* CONGEST *model that delivers all the messages in $\tau_G \cdot 2^{O(\sqrt{\log n})}$ rounds w.h.p.*

We also rely on the idea of sorting networks. Recall that we refer to the nodes by their unique IDs in $[n]$. In a sorting network, in each step r, the sorting network will pick a set of disjoint pairs of nodes. We use $\text{val}(x, r)$ to denote the value that node x holds in the beginning of step r. For each pair x and y (where $x < y$) that is picked, x will keep the smaller value $\min(\text{val}(x, r), \text{val}(y, r))$ and y will keep the larger value $\max(\text{val}(x, r), \text{val}(y, r))$. We treat NULL as $-\infty$. The sorting network can be constructed, solely based on n, so that after $t = O(\log n)$ steps, the values are sorted [1]. That is if $x < y$, then $\text{val}(x, t) \leq \text{val}(y, t)$.

In the CONGEST model, each node can generate the sorting network (note that the construction of the sorting network is independent of the topology of G and the values held by the nodes). Furthermore, each step can be simulated by invoking Theorem 9. Thus, we have the following.

▶ **Lemma 10.** *In the* CONGEST *model, we can sort the nodes' values in* $\tau_G \cdot 2^{O(\sqrt{\log n})}$ *rounds w.h.p.*

We now complete the proof of our first main result.

Proof of Theorem 3. We now use $\mathrm{val}(v)$ to refer to the value that v holds after sorting. We say a node v is a *head* or a *tail* if $\mathrm{val}(v) \neq -\infty$ and its ID is the smallest or the largest respectively among the IDs of the nodes that hold the value $\mathrm{val}(v)$. A node v can tell that if it is a head or a tail by checking with the nodes $v+1$ and $v-1$ respectively using the routing algorithm in Theorem 9. We use $\mathrm{head}(i)$ and $\mathrm{tail}(i)$ to denote the IDs of the head and the tail of value i respectively.

Now, every node that is not a head or a tail marks its value as $-\infty$. Each remaining node forms a token consisting of its value, ID, and whether if it is a head or a tail (or both). We then use sorting networks again to sort the values in the graph. We will also swap the tokens if two nodes swap their values. Afterward, the head and the tail tokens of a value i will be at some two nodes v and $v+1$ (or just at a node v if $f_i = 1$). To this end, each node v that holds a head token (that is not also a tail token) with value i will check with nodes $v+1$ and $v-1$, using the routing algorithm, to collect $\mathrm{tail}(i)$ since either $v+1$ or $v-1$ must have the tail token of i. Now, v can compute $g(f_i) = g(\mathrm{tail}(i) - \mathrm{head}(i) + 1)$ and set this as its value. All the nodes that do not hold a head token set their values to 0. We then compute $\sum_{i=1}^{N} g(f_i)$ using the BFS tree in $O(D)$ rounds. ◀

The algorithm above is more robust compared to the AMS sketch since it can handle all fixed and computable functions g. The AMS sketch cannot guarantee sublinear space in the streaming model (or sublinear time in the CONGEST model) for many functions [5–7]. The above algorithm also immediately leads to an algorithm that finds the top k frequent elements.

Finding the top k frequent elements. At the end of the above algorithm, the occurrence of each value i is held by some node v. Recall we can find the top k elements in the graph using $O(D+k)$ rounds via upcasts. This immediately leads to the following result.

▶ **Theorem 11.** *There exists an algorithm that finds the top* k *elements (along with their occurrences) in the* CONGEST *model in* $O(k + \tau_G \cdot 2^{O(\sqrt{\log n})})$ *rounds w.h.p.*

3 Emulation of GOSSIP Model in the CONGEST Model

In Section 2, we have shown that the moments can be computed exactly in $\tau_G \cdot 2^{O(\sqrt{\log n})}$ rounds. If the permutation routing algorithm can be improved to polylog(n) rounds, then the running time of our algorithms would be improved to $\tilde{O}(\tau_G)$ rounds. Whether the $2^{O(\sqrt{\log n})}$ factor can be improved to polylog(n) is an intriguing open question.

Instead of tackling the complexity of permutation routing, in this section, we show that one round of the GOSSIP model can be emulated almost-perfectly in $\tilde{O}(\tau_G)$ rounds in the CONGEST model. Therefore, if there is a polylog(n)-round algorithm in the GOSSIP model, it implies a $\tilde{O}(\tau_G)$ rounds algorithm in the CONGEST model. In Section 4, we present efficient algorithms in the GOSSIP model when F_0 is small (or when the number of empty nodes is large) which can be translated back to the CONGEST model using the emulation result in this section.

Recall that $P_u^t = (P_u^t(v_1), \ldots, P_u^t(v_n)) \in [0,1]^n$ denotes the probability distribution on the nodes after t steps of a lazy random walk that starts at u (see Section 1.2). Given λ, we let $\tau_G(\lambda)$ be the smallest t such that for any starting node u and any node v_i,

$$\left| P_u^t(v_i) - \frac{\deg(v_i)}{2m} \right| \leq \lambda.$$

Note that if $\lambda = 1/\operatorname{poly}(n)$ then $\tau_G(\lambda) = O(\tau_G)$ [15, Definition 2.1].

We will run several random walks in parallel. The following lemma from [15] shows that the parallel random walks can be performed efficiently in the CONGEST model.

▶ **Lemma 12** ([15], Lemma 2.5). *Let $G = (V, E)$ be an n-node graph and let $t \geq 1$ be a positive integer. Assume that we perform $T = O(\operatorname{poly}(n))$ steps of a collection of independent random walks in parallel. If each node $u \in V$ is the starting node of at most $t \cdot \deg(u)$ random walks, w.h.p., the T steps of all the random walks can be performed in $O((t + \log n) \cdot T)$ rounds in the* CONGEST *model.*

The main technical difficulty of the emulation lies in the fact that the stationary distribution is not necessarily uniform in general graphs. If G is regular, we could let each node u start a random walk that runs for $O(\tau_G)$ steps. The probability that u ends at each node is (nearly) uniform. If it ends at v then we set $t(u) = v$. Moreover, by Lemma 12, all the random walks can be performed simultaneously in $\tilde{O}(\tau_G)$ rounds.

In irregular graphs, such approach does not work because the stationary distribution is not uniform. One remedy is to regularize the random walk (i.e. adding self-loops to non-maximum degree nodes). However, this may significantly increase the mixing time of the graph (e.g., a star graph). In the following, we give an emulation algorithm whose running time is within a polylog(n) factor of the mixing time.

For each node u in G, we split it into $\deg(u)$ *compartments*. When a random walk enters a node, it is assigned randomly to one of its compartments. There are $2m$ compartments in G in total. We outline the emulation algorithm below.

1. Let $k = \lfloor 1.5m/n \rfloor$. Each node creates k *destination tokens* and distributes them over the compartments in G so that each compartment contains at most *one* destination token. Now $n \cdot k \approx 1.5m$ compartments are filled with tokens.
2. Each node sends out a *source token*. Each source token starts a random walk to distribute itself randomly over the compartments at the end. If the source token of node u ends in a compartment with the destination token of some node v, we set $t(u) = v$.
3. Route the message between u and $t(u)$ for each u simultaneously.

We explain how to implement each step in details.

Step 1. Each node u creates a destination token (u, k) initially. The first component of the token is its identity while the second component of the token is its multiplicity. The goal is to split the tokens and distribute them across the compartments so that all tokens have multiplicity of 1 and each compartment holds at most one token. We divide Step 1 into the **splitting phase** and the **distributing phase**.

The splitting phase is further divided into $\lceil \log k \rceil$ stages. At the beginning of each stage, if $W > 1$, each token (u, W) is split into two tokens $(u, \lceil W/2 \rceil)$ and $(u, \lfloor W/2 \rfloor)$. Then all tokens perform τ_G steps of random walks.

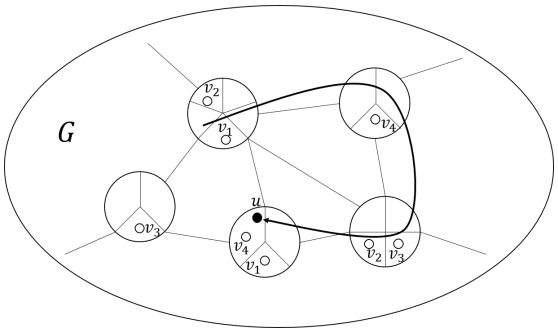

Figure 1 Illustration of Step 2. The random walk of u's source token ends in the compartment containing the destination token of v_4. Thus, $t(u) = v_4$.

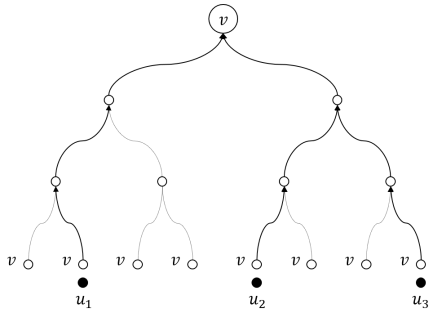

Figure 2 Illustration of Step 3. u_1, u_2, u_3 will follow the paths taken by the destination tokens of v back to v. The paths may overlap. W.h.p. every edge is contained in at most $O(\log n)$ paths.

We show that w.h.p., there are at most $O(\log n)$ tokens per compartment at the end of each stage. Given a stage, the probability that a token ends up in a given compartment in node v is at most

$$\left(\frac{\deg(v)}{2m} + \frac{1}{2mn}\right) \cdot \frac{1}{\deg(v)} \leq \frac{1}{m}.$$

Since there are at most $k \cdot n \leq 1.5m$ tokens, there are at most $O(1)$ tokens ending in a compartment in expectation. By standard Chernoff and union bound argument, w.h.p. there are at most $O(\log n)$ tokens in each compartment.

Moreover, since each node u holds at most $\deg(u) \cdot O(\log n)$ tokens at the beginning of each stage, the random walks can be performed in parallel in $O(\tau_G \cdot \log n)$ rounds by Lemma 12. Therefore, the splitting phase uses $O((\log k) \cdot (\tau_G \cdot (\log n))) = \tilde{O}(\tau_G)$ rounds. At the end of the splitting phase, the multiplicity of each token is one. Moreover, w.h.p. each compartment contains at most $O(\log n)$ tokens.

In the distributing phase, a compartment containing more than one token will start the random walks on *all except one* of its token for $\tau_G(0.1/2m)$ steps. Again, by Lemma 12, this can be done simultaneously for all nodes in $O(\tau_G \cdot \log n)$ rounds. At the end of the random walks, we say a token succeeds if it ends at a compartment without any other tokens. If a token does not succeed, it will go back to the origin. The process is repeated until there is no compartment containing more than one token. Since there are at most $n \cdot k \leq 1.5m$ tokens, at most $1.5m$ compartment can be occupied. Since we run the random walks for $\tau_G(0.1/2m)$ steps, the probability that a random walk ends at a specific compartment is at most $1.1/2m$. Thus, the probability that a token does not succeed is at most $(1.5m) \cdot (1.1/(2m)) = 1.65/2$.

Therefore, a token will succeed w.h.p. after at most $O(\log n)$ trials. By a union bound over the tokens, w.h.p. all tokens succeed after $O(\log n)$ trials. The total running time is $O(\log n \cdot (\tau_G \log n)) = \tilde{O}(\tau_G)$.

Step 2. Each node u creates a source token. The tokens start to perform random walk for $\tau_G(\lambda')$ steps, where $\lambda' = \min(\lambda/(8m), 0.1/m)$ (see Figure 1). If the source token of u ends up in one out of the k compartments with a destination token of v, $t(u)$ will be set to v. Otherwise, if it ends up in a compartment without any destination tokens, it will restart the random walk. The process will be repeated until the source token ends up in a compartment with some destination token.

By our choice of λ', the probability that a token ends at a specific node is at least $0.9/(2m)$. Therefore, the probability that a token successfully ends up in a compartment with a destination token after the random walk is at least

$$nk \cdot \frac{0.9}{2m} \geq (1.5m - n) \cdot \frac{0.9}{2m} \geq (1.5m - m - 1) \cdot \frac{0.9}{2m} \geq \left(\frac{1}{4} - \frac{1}{2m}\right) \cdot 0.9 \geq 0.9/8 \ .$$

The second inequality follows from $m \geq n - 1$ and the third inequality holds for $m \geq 4$. Thus, the number of random walks a token needs to perform until it ends up at a node with some destination token is at most $O(\log n)$ w.h.p. By taking a union bound over all the n tokens, we conclude that w.h.p. every token performs at most $O(\log n)$ random walks. The random walks can be performed simultaneously in $O(\tau_G \cdot \log n)$ rounds, so w.h.p. the total number of rounds is $O(\tau_G \cdot \log^2 n)$.

Next, we show that given two nodes u, v, $\Pr(t(u) = v) \in [(1 - \lambda)/n, (1 + \lambda)/n]$. Let $\mathcal{E}_v$ denote the event that the source token of u ends up in a compartment with a destination token of v. Let $\mathcal{E}$ denote the event that the source token of u ends up in a compartment with some destination token.

By our choice of $\tau(\lambda')$, we have that for all v, $\Pr(\mathcal{E}_v) \in \left[\frac{k}{2m} - k\lambda', \frac{k}{2m} + k\lambda'\right]$ and $\Pr(\mathcal{E}) \in \left[n\left(\frac{k}{2m} - k\lambda'\right), n\left(\frac{k}{2m} + k\lambda'\right)\right]$. Therefore,

$$\begin{aligned}
&\frac{\frac{k}{2m} - k\lambda'}{n\left(\frac{k}{2m} + k\lambda'\right)} \leq \Pr(t(u) = v) \leq \frac{\frac{k}{2m} + k\lambda'}{n\left(\frac{k}{2m} - k\lambda'\right)} \\
&\frac{1}{n} \cdot \frac{1 - 2m\lambda'}{1 + 2m\lambda'} \leq \Pr(t(u) = v) \leq \frac{1}{n} \cdot \frac{1 + 2m\lambda'}{1 - 2m\lambda'} \\
&\frac{1}{n} \cdot (1 - 8m\lambda') \leq \Pr(t(u) = v) \leq \frac{1}{n} \cdot (1 + 8m\lambda') \qquad \text{when } \lambda' \text{ is sufficiently small} \\
&\frac{1}{n} \cdot (1 - \lambda) \leq \Pr(t(u) = v) \leq \frac{1}{n} \cdot (1 + \lambda) \qquad \lambda' \leq \lambda/8m \ .
\end{aligned}$$

Note that since all the source tokens perform random walks independently, when we condition on the choice of nodes in Z for any $u \notin Z \subseteq V$, it is still true that

$$\Pr\left(\mathcal{E}_v \middle| \bigwedge_{z \in Z} t(z)\right) \in \left[\frac{k}{2m} - k\lambda', \frac{k}{2m} + k\lambda'\right]$$

and

$$\Pr\left(\mathcal{E} \middle| \bigwedge_{z \in Z} t(z)\right) \in \left[n(\frac{k}{2m} - k\lambda'), n(\frac{k}{2m} + k\lambda')\right] \ .$$

Thus, $\Pr\left(t(u) = v \middle| \bigwedge_{z \in Z} t(z)\right) \in [(1 - \lambda)/n, (1 + \lambda)/n]$.

Step 3. It remains to show that the messages from u to $t(u)$ can be routed simultaneously for every u in $\tilde{O}(\tau_G)$ rounds.

Let $mid(u)$ denote the node where the source token of u is located at the end of Step 2. The message from u to $mid(u)$ for every u can be simultaneously routed in $\tilde{O}(\tau_G)$ rounds by following the same path taken by the random walk of the source token of u.

Suppose that $t(u) = v$. After the message reaches $mid(u)$, it will follow the path taken by the random walk of the destination token of v to go to v (see Figure 2). Note that multiple source tokens may be matched to a node v (some possibly from the other destination tokens of v). When they follow the paths that lead back to $t(v)$, it is possible that these paths merge and create congestion. However, using a standard Chernoff Bound argument, we can show that for any node v w.h.p. at most $O(\log n)$ different nodes u have $t(u) = v$. Therefore, each step of the parallel random walk can be done with a $O(\log n)$ factor blowup. Thus, the messages between u and $t(u)$ can be routed in $\tilde{O}(\tau_G)$ rounds. This completes the proof of Theorem 4.

4 Algorithms in the GOSSIP Model

In this section, we show that if we have a small number of non-empty nodes, then the empty nodes help approximate F_p faster. As stated in Corollary 6, this result can be translated back to the CONGEST model using Theorem 4 with a blow-up factor $\tilde{O}(\tau_G)$. We exhibit a pre-processing step that duplicates the values so that $\Omega(n)$ nodes become non-empty which is crucial for the algorithms to work while preserving the occurrence ratios.

Throughout this section, for the sake of clarity, we consider the GOSSIP (0) model. However, running our algorithms in GOSSIP $(1/n^c)$, for some sufficiently large constant c, only incurs a small additive error $1/\operatorname{poly}(n)$.

▶ **Lemma 13.** *If the number of non-empty nodes $z < n/3$, we can duplicate the values so that $z\lceil(n/3)/z\rceil$ nodes become non-empty while preserving the occurrences ratios in $O(\log^2 n)$ rounds in the* GOSSIP *model.*

Proof. We divide the process into three phases.

Pre-processing. We assume that the number of non-empty nodes is less than $n/3$, otherwise, we are done. First, the nodes compute the number of non-empty nodes z in $O(\log n)$ rounds [32]. Each node v will form a token that contains $\text{val}(v)$ and t where t is originally set to $\lceil(n/3)/z\rceil$.

Splitting Phase. This phase consists of $O(\log n)$ stages each of which consists of $O(\log n)$ sub-stages. At the beginning of each stage, a node v has a collection of tokens (x_1, t_1), $(x_2, t_2), \ldots$ in its buffer. Each token (x_i, t_i) is split into two tokens $(x_i, \lceil t_i/2\rceil)$ and $(x, \lfloor t_i/2\rfloor)$. It will send these two tokens to two random nodes using two rounds and delete (x_i, t_i) from its buffer. Note that the new tokens $(x_i, \lceil t_i/2\rceil)$ and $(x, \lfloor t_i/2\rfloor)$ will not be split until the next stage. Every stage produces at most $z\lceil(n/3)/z\rceil \leq 2n/3$ new tokens. Each new token is sent to a random node and therefore each node contains $O(\log n)$ new tokens w.h.p by Chernoff bound at the end of that stage. Hence, each sub-stage requires at most $O(\log n)$ rounds to split all the tokens in its buffer w.h.p. After $O(\log n)$ stages, w.h.p all nodes contain $O(\log n)$ tokens and all tokens (x, t) satisfy $t = 1$.

Distributing Phase. At this point, we only have tokens in the form $(x, 1)$, or simply x. In each stage, if v holds more than one token, it will send all but one token (say the first that arrives at v) to the nodes that it talks to. By a standard Chernoff bound argument, each stage requires $O(\log n)$ rounds since each node always holds at most $O(\log n)$ tokens

w.h.p. We say a token x succeeds if it lands in a previously empty node u while no other token lands in u in the same round. Then, u never sends x away from this point onward. Since we have at most $z \cdot \lceil (n/3)/z \rceil \leq 2n/3$ tokens, at least $n/3$ nodes are empty at all times. Consider a token x. In each stage, conditioning on all other tokens' choices, with probability at least $1/3$, x succeeds. Hence, after $O(\log n)$ stages, x succeeds w.h.p and therefore all tokens succeed w.h.p by taking a union bound over all tokens. Since we have at least $\lceil n/3 \rceil$ tokens, the number of non-empty nodes is $\Omega(n)$. Note that the occurrence of each value is rescaled by a factor $\lceil (n/3)/z \rceil$. ◀

After we estimate F_p of the new instance, we can divide the estimator by $(\lceil (n/3)/z \rceil)^p$ to get an estimate for F_p in the original instance. From now on, we can safely assume that the number of non-empty nodes $F_1 = \Omega(n)$, otherwise, we can apply the above pre-processing. A *key observation* is that $F_0 \leq z$, and thus we can analyze our algorithms for when F_0 is small instead.

An ℓ_p-sampling primitive. An ℓ_p-sampling algorithm samples a value $i \in [N]$ with probability f_i^p / F_p. More formally, $\Pr(\text{sample } i) = f_i^p / F_p$.

The ℓ_p-sampling primitive (for $0 \leq p \leq 2$) has been extensively studied in the data stream model. An incomplete list includes [3, 26, 28, 36]. However, most streaming ℓ_p-samplers are rather complicated, and it is unclear how to implement them in the GOSSIP model.

It is trivial to obtain an ℓ_1-sample by virtue of the GOSSIP model. To obtain an ℓ_0-sample (a random value that occurs at least once), we broadcast a randomly chosen pairwise hash function $h : [N] \to [N^3]$ and identify the value corresponds to the smallest hash value in $O(\log n)$ rounds.

Assuming that p is fixed, we now show that if $F_0 = O\left(n^{1/(p-1)}\right)$, then we can perform ℓ_p-sampling in $O(\log n)$ rounds (hence ℓ_2-sampling can always be done in $O(\log n)$ rounds since $F_0 \leq n$). The sampling algorithm proceeds as follows.

Each node v uses p rounds to talk to p random nodes $u_1, \ldots, u_p$. It declares success if $\text{val}(u_1) = \ldots = \text{val}(u_p)$. In that case, let $\text{val}(u_1)$ be v's sample. Among the successful nodes, to break symmetry, broadcast the sample of the node with the smallest ID. If no node succeeds, repeat the process. The following lemma provides a lower bound on F_p based on F_0.

▶ **Lemma 14.** *If $F_1 = \Omega(n)$ and $F_0 = O\left(n^{1/(p-1)}\right)$, then $F_p = \Omega\left(n^{p-1}\right)$.*

Proof. Let the frequency vector be $f = (f_1, \ldots, f_N)$. Without loss of generality, suppose the potentially non-zero entries of f be $f_1, \ldots, f_{Kn^{1/(p-1)}}$ for some constant K. Note that based on our assumption, $f_j = 0$ for all $j > Kn^{1/(p-1)}$. Let $f' = (f_1, \ldots, f_{Kn^{1/(p-1)}})$ be the vector formed by the first $Kn^{1/(p-1)}$ entries. Note that $\|f'\|_1 \geq Cn$ for some constant $0 < C \leq 1$ as assumed.

We will use the following inequality: if the vector x has n entries then

$$\|x\|_q \leq \left(n^{1/q-1/p}\right) \|x\|_p \text{ , for } 0 < q < p \text{ .}$$

Note that f' has $Kn^{1/(p-1)}$ entries. Let $K' = K^{1-1/p}$. We have

$$\left(Kn^{1/(p-1)}\right)^{1-1/p} \|f'\|_p \geq \|f'\|_1$$

$$\|f'\|_p \geq \frac{\|f'\|_1}{K' n^{1/p}}$$

$$F_p \geq \frac{C^p n^p}{K^{p-1} n} = \Omega\left(n^{p-1}\right) \text{ .}$$

The last step follows since K and C are constants and p is fixed. ◀

▶ **Theorem 15.** *If $F_0 = O\left(n^{1/(p-1)}\right)$, then the described algorithm obtains an ℓ_p-sample in $O(\log n)$ rounds in the* GOSSIP *model w.h.p.*

Proof. We can apply the pre-processing step so that $F_1 = \Omega(n)$ while the occurrences ratios are preserved. The probability that a node succeeds is $\Omega\left(\sum_{i=1}^{N} f_i^p/n^p\right) = \Omega\left(F_p/n^p\right)$.

Appealing to Lemma 14, $F_p \geq n^{p-1}/K'$ for some constant K'. Hence, $\Pr(v \text{ succeeds}) \geq 1/(K'n)$. The probability that all n nodes fail is at most $(1-1/(K'n))^n \leq e^{-1/K'}$. We therefore succeed w.h.p by repeating $O(\log n)$ times. Given that v succeeds, the probability that it samples value i is $\left(f_i^p/n^p\right) / \left(\sum_{j=1}^{N} f_j^p/n^p\right) = f_i^p/F_p$ as required. ◀

Approximating F_p. The algorithm by Bar-Yossef et al. [4] that we discuss in the full version [39] for approximating F_0 up to a $1 \pm \epsilon$ factor w.h.p can be emulated in the GOSSIP model in $O(\epsilon^{-2} \log^2 n)$ rounds. We now focus on approximating higher frequency moments. Let $k \leq p$ be an integer. We present an algorithm that w.h.p approximates F_p (for $p \geq 2$) in $\tilde{O}\left(\epsilon^{-2} n^{1-k/p}\right)$ rounds if $F_0 = O\left(n^{1/(k-1)}\right)$. Recall that F_0 is at most the number of non-empty nodes. To approximate F_p, our algorithm makes use of an approximation of F_k and ℓ_k-sampling. This generalizes the approach in [3, 36]. We will prove the following theorem.

We first consider the following algorithm that approximates F_k. For $j = 1, \ldots, C\epsilon^{-2} \log n$, where C is some sufficiently large constant, in the j-th phase, each non-empty node v uses $k-1$ rounds to talk to $k-1$ random nodes $u_1, \ldots, u_{k-1}$. It declares success if $\text{val}(v) = \text{val}(u_1) = \ldots = \text{val}(u_{k-1})$. Let $I_{j,v}$ be the indicator variable for the event v succeeds in the j-th phase. Let $T = C\epsilon^{-2} \log n$. Return the estimate

$$\hat{F}_k = \frac{n^{k-1}}{T} \cdot \sum_{j=1}^{T} \sum_{v=1}^{n} I_{j,v} \ .$$

We now prove Theorem 5. This theorem first shows that $\hat{F}_k$ is a good approximation w.h.p. Then, it combines $\hat{F}_k$ with ℓ_k-sampling to compute a good estimate of F_p in $O\left(\epsilon^{-2} n^{1-k/p} \log^2 n\right)$ rounds.

Proof of Theorem 5. We again can assume that $F_1 = z = \Omega(n)$ as outlined earlier in this section. We first show that $\hat{F}_k = (1 \pm \epsilon) F_k$ w.h.p. In expectation,

$$\mathrm{E}\left[\hat{F}_k\right] = \frac{n^{k-1}}{T} \sum_{j=1}^{T} \sum_{v=1}^{n} \mathrm{E}\left[I_{j,v}\right] = \frac{n^{k-1}}{T} \sum_{j=1}^{T} \sum_{v=1}^{n} \frac{f_{\text{val}(v)}^{k-1}}{n^{k-1}} = \sum_{i=1}^{N} f_i \cdot f_i^{k-1} = F_k \ .$$

Since the indicator variables $I_{j,v}$ are independent, we can apply Chernoff bound directly.

$$\Pr\left(\left|\hat{F}_k - F_k\right| \geq \epsilon F_k\right) = \exp\left(-\Omega\left(\frac{T\epsilon^2 F_k}{n^{k-1}}\right)\right) \leq \exp\left(-\Omega\left(T\epsilon^2\right)\right) \leq 1/\operatorname{poly}(n) \ .$$

The first inequality follows from Lemma 14 and the second inequality is because $T = C\epsilon^{-2} \log n$ for some sufficiently large constant C. Hence, we can approximate F_k up to a $1 \pm \epsilon$ factor in $O(\epsilon^{-2} \log n)$ rounds.

To approximate F_p for $p > k$, we use the following estimator. Let i be an ℓ_k sample. We can compute f_i exactly in $O(\log n)$ rounds. Specifically, each node with value i will put 1 on it and 0 otherwise. Then, we can compute the sum using the algorithm in [32]. Consider the following estimator:

$$\hat{F}_p = \hat{F}_k \cdot f_i^{p-k} \ .$$

We rely on the following lemma. We defer the proof to the end of this section.

▶ **Lemma 16.** *We have* $\mathrm{E}\left[\hat{F}_p\right] = (1 \pm O(\epsilon))F_p$ *and* $\mathrm{V}\left[\hat{F}_p\right] \leq 2n^{1-k/p}F_p^2$.

Hence, by an application of Chebyshev bound, if we take the average of $O\left(n^{1-k/p}\epsilon^{-2}\right)$ estimators, with constant probability, $\hat{F}_p = (1 \pm \epsilon)F_p$. We can amplify the success probability to $1 - 1/\operatorname{poly}(n)$ by the standard median trick, i.e., taking the median of $O(\log n)$ such estimators. ◀

Proof of Lemma 16. In expectation,

$$\begin{aligned}\mathrm{E}\left[\hat{F}_p\right] &= \hat{F}_k \cdot \sum_{i=1}^{N} \frac{f_i^k}{F_k} f_i^{p-k} \\ &= (1 \pm O(\epsilon))F_p \ .\end{aligned}$$

We can bound the variance as follows.

$$\begin{aligned}\mathrm{V}\left[\hat{F}_p\right] &\leq (1 \pm O(\epsilon))F_k^2 \sum_{i=1}^{N} \frac{f_i^k}{F_k} f_i^{2(p-k)} \\ &\leq 2F_k F_{2p-k} \ .\end{aligned}$$

We have $\|f\|_k \leq n^{1/k-1/p}\|f\|_p$, and therefore $F_k \leq n^{1-k/p}F_p^{k/p}$. Additionally, $\|f\|_{2p-k} \leq \|f\|_p$ which implies $F_{2p-k} \leq F_p^{2-k/p}$. Therefore, $\mathrm{V}\left[\hat{F}_p\right] \leq 2n^{1-k/p}F_p^2$. ◀

References

1 Miklós Ajtai, János Komlós, and Endre Szemerédi. An O(n log n) Sorting Network. In *STOC*, pages 1–9. ACM, 1983.

2 Noga Alon, Yossi Matias, and Mario Szegedy. The Space Complexity of Approximating the Frequency Moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

3 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming Algorithms via Precision Sampling. In *FOCS*, pages 363–372. IEEE Computer Society, 2011.

4 Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting Distinct Elements in a Data Stream. In *RANDOM*, volume 2483 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2002.

5 Vladimir Braverman and Stephen R. Chestnut. Universal Sketches for the Frequency Negative Moments and Other Decreasing Streaming Sums. In *APPROX-RANDOM*, volume 40 of *LIPIcs*, pages 591–605. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

6 Vladimir Braverman, Stephen R. Chestnut, David P. Woodruff, and Lin F. Yang. Streaming Space Complexity of Nearly All Functions of One Variable on Frequency Vectors. In *PODS*, pages 261–276. ACM, 2016.

7 Vladimir Braverman and Rafail Ostrovsky. Zero-one frequency laws. In *STOC*, pages 281–290. ACM, 2010.

8 Joshua Brody and Amit Chakrabarti. A Multi-Round Communication Lower Bound for Gap Hamming and Some Consequences. In *IEEE Conference on Computational Complexity*, pages 358–368. IEEE Computer Society, 2009.

9 Amit Chakrabarti, Khanh Do Ba, and S. Muthukrishnan. Estimating Entropy and Entropy Norm on Data Streams. In *STACS*, volume 3884 of *Lecture Notes in Computer Science*, pages 196–205. Springer, 2006.

10 Amit Chakrabarti, Graham Cormode, and Andrew McGregor. A near-optimal algorithm for estimating the entropy of a stream. *ACM Trans. Algorithms*, 6(3):51:1–51:21, 2010.

11 Jen-Yeu Chen and Gopal Pandurangan. Almost-Optimal Gossip-Based Aggregate Computation. *SIAM Journal on Computing*, 41(3):455–483, 2012.

12 Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic Algorithms for Replicated Database Maintenance. In *Proc. 6th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 1–12, 1987.

13 A.M. Frieze and G.R. Grimmett. The shortest-path problem for graphs with random arc-lengths. *Discrete Applied Mathematics*, 10(1):57–77, 1985.

14 Sumit Ganguly. Taylor Polynomial Estimator for Estimating Frequency Moments. In *ICALP (1)*, volume 9134 of *Lecture Notes in Computer Science*, pages 542–553. Springer, 2015.

15 Mohsen Ghaffari, Fabian Kuhn, and Hsin-Hao Su. Distributed MST and Routing in Almost Mixing Time. In *PODC*, pages 131–140. ACM, 2017.

16 Mohsen Ghaffari and Jason Li. New Distributed Algorithms in Almost Mixing Time via Transformations from Parallel Algorithms. In *DISC*, volume 121 of *LIPIcs*, pages 31:1–31:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

17 George Giakkoupis, Anne-Marie Kermarrec, and Philipp Woelfel. Gossip Protocols for Renaming and Sorting. In Yehuda Afek, editor, *DISC*, 2013.

18 Phillip B. Gibbons and Srikanta Tirthapura. Estimating simple functions on the union of data streams. In *SPAA*, pages 281–291. ACM, 2001.

19 Oded Goldreich. Basic Facts about Expander Graphs. In *Studies in Complexity and Cryptography*, volume 6650 of *Lecture Notes in Computer Science*, pages 451–464. Springer, 2011.

20 Yu Gu, Andrew McCallum, and Donald F. Towsley. Detecting Anomalies in Network Traffic Using Maximum Entropy Estimation. In *Internet Measurment Conference*, pages 345–350. USENIX Association, 2005.

21 Bernhard Haeupler, Jeet Mohapatra, and Hsin-Hao Su. Optimal Gossip Algorithms for Exact and Approximate Quantile Computations. In *PODC*, pages 179–188. ACM, 2018.

22 Nicholas J. A. Harvey, Jelani Nelson, and Krzysztof Onak. Sketching and Streaming Entropy via Approximation Theory. In *FOCS*, pages 489–498. IEEE Computer Society, 2008.

23 Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–561, 2006.

24 Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, 2006.

25 Piotr Indyk and David P. Woodruff. Optimal approximations of the frequency moments of data streams. In *STOC*, pages 202–208. ACM, 2005.

26 Rajesh Jayaram and David P. Woodruff. Perfect Lp Sampling in a Data Stream. In *FOCS*, pages 544–555. IEEE Computer Society, 2018.

27 Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. Gossip-based Peer Sampling. *ACM Trans. Comput. Syst.*, 25(3), August 2007. doi:10.1145/1275517.1275520.

28 Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for Lp samplers, finding duplicates in streams, and related problems. In *PODS*, pages 49–58. ACM, 2011.

29 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *PODS*, pages 41–52. ACM, 2010.

30 R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized rumor spreading. In *Proc. 41st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 565–574, 2000.

31 Srinivas Kashyap, Supratim Deb, K. V. M. Naidu, Rajeev Rastogi, and Anand Srinivasan. Efficient Gossip-based Aggregate Computation. In *Proc. of the 25th ACM Symposium on Principles of Database Systems (PODS)*, pages 308–317, 2006.

32 David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *Proc. of the Symp. on Found. of Comp. Sci. (FOCS)*, pages 482–491, 2003.

33 Fabian Kuhn, Thomas Locher, and Stefan Schmid. Distributed computation of the mode. In *PODC*, pages 15–24. ACM, 2008.

34 Fabian Kuhn, Thomas Locher, and Rogert Wattenhofer. Tight Bounds for Distributed Selection. In *Proc. of 19th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 145–153, 2007.

35 Fabian Kuhn and Rotem Oshman. The Complexity of Data Aggregation in Directed Networks. In *DISC*, volume 6950 of *Lecture Notes in Computer Science*, pages 416–431. Springer, 2011.

36 Morteza Monemizadeh and David P. Woodruff. 1-Pass Relative-Error L_p-Sampling with Applications. In *SODA*, pages 1143–1160. SIAM, 2010.

37 David Peleg. *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

38 Boris Pittel. On Spreading a Rumor. *SIAM Journal on Applied Mathematics*, 47(1):213–223, 1987.

39 Hsin-Hao Su and Hoa T. Vu. Distributed Data Summarization in Well-Connected Networks. *CoRR*, abs/1908.00236, 2019. `arXiv:1908.00236`.

40 Arno Wagner and Bernhard Plattner. Entropy Based Worm and Anomaly Detection in Fast IP Networks. In *WETICE*, pages 172–177. IEEE Computer Society, 2005.

41 David P. Woodruff. Optimal space lower bounds for all frequency moments. In *SODA*, pages 167–175. SIAM, 2004.

42 Kuai Xu, Zhi-Li Zhang, and Supratik Bhattacharyya. Profiling internet backbone traffic: behavior models and applications. In *SIGCOMM*, pages 169–180. ACM, 2005.