Deterministic Near-Optimal Distributed Listing of Cliques∗

Keren Censor-Hillel
ckeren@cs.technion.ac.il
Technion - Israel Institute of Technology
Haifa, Israel

Dean Leitersdorf
dean.leitersdorf@gmail.com
Technion - Israel Institute of Technology
Haifa, Israel

David Vulakh
dvulakh@mit.edu
Massachusetts Institute of Technology
Cambridge, MA, USA

ABSTRACT
The importance of classifying connections in large graphs has been the motivation for a rich line of work on distributed subgraph finding that has led to exciting recent breakthroughs. A crucial aspect that remained open was whether deterministic algorithms can be as efficient as their randomized counterparts, where the latter are known to be tight up to polylogarithmic factors.

We give deterministic distributed algorithms for listing cliques of size \( p \) in \( n^{1-2/p+o(1)} \) rounds in the CONGEST model. For triangles, our \( n^{1/3+o(1)} \) round complexity improves upon the previous state of the art of \( n^{2/3+o(1)} \) rounds [Chang and Saranurak, FOCS 2020]. For cliques of size \( p \geq 4 \), ours are the first non-trivial deterministic distributed algorithms. Given known lower bounds, for all values \( p \geq 3 \) our algorithms are tight up to a \( n^{o(1)} \) subpolynomial factor, which comes from the deterministic routing procedure we use.

1 INTRODUCTION

Subgraph finding is a crucial component in many computation settings, as it characterizes the connections in a graph. In distributed computing, further motivation for finding small subgraphs is that some algorithms run faster on networks without certain subgraphs: [28, 37] show fast algorithms for coloring and finding large cuts, respectively, in triangle-free graphs, and [7] quickly computes exact girth in graphs without small cycles. In this paper, we focus on clique listing in the CONGEST model, in which \( n \) vertices in a synchronous network are required to find all cliques in the network graph by exchanging messages of \( O(\log n) \) bits.

A major open question about the CONGEST model is whether randomization is necessary for this problem. A recent groundbreaking line of work [5, 10, 14, 15, 20, 30] culminated in clique-listing algorithms whose complexities are within polylogarithmic factors of the lower bounds presented by [22, 30, 36]. While these are all randomized algorithms, powerful recent work by Chang and Saranurak [16] showed the first non-trivial deterministic algorithm for the case of triangle listing. Still, it exceeds the lower bound by a polynomial number of rounds, leaving the following major question open (see also Open Problem 2.2 in a recent survey [4]):

What is the complexity of deterministic clique listing in the CONGEST model, and is there a separation between randomized and deterministic algorithms?

We present an algorithm answering this question for all \( p \)-cliques (denoted \( K_p \)). Our algorithm finishes in \( n^{1-2/p+o(1)} \) rounds, within \( n^{o(1)} \) rounds of the \( \Omega(n^{1-2/p}) \) lower bound of [22, 30, 36], and for listing cliques larger than triangles is the first non-trivial deterministic algorithm. This complexity matches the optimal randomized results of [5, 13], up to the additional \( n^{o(1)} \) factor. This factor is due to the deterministic expander routing scheme of [16], and might be improved upon given future progress in routing.

THEOREM 1. Given a constant \( p \geq 3 \) and a graph \( G = (V,E) \) with \( n = |V| \) vertices, there exists a deterministic CONGEST algorithm that completes in \( n^{1-2/p+o(1)} \) rounds and lists all instances of \( K_p \) in \( G \).

Our key technical contribution that enables showing Theorem 1 is a simulation in CONGEST of a class of algorithms we call partial-pass streaming algorithms. We use these to simulate deterministic load balancing tools from the stronger CONGESTED CLIQUE model (where every two vertices can exchange \( O(\log n) \)-bit messages per round). It is possible that our approach of extracting a partial-pass streaming algorithm and simulating it could be useful elsewhere.

1.1 Background and Challenges
The groundbreaking triangle listing result of Chang, Pettie, Saranurak, and Zhang [13] (which combines the two conference papers [14, 15]) consists of a distributed expander decomposition, in which the graph is broken up into well-connected (high conductance / low mixing time) clusters, and CONGESTED CLIQUE-inspired algorithms that run efficiently in each cluster using the expander routing of [25, 26]. The remaining \( \epsilon \)-fraction of inter-cluster edges are handled recursively. This general framework has become the foundation for further work in this area [5, 7, 8, 10, 16, 20, 31].
Specifically, after running the expander decomposition, the $K_3$ listing algorithm of [13] runs a randomized version of the triangle listing CONGESTED CLIQUE algorithm of Dolev, Lenzen, and Peled [18]. For the case of $p \geq 4$, significant new challenges arise as cliques can have their edges spread out over multiple clusters in the decomposition. This implies that some edges need to be learned by a cluster although neither of their endpoints is in the cluster, and thus the number of edges that a cluster has to process is much larger. The listing algorithms of Eden, Fiat, Fischer, Kuhn, and Oshman [20], Censor-Hillel, Le Gall, and Leitersdorf [10], and Censor-Hillel, Chang, Le Gall, and Leitersdorf [5] use various techniques for resolving the above.

Usage of randomization in previous approaches. Together, [5, 13] show optimal randomized algorithms for listing cliques of any size in CONGEST. Conceptually, randomization is used in three components of the algorithms. The first two are expander decomposition and routing; the third is load balancing for the listing operation inside clusters, upon which we now elaborate.

In the load balancing task, each cluster knows some set of edges (distributed across the cluster vertices) — only the cluster’s own edges for $K_3$, and for $K_{p \geq 4}$ perhaps also edges from other clusters. The goal is for the vertices to redistribute these edges, such that any instance of $K_p$ composed of these edges is known to at least one vertex in the cluster. This must happen without any vertex slowing down the network by sending or receiving too much information.

The standard approach is to construct a random partition of the vertex set $V = V_1 \cup V_2 \cup \cdots \cup V_x$, for $x = \Theta(n^{1/P})$, so that the number of edges between two parts $V_i, V_j$ is, with high probability, roughly $|E|/x^2$. Then, each vertex $v$ is assigned some parts $V_i$ and learns all edges between them. By assigning a number of parts proportional to the degree of $v$, the near-optimal round complexity is obtained.

Challenges in removing randomization. Chang and Saranurak [16] show deterministic expander decomposition and routing procedures. However, deterministically solving the above load balancing task is difficult to do inside the clusters, implying a polynomial overhead to their triangle listing round complexity.

The challenge of finding an efficient deterministic algorithm for the load balancing task arises not only in CONGEST, but also in CONGESTED CLIQUE. Specifically, it is unknown how to efficiently find in a deterministic manner a well-balanced partition of the graph such that the number of edges between every two parts is roughly the same. In CONGESTED CLIQUE, [11] bypassed this challenge for triangles, and [7] were able to do so for any subgraph (with at most $\log n$ vertices). The approach of the latter is that instead of using one partition, they design a polynomial number of partitions and different vertices choose to work with different combinations of partitions. The different partitions are related to each other using a structure called a partition tree.

One approach for deterministic load balancing in CONGEST for clique listing is to construct the above partition trees inside clusters. However, the above algorithm for building these partitions cannot efficiently be directly adapted to CONGEST. This is because the algorithm has low round complexity, but high message complexity: although it finishes in $O(1)$ rounds in the CONGESTED CLIQUE model, it involves sending $\Theta(n^2)$ messages. This high message complexity is prohibitive for the CONGEST model, as the bandwidth available, i.e. the number of communication edges in the graph or in a cluster, might be dramatically smaller than $\Theta(n^2)$, as we now elaborate.

When dealing with $K_3$, our target round complexity is $O(n^{1/3})$, as the known lower bound stated above is $\Omega(n^{1/3})$. This implies a hard restriction on the bandwidth available within the clusters. Concretely, it can be shown that some low-degree vertices in the clusters can only receive $O(n^{2/3})$ messages in total throughout the entire algorithm. Therefore, for clusters with $\Omega(n^{2/3})$ vertices, some vertices might not even know basic information about the other cluster vertices, such as how much data each holds, and so performing any sort of load balancing is clearly tricky.

For $K_{p \geq 4}$, our target complexity inherently increases, as the lower bound is $\Omega(n^{1-2/P})$, at least $\Omega(n^{1/2})$ for $p \geq 4$. While this eases the pressure on the bandwidth, we face a new challenge: constructing partition trees on data from multiple clusters. As described above, a clique of size $\geq 4$ can have edges in multiple clusters, and so as in [5], we need to make some edges from outside a cluster known to the vertices of the cluster. To capture this, we must substantially generalize partition trees to distinguish between edges in the cluster and those brought into it.

1.2 Our Approach

We introduce partial-pass streaming algorithms, and show that simulating them in CONGEST requires few messages. Then, we show a partial-pass streaming algorithm that constructs a generalized partition tree. Finally, we plug this into the deterministic expander decomposition and routing of [16] to list $K_p$.

Partial-pass streaming algorithms. Partial-pass streaming algorithms are similar to standard streaming algorithms (see, e.g., [2, 34] or the survey [35]), as they maintain a small state, sequentially process an input stream, and cannot return to previous parts of the input. However, a key distinction of partial-pass streaming algorithms is that they strive to make less than one pass over the input stream by skipping some of it. In this setting, a stream is composed of tokens reflecting two levels of granularity: a large number of auxiliary tokens, each with some fine-grained data, and a small number of main tokens with very coarse-grained data summarizing some number of auxiliary tokens. Partial-pass streaming algorithms are able to access all the main tokens in a given stream, while reading only a limited number of auxiliary tokens. As such, the number of stream tokens accessed by a partial-pass streaming algorithm is much smaller than the total stream length.

As intuition for this definition and why it fits well with a CONGEST simulation, recall that in CONGEST the input is dispersed across vertices. The raw data each vertex gets can be seen as auxiliary tokens, and each vertex can locally compute a summary of its data, creating a main token. When an algorithm runs, it might suffice to sometimes only read the main token produced by a vertex, while at other occasions the internal auxiliary tokens are needed.

Simulating partial-pass streaming algorithms in CONGEST. For simplicity, suppose each vertex has a single main token that summarizes the possibly many auxiliary tokens it holds. Assume also that the vertices are ordered in the way their tokens appear in the stream, i.e., vertex with identifier 1 has the first main token. The algorithm that we run on the stream has a small polylog(n)-bit state. Consider two extreme approaches.
Approach 1: State Passing. Vertex 1 starts simulating the algorithm and locally executes it on its tokens. Then, it passes the state of the algorithm to vertex 2 and so forth. While only $O(n)$ messages are used, this takes $\Theta(n)$ rounds.

Approach 2: Leader with Queries. An arbitrary vertex $v$ is denoted as leader and learns all $n$ main tokens. Then, $v$ locally simulates the algorithm. When auxiliary tokens of a certain main token are needed, $v$ sends the state of the algorithm to the vertex $u$ which sent $v$ the specific main token. Vertex $u$ simulates the algorithm on its local auxiliary tokens, sends the state back to $v$. As main tokens summarize the auxiliary tokens and partial-pass streaming algorithms are able to skip the vast majority of auxiliary tokens, $v$ sends the state to other vertices only few times during the entire execution. However, we are burdened by the fact that $v$ initially learns all $n$ main tokens by itself.

Our approach. We opt for a combination of both approaches. Denote some set of vertices $A$ of sublinear polynomial size, ordered as $A = \{a_1, \ldots, a_{|A|}\}$. The vertices in $A$ collectively learn all main tokens $a_1$ learns the first $n/|A|$ main tokens, followed by $a_2$ which learns the next $n/|A|$, and so forth. Vertex $a_1$ starts executing the algorithm on its local data. Whenever it hits a main token for which the corresponding auxiliary tokens are needed, it sends the state of the algorithm back to whichever vertex sent it the relevant main token. That vertex simulates the algorithm, and returns the state back to $a_1$. After $a_1$ processes the main tokens it knows, it forwards the state of the algorithm to $a_2$, and so forth, until $a_{|A|}$ completes the run of the algorithm. This approach passes the state only few times and each vertex learns only few main tokens, thus it has both low round and low message complexities.

Constructing generalized partition trees by a partial-pass streaming algorithm. We design a partial-pass streaming algorithm that constructs the generalized partition trees we require, so that we can then invoke our simulation of it in CONGEST. Specifically, each cluster in CONGEST runs a setup in order to create an input stream $S$, and then we simulate a partial-pass streaming algorithm that runs on $S$. We begin by describing how nodes of a cluster collectively create $S$, and then we define the partial-pass streaming algorithm which they simulate.

At a high level, a common operation in constructing partition trees is that there is a globally known vertex set $W$, and we wish to partition vertices $V$ into sets $V_1, \ldots, V_x$, for some $x$, such that for each $i, j \in [x]$, the number of edges in $E(V_i, W)$ is roughly the same as in $E(V_j, W)$. Denoting $V_C$ as the set of vertices in cluster $C$, each $v \in V_C$ might also hold edges incident to vertices outside $C$, a set which we denote as $F(v) \subseteq V \setminus V_C$. For each $v \in V_C$, define $|F(v)| + 1$ auxiliary tokens as the value $|E(v, W)|$ and values $|E(u, W)|$ for each $u \in F(v)$. Also define a main token for $v$ as the sum of these auxiliary tokens. The input stream $S$ is the concatenation of the tokens defined for each $v \in V_C$.

The partial-pass streaming algorithm runs on $S$ and maintains an incomplete part $V_j$ in the partition to which it currently adds vertices. When reaching the main token of vertex $v \in V_C$, it attempts to add all of $F(v) \cup \{v\}$ to $V_j$. From the main token of $v$, it knows how many edges will be added to $E(V_j, W)$ if it does so. If this value is acceptable, the algorithm proceeds. Otherwise, it requests all the auxiliary tokens of $v$, and adds the vertices one by one until $V_j$ is too large, at which point it closes $V_j$ and starts filling part $V_{j+1}$.

Roadmap. We provide required background in Section 2, and in Section 3 introduce and efficiently simulate partial-pass streaming algorithms in high-conductance clusters in CONGEST. In Section 4, we construct partition trees using partial-pass streaming algorithms. In the full version [12], we wrap these into $K_3$ and $K_{2,4}$ listing algorithms.

1.3 Further Related Work

Triangle finding. In CONGEST clique, the seminal work of [18] presents an $O(n^{1/3}/\log n)$-round deterministic algorithm for $K_3$ listing and faster results for detection and counting for certain graphs. Matching listing lower bounds are shown in [30, 36]. For sparse graphs, [11, 18, 36] show faster results, some of which are tight due to [36]. The state-of-the-art for detection and counting is $O(n^{0.158})$ rounds, based on fast matrix multiplication [9]. It is considered hard to show lower bounds for this, due to the reduction to circuit complexity of [19].

In CONGEST, the first breakthrough for triangle finding is by [30], who show the first non-trivial algorithms for listing and detection in $O(n^{3/4})$ and $O(n^{2/3})$ rounds w.h.p., respectively. Subsequently, [13] showed a breakthrough by listing in optimal complexity of $O(n^{1/3})$ rounds, w.h.p., using an expander decomposition together with the routing techniques of [25, 26]. [29] show an $O(\Delta/\log n + \log \log \Delta)$-round algorithm, where $\Delta$ is the maximal degree in the graph. The first non-trivial deterministic algorithm was given by [16], taking $O(n^{0.58})$ rounds for detection and $\pi^{2/3}+o(1)$ for listing. On the lower bound side, it is only known that more than one round is needed for detection [1, 22]. Proving any polynomial lower bound is considered difficult [20].

Finding larger cliques. In CONGEST clique, [18] show an $O(n^{1-2/\log n})$-round result for $K_p$ listing (or any subgraph with $p$ vertices). This is optimal for $K_p$ due to the lower bound of [22]. A randomized sparsity aware algorithm was shown for $K_p$ listing, for $p \geq 4$, in [10], and a deterministic construction using partition trees was later given by [7]; these results are tight, for graphs of all sparsities, due to [22, 30, 36].

In CONGEST, the expander decompositions of [13] are used by [20] to show $K_4$ and $K_3$ listing in $n^{5/6+o(1)}$ and $n^{7/75+o(1)}$ rounds, respectively. Later, [10] show the first sub-linear result for all $p \geq 4$, followed by the $O(n^{1-2/\log n})$-round result of [5], which is optimal due to the lower bound of [22]. For $p = 4$, the optimal $\Theta(n^{1/2})$-round $K_4$ listing is also the best possible for detecting $K_4$, due to the lower bound by [17].

Other subgraphs and additional related results. Small cycles received much attention with many algorithms and hardness results, shown in CONGEST clique by [7, 9, 11], and in CONGEST by [7, 19, 20, 22, 33]. For all $p \geq 3$, a gap between $K_p$ detection and listing is shown in quantum CONGEST and quantum CONGESTED clique in [8, 31]. Finding trees was studied in [19, 33], and more general subgraphs in [20, 22, 27]. Testing, in which we need to determine if a graph is either free of a subgraph or many edges need to be removed from it to reach this state, was studied in [3, 6, 21, 23, 24].
2 PRELIMINARIES

Notations. Given a graph \( G = (V, E) \), and a set \( E' \subseteq E \), denote by \( G[E'] \) the graph induced by \( E' \) on \( G \). For a vertex \( v \in V \) and a subgraph \( G' = (V', E') \) of \( G \), denote by \( \deg_G(v) \) the degree of \( v \) in \( G' \), \( \deg_{G'}(v) = |\{u \in E' : (v, u) \in E'\}| \). For a subset of vertices \( V' \subseteq V \) and a vertex \( v \in V \), denote by \( \deg_{V'}(v) \) the degree of \( v \) into \( V' \), \( \deg_{V'}(v) = |\{u \in V' : (v, u) \in E\}| \). We restate expander decompositions and routings that underpin CONGEST listings algorithms.

Definition 2 (Conductance). Given a graph \( G = (V, E) \) and a cut \( S \subseteq V \), let \( \varphi(S) = \sum_{v \in S} \deg(v) \) and \( \varphi(S) = E(S, V \setminus S) \). Then, the conductance of a cut \( S \subseteq V \) is \( \Phi(S) = |\varphi(S)|/\min(\varphi(S), \varphi(V \setminus S)) \) and the conductance of \( G \) is the minimum conductance over all nontrivial cuts \( S, \Phi(G) = \min_{S \subseteq V : S \neq \emptyset, \varphi(S)} \Phi(S) \).

A related value is mixing time, denoted \( \tau(G) \), which is roughly the number of steps a random walk takes to reach the stationary distribution — a precise definition is not required for our purposes. As demonstrated by [32, Corollary 2.3], \( \tau(G) \) and \( \Phi(G) \) are related by \( \tau(G) \leq \Theta(\log^2 \Phi(G)) \). As \( \tau(G) \) is always at least the diameter, this shows the following.

Theorem 3. The diameter of a graph with conductance \( \Phi \) is \( O(\Phi^{-2} \log n) \).

Definition 4 (\( \phi \)-Cluster, Expander Decomposition). A subgraph \( C = (V_C, E_C) \) of a given graph \( G = (V, E) \) is called a \( \phi \)-Cluster in \( G \) if \( \Phi(C) \geq \phi \). An \( (\epsilon, \phi) \)-expander decomposition of a graph \( G \) is a partition of \( E = E_1 \cup E_2 \cup \cdots \cup E_k \) for some \( k \), such that the subgraphs \( G[E_1], \ldots, G[E_k] \) are vertex-disjoint \( \phi \)-clusters in \( G \), and \( |E_i| \leq \epsilon|E| \). We require the two following deterministic tools from [16].

Theorem 5 (CONGEST Expander Decomposition [16, Theorem 1.1]). Let \( 0 < \epsilon < 1 \) be a parameter. An \( (\epsilon, \phi) \)-expander decomposition of a graph \( G = (V, E) \) with \( \phi = \text{poly}(\epsilon) \cdot 2^O(\sqrt{\log n \log \log n}) \) can be computed by a deterministic algorithm in CONGEST in \( \text{poly}(\epsilon^{-1}) \cdot 2^O(\sqrt{\log n \log \log n}) \) rounds.

Theorem 6 (CONGEST Routing [16, Theorem 1.2]). Let \( G = (V, E) \) be a graph with conductance \( \phi \), where each vertex \( v \in V \) is a source and destination of \( O(L) \cdot \deg(v) \) messages. Then there is a deterministic algorithm in the CONGEST that routes all messages to their destination in \( L \cdot \text{poly}(\phi^{-1}) \cdot 2^O(\log^{11} n \log^{11} \log n) \) rounds.

In order to achieve sufficient bandwidth within clusters, we need both good conductance and good guarantees about the degrees of vertices in the cluster. The following definition refines \( \phi \)-clusters with the needed constraints. Figure 1 illustrates the different designations for vertices in such clusters.

Definition 7 ((\( \phi \), \( \delta \))-Communication Cluster). A \( (\phi, \delta) \)-communication cluster is a \( \phi \)-cluster \( C = (V_C, E_C) \) in graph \( G = (V, E) \), together with a \( V_C^\ast \subseteq V_C \) such that for each \( v \in V_C^\ast \), \( \deg_C(v) \geq \delta \), and \( v \) knows \( V_C \) and values \( \delta, n = |V|, K = |V_C|, k = |V_C^\ast| \). The communication degree of \( v \in V_C^\ast \) is the number of edges in \( E_C \) adjacent to \( v \), \( \deg_C(v) = |\{u \in V_C : (v, u) \in E_C\}| \). Denote by \( \mu \) the average degree of vertices in \( V_C^\ast \), that is, \( \mu = \frac{1}{k} \sum_{v \in V_C^\ast} \deg_C(v) \), and by \( V_C^\ast \) the vertices in \( V_C^\ast \) with at least half-average communication degree, that is, \( V_C^\ast = \{v \in V_C^\ast : \deg_C(v) \geq \frac{\mu}{2}\} \).

In the full version [12], we use the decomposition of Theorem 5 as the foundation of a recursive listing algorithm. It produces a \( (\phi, \delta) \)-communication cluster \( C_i \) from each \( E_i \) and uses Theorem 6 to run algorithms for listing \( K_{\mu} \) that intersect each cluster. With correctly chosen \( C_i, \phi \) and \( \delta \) are sufficiently high. Here, we show a strategy for selecting an edge set for which we list all cliques that intersect it and state a lemma of [16] showing that a constant fraction of the total edges are eliminated in each recursive step.

Theorem 5 partitions \( E \) into \( E = E_1 \cup \cdots \cup E_k \), where \( |E_i| \leq \epsilon|E| \). Let the vertices incident to \( E_i \) be \( V_i \) and those in \( V_i \) with the majority of their edges in \( E_i \) be \( V_{C_i} = \{v \in V_i : \deg_{E_i}(v) \geq \frac{\deg_{E_i}(v)}{2}\} \). We use a cluster on \( E_i \) to list cliques intersecting \( E_{C_i} \cap V_{C_i} \). Denote by \( E_{C_i} \) the edges in \( E_i \cap V_{C_i} \times V_{C_i} \). If the number of edges not in some \( E_{C_i} \) is small, we can recursively repeat the algorithm on the graph induced by these remaining edges, for an \( O(\log n) \) recursion depth. As a corollary of the following, the total number of edges in \( E \) that are not in some \( E_{C_i} \) is at most \( 3e|E| \).

Lemma 8 ([16, Lemma 6.1]). \( | \cup_{1 \leq i \leq k} E_i \setminus E_{C_i}^\ast | \leq 2e|E| \).

3 SIMULATING PARTIAL-PASS STREAMING ALGORITHMS IN CONGEST

A partial-pass streaming algorithm for parameters \( L, N_m, N_{out}, B_{aux}, B_{write} \) satisfies the following. There is an input stream \( S \) of \( N_m \) main tokens, \( S = (\tau_1, \tau_2, \ldots, \tau_{N_m}) \). Each token \( \tau_i \) has \( t_i \) associated auxiliary tokens, \( a_{\tau_i} = (a_{\tau_1}, a_{\tau_2}, \ldots, a_{\tau_{N_m}}) \). The algorithm produces a write-only output stream \( R = (\rho_1, \rho_2, \ldots, \rho_{N_m}) \), by invoking READ, GET-AUX, or WRITE operations. READ reads the next token from \( S \); GET-AUX adds the auxiliary tokens associated with the last-read main token to the beginning of \( S \) — if the previous READ gave \( \tau_i \), the next \( t_i \) READ operations give the sequence \( a_{\tau_i} \); WRITE writes a token to the end of the output.

![Figure 1: A schematic of the notations for the types of vertices and edges used in this paper. The rings partition the sets \( V_C^\ast \subseteq V_C \subseteq V \), so black vertices are members of \( V_C^\ast \), grey \( V_C \), hatched \( V_{C_i} \), and white \( V \). Bold edges are in \( E(V_C^\ast, V_C) \); these are the edges for which we endeavor to list cliques. Solid edges are used for communication, \( E(V_C^\ast, V_C) \). Solid edges that cross the second ring and dashed edges are in \( E \subseteq E(V_C^\ast, V_C) \). Dotted edges are in \( E' \subseteq E(V_C^\ast, V_C) \).

274
stream $R$. The algorithm performs $\text{GET-AUX}$ at most $B_{\text{aux}}$ times and performs at most $B_{\text{write}}$ WRITE operations between READ operations that read consecutive main tokens $t_i, t_{i+1}$. All main, auxiliary, and output tokens are at most $L$ bits long. Apart from the output stream $R$ for which $N_{\text{out}}$ may be large, the rest of the space used must be polynomial in $L$.

The motivation for the restriction of accessing auxiliary tokens for at most $B_{\text{aux}}$ main tokens is to use the main tokens as “summaries” of their corresponding sequences of auxiliary tokens. Our algorithms operate on extremely long sequences of input; that is, $\sum_{1 \leq j \leq N_{\text{in}}} t_j$ is huge. We cannot afford to move all of the auxiliary tokens around the cluster during a simulation in CONGEST; however, most of the time, all necessary information about the input can be obtained from some summarizing function that aggregates auxiliary tokens into one main token. When an algorithm finds it needs more granular information, it inspects the auxiliary tokens corresponding to a main token. Since doing so is expensive, we need a good bound $B_{\text{aux}}$ on the number of times this happens.

### 3.1 The Simulation

We efficiently simulate partial-pass streaming algorithms when their input is distributed to vertices of a cluster. We require that each vertex holds a contiguous part of the input stream and the vertices hold these parts in order of their number, i.e., the lowest-numbered vertex holds a prefix of the input stream, the next-lowest-numbered one a prefix of the rest of the stream, etc. The reason for this is that we simulate a partial-pass streaming algorithm by moving its state from one vertex to another in the order that they hold the input. We require a predetermined ordering as the degree of vertices in the communication cluster can be low and we may not have enough bandwidth to distribute information about which part of the input is held by which vertex. In our applications, it is easy to satisfy this.

**Definition 9** (Streaming Input Cluster). A streaming input cluster for a parameter $T_{\text{max}}$ is a $(\phi, \delta)$-communication cluster $C = (V_C, E_C)$ in $G = (V, E)$ for some values $\phi \in O(\text{polylog } n)$ and $\delta$, together with a set of streaming algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_C$ with parameters $L, N_{\text{in}}, N_{\text{out}}, B_{\text{aux}}, B_{\text{write}}$ known to all $v \in V_C$. The token lengths satisfy $L = O(\text{polylog } n)$.

$C$ receives the input to the algorithms distributed among its vertices so that each vertex $v \in V_C$ holds a contiguous interval $\sigma_{j,v}$ of at most $T_{\text{max}}$ main tokens from the input stream $S_j$ to algorithm $\mathcal{A}_j$, and the vertices $V_C$ are numbered in order of the input intervals they hold; that is, concatenating the intervals of main tokens $(\sigma_{j,1}, \ldots, \sigma_{j,v})$ produces exactly $S_j$ for all algorithms $\mathcal{A}_j$. We call this condition input contiguity. Additionally, for each main token $t_i$ from stream $S_j$ held by a vertex $v$, $v$ also holds the associated auxiliary tokens $a_{i,1}, a_{i,2}, \ldots, a_{i,t_i}$.

Several procedures presented in this paper involve delegating the responsibility of collecting information from some set of vertices $V'$, so that each vertex in a set $V'$ is responsible for a relatively small number of vertices $u \in V'$. We often additionally require that the vertices $u \in V'$ for which some $v \in V'$ is responsible be contiguously numbered. We formalize this notion here.

**Definition 10** ($\langle \beta, V' \rangle$-Vertex Chain). Given a graph $G = (V, E)$ and contiguously-numbered vertices $V' \subseteq V$, a $\langle \beta, V' \rangle$-vertex chain for integer $\beta$ is an ordered set of $y = \left\lceil \frac{|V'|}{\beta} \right\rceil$ vertices $V' \subseteq V$, denoted $V'[1], \ldots, V'[y]$. Each $u \in V'$ is assigned to a $v \in V$ according to a many-to-one function $f_{UV}(u)$, which satisfies (i) for all $v \in V'$, there are at most $\beta$ vertices in $f_{V'[1]}(v)$ and they are contiguously numbered and known to $v$, and (ii) $u \in V'$ knows $f_{UV}(u)$.

We now state and prove the conditions for efficient simulation of partial-pass streaming algorithms in CONGEST.

**Theorem 11.** Given a streaming input cluster $C = (V_C, E_C)$ with $k = |V_C|$, for any integer parameter $1 \leq \lambda \leq \frac{k}{\xi}$, all $\mathcal{A}_1, \ldots, \mathcal{A}_C$ can be simulated in parallel in CONGEST in $\left(\frac{T_{\text{max}}}{\xi} + \frac{\xi}{\lambda} + (B_{\text{aux}} + 1)(\lambda + \frac{\xi}{\lambda})\right) \cdot n^{o(1)}$ rounds. After the simulation, each token in the output streams $R_1, \ldots, R_C$ is known to some vertex $v \in V_C$, and each $v \in V_C$ knows $O(\min(N_{\text{out}}, B_{\text{write}}, T_{\text{max}} \frac{1}{\xi}))$ output tokens if $B_{\text{aux}} = 0$ and $O(\min(N_{\text{out}}, B_{\text{write}}, T_{\text{max}} \frac{1}{\lambda}))$ output tokens otherwise.

Proof. The high-level idea of the proof is to simulate each of the algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_C$ over several cluster vertices by using the fact that the state (which, by definition, has size polynomial in $L = O(\text{polylog } n)$) of each algorithm $\mathcal{A}_j$ can be sent from one vertex in $V_C$ to another in $n^{o(1)}$ rounds, using Theorem 6.

The simulation that processes the stream $S_j$ of algorithm $\mathcal{A}_j$ is coordinated by a $\left(\frac{\xi}{\lambda}, V_C\right)$-vertex chain $X_j$ (Definition 10) that contains $\lambda$ vertices; we call these chains simulator chains, and we make sure that they are disjoint. The simulator chain $X_j$ collects all main tokens for algorithm $\mathcal{A}_j$ so that, except for the at most $B_{\text{aux}}$ occasions when the algorithm invokes $\text{GET-AUX}$, $\mathcal{A}_j$ can be simulated entirely by $X_j$. The protocol proceeds in phases, as follows.

**Phase 0:** Assigning simulator chains in $0$ rounds. All vertices in $V_C$ locally assign vertices to simulator chains $X_j$ for each algorithm $\mathcal{A}_j$ in an identical and predetermined manner. Each vertex is assigned to some position in a simulator chain at most once and the simulator chains are disjoint. This is possible because the parameter $\beta$ of each chain is $\beta = \frac{k}{\lambda}$ and therefore contains $\lambda$ vertices, and $\lambda$ satisfies $\lambda \leq \frac{k}{\xi} \leq k$.

**Phase 1:** Sending stream information in $\frac{T_{\text{max}}}{\xi} \left(\frac{\xi}{\lambda} \cdot n^{o(1)}\right)$ rounds. For each algorithm $\mathcal{A}_j$, each simulator vertex $v = X_j[i]$ receives the main tokens $\tau$ from $S_j$ held by all $u \in f_{X_j[i]}(v)$. Since the number of main tokens in $\sigma_{j,v}$ at most $T_{\text{max}}$, each vertex sends at most $\xi T_{\text{max}}$ during this phase, and each simulator vertex receives at most $\frac{k}{\lambda} T_{\text{max}}$ tokens. Since each token has length $L = O(\text{polylog } n)$, by Theorem 6 all of the main tokens $\tau$ can be sent and received in $\left(\frac{T_{\text{max}}}{\xi} + \frac{\xi}{\lambda} \cdot n^{o(1)}\right)$ rounds.

At the end of this phase, each main token $\tau$ in the stream $S_j$ is known by some $v \in X_j$. Additionally, because of the input contiguity property of Definition 9, the stream $S_j$ can be partitioned into $\lambda$ contiguous intervals $s_{j,1}, \ldots, s_{j,\lambda}$, where $s_{j,i} = \bigcup_{u \in f_{X_j[i]}(X_j)} \sigma_{j,u}$, so that each $X_j[i]$ knows the main tokens $\tau$ in $s_{j,i}$. Because the indices of the endpoints of the intervals $\sigma_{j,u}$ may not be known, it is possible that the indices of the endpoints of $s_{j,i}$ also are not known even to $X_j[i]$ — however, $X_j[i]$ knows all $\tau \in s_{j,i}$, knows that this set of tokens is guaranteed to be contiguous, and knows the identities of the other members of $X_j$. 


Phase 2: Simulating in \((B_{aux} + 1)(\lambda + \frac{\omega}{\delta}) \cdot n^{o(1)}\) rounds.

The simulator chain \(X_j\) can now begin simulating \(A_j\). Initially, all \(v = X_j[i]\) for \(i > 1\) begin in an inactive state, in which they wait to be activated by the previous vertex \(X_j[i - 1]\). The vertices \(X_j[1]\) are activated as soon as Phase 1 finishes and locally construct the starting state of the algorithm \(A_j\). Upon activation, the simulator vertex \(X_j[1]\) behaves as follows:

- If the next operation of \(A_j\) is \(\text{READ}\) on a main token \(\tau_j\), \(X_j[1]\) simulates \(A_j\) using the value of \(\tau_j\) if it is known (i.e., if \(\tau_j \in s_j\)). If it is not known, \(X_j[1]\) sends the state of \(A_j\) to the next simulator vertex \(X_j[i+1]\) and activates it.
- If the next operation of \(A_j\) is \(\text{GET-AUX}\) on the auxiliary tokens \(a_t\), \(X_j[1]\) sends the state of \(A_j\) to the vertex \(v\) that initially held \(\tau_j\). Vertex \(v\) simulates \(A_j\) until \(\text{READ}\) is performed on the next main token \(\tau_{j+1}\), which it can do since it holds all needed auxiliary tokens. Then, \(v\) returns the updated state of \(A_j\) to \(X_j[i]\). Vertex \(v\) does not return output tokens emitted by the simulated algorithm; instead, it stores these locally.
- If the next operation of \(A_j\) is \(\text{WRITE}\) on token \(\rho_t\), vertex \(X_j[i]\) simulates \(A_j\) by storing \(\rho_t\) locally.

During the simulation, most congestion occurs around vertices that receive many requests to simulate \(\text{GET-AUX}\). The total number of \(\text{GET-AUX}\) operations is \(O(\xi B_{aux})\), thus we hope to handle them in \(\frac{\xi B_{aux}}{\delta} \cdot n^{o(1)}\) rounds. However, a \(v \in V'_C\) can receive up to \(\xi B_{aux}\) requests, thus if \(v\) sends responses arbitrarily, the request could take \(\frac{\xi B_{aux}}{\delta} \cdot n^{o(1)}\) rounds to answer. This is because a vertex saturated with such requests may simulate \(O(\xi B_{aux})\) of a large fraction of \(\text{GET-AUX}\) operations of other algorithms before processing even the first \(\text{GET-AUX}\) of some \(A_j\). As \(A_j\) cannot continue until it receives the response, \(A_j\) may be delayed during a large fraction of its \(B_{aux}\) requests to simulate \(\text{GET-AUX}\) operations and expend \(o(\frac{\xi B_{aux}}{\delta})\) rounds waiting. This would increase the \((B_{aux} + 1)(\lambda + \frac{\omega}{\delta}) \cdot n^{o(1)}\) term of the complexity by a factor of \(B_{aux}\). Thus we sequence the simulation of \(\text{GET-AUX}\) operations such that the first \(\text{GET-AUX}\) of every algorithm is complete before vertices begin the second \(\text{GET-AUX}\) of some algorithms, ensuring delays cannot accumulate.

More precisely, we split Phase 2 into \(B_{aux} + 1\) steps of \((\lambda + \frac{\omega}{\delta}) \cdot n^{o(1)}\) rounds each. Within each step, the simulation of each algorithm proceeds until the next \(\text{GET-AUX}\) is performed on some sequence of auxiliary tokens \(a_t\) (or until the end of the algorithm, if there are no more invocations of \(\text{GET-AUX}\)), then pauses until the beginning of the next step. During step \(t\) for \(1 \leq t \leq B_{aux}\), the following actions are performed:

1. Any vertex that received a request to simulate \(A_j\) invoking \(\text{GET-AUX}\) on \(a_t\) performs the simulation until \(\text{READ}\) is invoked on the next main token \(\tau_{t+1}\), then sends the updated state of \(A_j\) to the appropriate vertex in \(X_j\).
2. Upon receiving the response, each chain \(X_j\) continues simulating \(A_j\) until the next \(\text{GET-AUX}\) is invoked on a sequence of auxiliary tokens \(a_{\tau'}\). Then \(X_j\) sends the current state of \(A_j\) to the vertex that holds \(a_{\tau'}\).

In each step and for each algorithm, \(v \in V'_C\) responds to and receives at most one request to process a sequence of auxiliary tokens, thus these can be delivered in \(\frac{\omega}{\delta} \cdot n^{o(1)}\) rounds. Additionally, the time required to simulate \(A_j\) before it reads the next auxiliary token is \(\lambda \cdot n^{o(1)}\), as the only communication required is propagation of the state of \(A_j\) through the at most \(\lambda\) vertices in \(X_j\). Phase 2 overall therefore finishes in \((B_{aux} + 1)(\lambda + \frac{\omega}{\delta}) \cdot n^{o(1)}\) rounds.

To prove the distribution of output tokens satisfies the guarantees of the theorem, consider the \(O(N_{out})\) output tokens of \(A_j\). Each \(v = X_j[i]\) knows \(O(\text{READ})\) tokens for each \(\tau \in s_j\). Thus, if \(B_{aux} \neq 0\), any vertex that processed auxiliary tokens holds \(O(\text{READ})\) tokens for each of the \(O(T_{\text{max}}\xi)\) sequences of auxiliary tokens processed. In both cases, a vertex knows at most all \(O(N_{out})\) output tokens for a given \(A_j\). So the number of output tokens held by any vertex is \(O(\min\{N_{out}, B_{aux}T_{\text{max}}\frac{\xi}{\lambda}\})\) if \(B_{aux} = 0\). Otherwise, it is \(O(\min\{\xi N_{out}, B_{aux}T_{\text{max}}\frac{\xi}{\lambda}, \xi\})\), which, since \(\xi\) is bounded by \(\frac{\omega}{\delta}\), is \(O(\min\{\xi N_{out}, B_{aux}T_{\max}\frac{\xi}{\lambda}\})\).

4 PARTITION TREES

We describe two classes of partition trees. Due to the low bandwidth of \(\text{Congest}\) and the need to incorporate information from vertices outside high-conductance expanders, our partition trees require stronger balancing properties than those initially presented in [7] for \(\text{Congest}\) \(\text{CLIQUE}\). We demonstrate that, within a suitable communication cluster, the partition tree needed to list all instances of \(K_p, p \geq 3\), can be constructed in \(n^{1/2+o(1)}\) rounds.

Definition 12 ([7], \(p\)-partition tree). Let \(G = (V,E)\) be a graph with \(n = |V|, m = |E|\), and let \(p \leq \log n\). A \(p\)-partition tree \(T = T_{G,p}\) is a tree of \(p\) layers (depth \(p - 1\)), where each non-leaf node has at most \(x = n^{1/p}\) children. Each tree node is associated with a partition of \(V\) consisting of at most \(x\) parts.

The partition associated with the root of \(T\) is denoted \(U_0\). Given a node with partition \(U_{(i_1,\ldots,i_{p-1})}\), the partition associated with its \(j\)th child, for \(0 \leq j \leq x - 1\), is \(U_{(i_1,\ldots,i_{p-1},j)}\).

The parts of each \(U_{(i_1,\ldots,i_{p-1})}\) are denoted \(U_{(i_1,\ldots,i_{p-1},j)}\), for \(0 \leq j \leq x - 1\). For each \(0 \leq j \leq x - 1\), also denote \(\text{parent}(U_{(i_1,\ldots,i_{p-1},j)}) = U_{(i_1,\ldots,i_{p-1},j)}\) and recursively define the set of ancestor parts for \(S = (i_1,\ldots,i_{p-1},j)\)

\[
\text{anc}(U_{S,j}) = \begin{cases} 
\text{anc}(\text{parent}(U_{S,j})) \cup \{U_{S,j}\} & S \neq \emptyset \\
\{U_{S,j}\} & S = \emptyset
\end{cases}
\]

The following distributes the work of detecting all instances of a subgraph \(H\) by assigning the task of learning all edges between parts in \(\text{anc}(U_{S,j})\) for each leaf part \(U_{S,j}\) to some vertex. The idea of the proof is to consider an instance \(H'\) of \(H\) in \(G\) with vertices \(v_0, v_1, \ldots, v_{p-1}\) and trace a path from the root of a \(p\)-partition tree \(T\) to the leaf layer, choosing first the part that contains \(v_0\), then the part that contains \(v_1\), etc. As each vertex is in a distinct part, each edge of \(H'\) must be present between some pair of distinct parts. We refer the reader to the proof of [7, Theorem 11].

Theorem 13. For any instance \(H'\) of a \(p\)-vertex subgraph \(H\) in \(G = (V,E)\) and any \(p\)-partition tree \(T\) of \(G\), there exists a leaf part \(U_{S,j}\) in \(T\) for which all edges of \(H'\) are contained in the union \(\cup_{U \in \text{anc}(U_{S,j})} \cup_{W \in \text{anc}(U_{S,j})} (U \neq W) E(U, W)\).
4.1 Partition Trees for $K_3$ Listing

Theorem 13 shows that the leaves of a $p$-partition tree can be used to distribute the work required for subgraph listing. In [7], $p$-partition trees are further constrained to ensure that the number of edges from each part $U_{k,j}$ to its ancestors is balanced to within an error term of at most $n$. However, in CONGEST, we are forced to reduce the permissible error to $n^{1-1/p}$; this error is represented by the $c_3 pk/\epsilon$ term of the $\updeG$ constraint below.

**Definition 14** ($H$-partition Tree). Let $G' = (V', E')$ be a subgraph of $G = (V, E)$ with $n = |V|$, $k = |V'|$, $m = |E'|$, and let $H = (V_H, E_H)$ be a graph with $p = |V_H| \leq \log n$ vertices $\{z_0, \ldots, z_{p-1}\}$, and denote $d_i = |\{(z_i, z_j) \in E_H \mid t < i\}|$ for each $0 \leq i \leq p-1$, $x = k^{1/p}$ and $\bar{m} = \max \{m, kx\}$. An $H$-partition tree $T = T_{G', H}$ is a $p$-partition tree with the following additional constraints, for some constants $c_1, c_2, c_3$:

1. **$DEG$**: For every part $U_{k,j}$, it holds that $|E(U_{k,j}, V')| \leq c_1 \bar{m}/x$.
2. **$\updeG$**: For every part $U_{k,j}$, and for all of its ancestor parts $W \in \anc(U_{k,j})$,
   \[ \sum_{W \in \anc(U_{k,j})} \frac{|E(U_{k,j}, W)|}{|E(W)|} \leq c_2 d_i m / x^2 + c_3 \bar{m}/x. \]
3. **$SIZ$**: For each part $U_{k,j}$, it holds that $|U_{k,j}| \leq c_3 k/x$.

We use the simulation in Section 3 to efficiently construct these trees in $(\phi, \delta)$-communication clusters (Definition 7) with suitable $\phi$ and $\delta$. As before, for $C$ and its corresponding $V_C$, denote $n = |V|$, $k = |V'|$, $K = |V_C|$. Let $\mu$ be the average communication degree of vertices in $V_C^-$ and $V_C^+$ be the set of $v \in V_C$ with communication degree at least $\mu/2$.

**Definition 15** ($K_3$-Compatible Cluster). A $K_3$-compatible cluster is a $(\phi, \delta)$-communication cluster $C = (V_C, E_C)$ in $G = (V, E)$ for $\phi = O(\polylog n)$ and $\delta = K^{1/3}$.

**Theorem 16.** Given a $K_3$-compatible cluster $C = (V_C, E_C)$, there exists a deterministic CONGEST algorithm on $C$ that, in $k^{1/3} \cdot n^{o(1)}$ rounds, constructs a $K_3$-partition tree $T$ of $C[V_C^+]$, such that:

- The root and first layer of $T$ are known to all vertices in $V_C^-$.
- Each vertex $v \in V_C^-$ knows $O(\frac{1}{\mu} \deg_G(v))$ parts of the leaf layer of $T$.
- Each part of the leaf layer is known to some vertex $v \in V_C^-$.

We prove there is a partial-pass streaming algorithm for constructing one layer of an $H$-partition tree an simulate it using Theorem 11 to construct one layer of a $K_3$-partition tree of $C(V_C^+)$. A proof of the following is in the full version [12].

**Lemma 17.** There exist constants $c_1, c_2, c_3$ such that, given a $(\phi, \delta)$-communication cluster $C = (V_C, E_C)$ in graph $G = (V, E)$ for $\phi = O(\polylog n)$ and $\delta = K^{1/3}$, where each $v \in V_C^-$ knows the first $0 \leq i < 3$ layers of some $K_3$-partition tree $T_{C, K_3}$, with constants $c_1, c_2, c_3$, there exists a deterministic CONGEST algorithm that finishes in $k^{1/3} \cdot n^{o(1)}$ rounds and constructs a next layer for the tree $T_{C, K_3}$ so that each endpoint of a part of the next layer is known to some vertex $v \in V_C^-$ and each $v \in V_C^-$ knows $O(k^{1/3})$ endpoints of parts.

**Proof Sketch.** We first outline a partial-pass streaming algorithm that, for some part $U_{k,j}$ in the lowest known layer of the tree, takes as input a stream containing the degrees of each vertex into each part in $\anc(U_{k,j})$. The algorithm starts with an empty part and processes vertices $v$ one at a time, adding vertices to the current part and maintaining counters for each of the balancing constrains $DEG, \updeG,$ and $SIZE$ in the $K_3$-partition tree definition. Each time the algorithm reads the degrees of a new vertex $v$, it includes $v$ in the current part and adds its degrees to the corresponding counters. If some counter would exceed the maximum value given in the $K_3$-partition tree definition, the current part is outputted and $v$ is instead added to a new part.

We can see that this algorithm maintains a small state — just the values in the counters and the endpoints of the range of vertices contained in the current part — and only reads the input stream once. This algorithm is therefore a partial-pass streaming algorithm with parameters $L = O(\polylog n)$, $N_m = \tilde{O}(k)$, $N_{out} = O(k^{1/3})$, $B_{aux} = 0$, $B_{write} = N_{out}$, so that it can be simulated in $k^{1/3}n^{o(1)}$ rounds using the parameters $T_{max} = O(1), \lambda = k^{1/3}$ in Theorem 11. □

**Load balancing the output.** The distribution of the output tokens of the partial-pass streaming algorithm as simulated with Theorem 11 is very loosely constrained, so we redistribute the output. After applying Lemma 17 to construct a new layer of a $K_3$-partition tree, each part of the layer is known only to one vertex, and each vertex knows $O(k^{1/3})$ parts. In order to apply Lemma 17 again to construct the next layer, and in order to satisfy the guarantees of Theorem 16, we distribute the root and middle layers of the $K_3$-partition tree so they are known by all vertices in $V_C$ in $k^{1/3} \cdot n^{o(1)}$ rounds using Lemma 19 from the full version [12]. Likewise, to satisfy the guarantees of Theorem 16, we distribute the leaf layer so that each $v \in V_C$ knows $O(\frac{1}{\mu} \deg_G(v))$ parts using Lemma 18.

Together, these prove Theorem 16: Apply Lemma 17 to build the root partition of a $K_3$-partition tree $T$. Then make it known to all $V_C$ in $k^{1/3} \cdot n^{o(1)}$ rounds. Then, again invoke Lemma 17 to build and distribute the middle layer of $T$ in $k^{1/3} \cdot n^{o(1)}$ rounds. Finally, Lemma 17 builds the leaf layer of $T$ in $k^{1/3} \cdot n^{o(1)}$ rounds and Lemma 18 distributes it according to the guarantees of Theorem 16.

It remains to show Lemma 18. In the following, we show how $K_3$-compatible clusters distribute $O(k)$ messages so that each $v \in V_C^-$ knows $O(\frac{1}{\mu} \deg_G(v))$ messages. We present a partial-pass streaming algorithm whose input is the degrees of vertices $v \in V_C$. For each $v \in V_C^-$, the algorithm outputs an interval of messages that $v$ is responsible for learning. We simulate the algorithm by Theorem 11 and redistribute the intervals so that each $v$ knows the number of the messages it is responsible for. For each such message, $v$ asks the vertex $u$ that holds that message for its contents.

**Lemma 18.** Given a $(\phi, \delta)$-communication cluster $C = (V_C, E_C)$ in graph $G = (V, E)$ for $\phi = O(\polylog n)$, $\delta = K^{1/3}$, $K = |V_C|$, $k = |V_C|$, and $O(k)$ numbered messages distributed so that each message is initially known by exactly one $v \in V_C^-$ and each $v \in V_C^-$ initially knows $O(k^{1/3})$ messages, there is a deterministic CONGEST algorithm that finishes in $k^{1/3} \cdot n^{o(1)}$ and redistributes the messages so that each $v \in V_C^-$ knows $O(\frac{1}{\mu} \deg_G(v))$ messages.

**Proof.** By Theorem 3, the diameter of $C$ is $O(\polylog n)$, thus all $V_C^-$ can know $m = |E(V_C^-, V_C^+)|$, the average $\mu = 2m/k$, and the total number of messages $M$ in $O(1)$ rounds. Let $e = |M|$. As $M = O(k)$, then $e = O(1)$. We deterministically assign at most $c$ messages to each $v \in V_C^-$ in a way that is locally computed by
all vertices — message $m_j$ is assigned to the vertex with number $j/c$. Then each vertex sends each message $m_j$ it originally holds to vertex $j/c$. Each vertex sends $O(k^{1/3})$ messages and receives $O(1)$ messages, so this process requires $n^{o(1)}$ rounds.

We present a simple partial-pass streaming algorithm (Algorithm 1) that, given an input stream $S$ containing the degrees of all $v \in V'_C$, produces an output stream $R$, containing the endpoints of a range of messages to be delivered to $v$ for each $v \in V'_C$. All messages are assigned to some $v \in V'_C$ in this way and each $v \in V'_C$ receives $O(\frac{1}{k} \deg_C(v))$ messages. Each input token is a tuple $(v, \deg_C(v))$ and each output token is a tuple $(v, l)$, where $l$ is $\emptyset$ or some $[a, b]$. The total communication degree of $v$ is $\in L$ input token, and so is polylogarithmic in each $v$ that.

| Algorithm 1 Balance messages by communication degree |
|-----------------------------------------------------|
| 1. leaf ← 0 |
| 2. for $t_l \in S$ do |
| 3. READ $t_l = (v_l, \deg_C(v_l))$ |
| 4. if $\deg_C(v_l) < \mu/2$ then |
| 5. WRITE $(v, \emptyset)$ |
| 6. else |
| 7. $l \leftarrow 2[M \deg_C(v_l)/m]$ |
| 8. WRITE $(v, \lfloor \mu + 1 \rfloor)$ |
| 9. leaf ← leaf + 1 |

The algorithm allocates at most $2[M \deg_C(v)/m]$ messages to each $v \in V'_C$. Since $M = O(k)$ and $m = E(V'_C, V'_C)$, it holds that $M/m = O(1/\mu)$, so that the number of messages delegated to each $v \in V'_C$ is $O(\frac{1}{k} \deg_C(v))$. All messages are allocated in this way, as $\sum_{v \in V'_C} 2 \deg_C(v) \geq k \mu$. Since $m = k \mu$, multiplying both sides by $M$ and rearranging gives

$$\sum_{v \in V'_C} \frac{2 \deg_C(v) \cdot M}{m} \geq M.$$

The total communication degree of $V'_C$ is at least half $k \mu$ as $E(V'_C, V'_C) \geq E(V'_C, V'_C)$ and we can show this by contradiction. Let $V'_C = V'_C \setminus V'_C$ be the set of low degree vertices and instead $E(V'_C, V'_C) < E(V'_C, V'_C)$. Then,

$$|E(V'_C, V'_C)| \leq |E(V'_C, V'_C)| + |E(V'_C, V'_C)| < 2|E(V'_C, V'_C)| <$$

$$< 2 \sum_{v \in V'_C} \frac{1}{2} \mu |V'_C| \mu |V'_C| \mu = |E(V'_C, V'_C)|,$$

which is a contradiction. So we must have $E(V'_C, V'_C) \geq E(V'_C, V'_C)$, from which $E(V'_C, V'_C) \geq \frac{1}{k} k \mu$ as desired.

We simulate this partial-pass streaming algorithm using Theorem 11. First, note that the token length is $L = O(\log n)$, as a token contains $O(1)$ numbers, each of which is $O(n)$. Additionally, the state of the algorithm is the numbers $l$ and $\frac{1}{k}$ input token, and so is polylogarithmic in $L$. Since the input and output streams both contain one token for each $v \in V'_C$, we get that $N_{in}, N_{out} = O(k)$. Each vertex initially holds its own input token, so $T_{Max} = 1$. The algorithm never performs an GET-AUX operation, so $B_{Write} = 0$. Finally, $B_{Write} = 1$, because the algorithm outputs exactly one token between READ operations. Thus, using $\lambda = k^{1/3}$, the simulation finishes in $(\frac{1}{k^{1/3}} (1 + k^{1/3}))^n = k^{1/3} \cdot n^{o(1)}$ rounds. At the end of the simulation, each $v \in V'_C$ knows $O(B_{Write} T_{Max}^2) = O(k^{2/3})$ output tokens.

We now redistribute the messages. First, each $v \in V'_C$ sends each of the output tokens it holds to the vertex whose assignment is contained in the token. Each vertex sends $O(k^{2/3})$ messages and receives $O(1)$, as $\delta = k^{2/3} \geq k^{1/3}$, this takes $k^{1/3} \cdot n^{o(1)}$ rounds. Next, each $v \in V'_C$ upon receiving the endpoints of its interval $[a, b]$ of message numbers, for each $j \in [a, b]$ sends a request to vertex $j/c$ to obtain message $m_j$. Each $v \in V'_C$ sends $O(\frac{1}{k} \deg_C(v))$ messages and receives $O(1)$ messages, so this can be done in $n^{o(1)}$ rounds. The entire process takes $k^{1/3} \cdot n^{o(1)}$ rounds, as desired.

4.2 Partition Trees for $K_p$ Listing, $p > 3$

For $p > 3$, we are less constrained by the cluster bandwidth as our target listing time is higher. Thus, we can adopt an error term of $n$ (instead of the $n^{1/3}$ term employed in the case of $p = 3$) on the edge-balancing upper bounds. However, because in $K_p$-listing algorithms for $p > 3$ a cluster $C$ can be responsible for listing cliques with some vertices in $V \setminus V_C$, we must adapt the partition trees to simultaneously balance the distribution of three types of edges: edges that lie entirely inside $C$, edges that lie entirely outside $C$, and edges that cross the boundary of $C$ (with one vertex in $C$ and the other outside). Definition 19 describes these three edge sets, and Definition 20 describes a new class of partition tree with these more complex balancing properties.

Definition 19 (Split Graph). A split graph is a graph $G = (V, E)$, together with disjoint sets $V_1, V_2, E_1, E_2$ such that $V = V_1 \cup V_2$, $E = E_1 \cup E_2$, and $E_1 \cup E_2 \cup V_1 = V_2 \cup V_2$, and $E_1 \cup E_2 \cup V_1 = V_2 \cup V_2$. Denote $n = |V|$, $k = |V_1|$, $m_1 = |E_1|$, $m_2 = |E_2|$, and $m_12 = |E_12|$.

We construct a partition tree over a split graph, where each vertex is a partition of either $V_1$ or $V_2$. The constraints $\text{DEG}_{2to2}, \text{UP}_2, \text{DEG}_{1to2}, \text{DEG}_{1to1}, \text{UP}_2, \text{DEG}_{2to1}, \text{UP}_2, \text{DEG}_{1to1}, \text{UP}_2$ provide additional balancing for partitions of different vertex sets.

Definition 20 ((p′, p)-Split $K_p$-Tree). Let $G$ be a split graph, and let $1 < a < b < n$ and $1 \leq p' < p < \log n$ be some integer parameters. Define $m_1 = \max(m_1, ka), m_2 = \max(m_2, nb)$, and $m_{12} = \max(m_{12}, na)$.

A $(p', p)$-split partition tree $T$ is a tree of $p$ layers (depth $p - 1$), where each non-leaf node in the first $p - p'$ layers has at most $a$ children and is associated with a partition of $V_2$ into at most $b$ parts, and each non-leaf node in the bottom $p'$ layers has at most $a$ children and is associated with a partition of $V_1$ into at most $b$ parts. The partitions $U_S$, the parent listing $U_S$, and set of parts $\text{anc}(U_{S, j})$ are defined inductively as in Definition 12.

A $(p', p)$-split $K_p$-partition tree is a $(p', p)$-split partition tree with these constraints for $\pi = p' - p$ and values $c_1, c_2$.

1. $\text{DEG}_{2to2}$: For every part $U_{S,j}$, $|S| < \pi$, it holds that $\left| E(U_{S,j}, V_2) \right| \leq c_1 m_2 b + n$.
2. $\text{UP}_2$-to-$2$: For every part $U_{S,j}$, $|S| < \pi$, and for all of its ancestor parts $U_{S,j'} \in \text{anc}(U_{S,j})$, it holds that $\sum_{S | S' \in \text{anc}(U_{S,j}) \setminus \text{anc}(U_{S,j})} |E(U_{S,j}, V_{S,j})| \leq c_2 |S| m_2 b + n$. 

278
(3) $\textsc{Deg}_2to1$: For every part $U_{S,j}$, $|S| < \pi$, it holds that $|E(U_{S,j}, V_i)| \leq c_1 m_12 / b + n$.

(4) $\textsc{Deg}_1to1$: For every part $U_{S,j}$, $|S| \geq \pi$, it holds that $|E(U_{S,j}, V_i)| \leq c_1 m_1 / a + k$.

(5) $\textsc{Up Deg}_1to1$: For every part $U_{S,j}$, $|S| \geq \pi$, and for all of its ancestor parts $U_{S',j'} \in \text{anc}(U_{S,j})$, $|S'| \geq \pi$, it holds that $\sum_{S'' \in \text{ancec}(U_{S,j})} |E(U_{S,j}, U_{S'',j'})| \leq c_2 (|S| - \pi) m_1 / a^2 + n$.

(6) $\textsc{Up Deg}_1to2$: For every part $U_{S,j}$, $|S| \geq \pi$, and for all of its ancestor parts $U_{S',j'} \in \text{anc}(U_{S,j})$, $|S'| < \pi$, it holds that $\sum_{S'' \in \text{ancec}(U_{S,j}) : |S''| < \pi} |E(U_{S,j}, U_{S'',j'})| \leq c_2 \pi m_1 / (ab) + n$.

The following is an analogue of Theorem 13 for $(p, p')$-split partition trees, and shows they can load balance the listing of all instances of a subgraph $H$ in a split graph $G$ with $p' \geq 2$ vertices in $V_1$. Consider an instance $H'$ of $H$ in $G$ with vertices $v_0, v_1, \ldots, v_{p-1}$, where the first $p - p'$ vertices are in $V_2$ and the remaining $p'$ are in $V_1$, and trace a path from the root of a $(p, p')$-split partition tree $T$ to the leaf layer, choosing first the part that contains $v_0$, then the part with $v_1$, etc. As each vertex is in a distinct part, each edge of $H'$ must be present between some pair of distinct parts.

**Theorem 21.** For any instance $K^p_G$ of a $K_p$ in a split graph $G$, where $K^p_G$ has at least $p' \geq 2$ vertices in $V_1$, and any $(p, p')$-split partition tree $T$ of $G$, there exists a leaf part $U_{S,j}$ in $T$ for which all edges of $K^p_G$ are contained in $\bigcup_{U \in \text{anc}(U_{S,j})} \bigcup_{W \in \text{ancec}(U_{S,j})} W^1 E(U, W)$.

To list within clusters, we generalize $K_3$-compatible clusters. A cluster $C$ can be responsible for listing cliques that do not lie entirely inside $C$, and so receives input about edges not incident to vertices in $C$. Therefore, in addition to the constraint that all elements of $V^*_C$ have communication degree at least $n^{1-2/p}$, Definition 22 details the necessary input and the constraints about its distribution. These constraints are easy to satisfy, but do not easily permit constructing $(p, p')$-split $K_p$-partition trees. Thus, Definition 23 gives stricter constraints on the input distribution within clusters. We show clusters can rearrange the input given in Definition 22 to satisfy the conditions of Definition 23, and then construct a $(p, p')$-split $K_p$-partition tree using Theorem 11.

**Definition 22 ($K_p$-Compatible Cluster).** A $(p, p')$-compatible cluster for $p > 3$ is a set $\phi$ of $(\delta, \delta)$-communication clusters $C = (V_C, E_C)$ in $G = (V, E)$ for $\delta = O(\text{polylog } n)$ and $\delta = n^{1-2/p}$. The cluster holds two sets of directed edges $\bar{E} \subseteq E(V(V_C, V_{C'}))$ and $E' \subseteq E(V(V_C, V_{C'}))$. Define the input degree $\deg_C(v)$ of $v$ to be the number of edges in $\bar{E} \cup E'$ with tail $v$, $\deg_C(v) = |\{u \in V : (v, u) \in \bar{E} \cup E'\}|$. $\bar{E}$ and $E'$ are distributed to $V^*_C$ as follows:

- Any vertex $v \in V_{C'}$ is incident to $O(\deg_C(v))$ edges from $\bar{E}$ and knows all such edges $(u, v)$.
- Each vertex $u \in V_{C'}$ receives $O(n^{1-2/p} \cdot \deg_C(u))$ edges from $E'$. Additionally, $\mu \in \Omega \left( \frac{|E'|}{n^2} \right)$.
- For each $u \in V$ that is the tail of at least one $e \in \bar{E} \cup E'$, exactly one $v \in V_{C'}$ holds $\deg_C(u)$.

**Definition 23 ($K_p$-Input Cluster).** A $(p, p')$-input cluster $C$ is a $K_p$-compatible cluster in which $\bar{E}$ and $E'$ are distributed to $V^*_C$ in the following manner. There is a $(\beta, V)$-vertex chain $E = V_1'$ with $\beta = n$, whose elements $E[1], \ldots, E[k]$ are numbered in increasing order. Each $v = E[i]$, for each $u \in E^{-1}(v)$ knows all edges $e \in E' \cup \bar{E}$ with their tail at $u$, and there are $\tilde{O}(n^{1-2/p} \cdot \deg_C(v))$ such edges.

We now state the main result of this section, that we can construct a $(p, p')$-split $K_p$-partition tree.

**Theorem 24.** Given a $K_p$-compatible cluster $C = (V_C, E_C)$ and constants $p, p' \leq p, p' > 3$, there is a deterministic CONGEST algorithm on $C$ that, in $n^{1-2/p}(1)$ rounds, constructs a $(p', p')$-split $K_p$-partition tree $T$ for $V_1 = V_{C'}$, $V_2 = V \setminus V_{C'}$, $E_1 = E(V_C, V_{C'})$, $E_2 = E'$, and $E_{12} = \tilde{E}$ for $a, b = O(k^{1/p})$ so that all parts in $T$ are known to all of $V_{C'}$.

We split the proof to two parts. Theorem 25 shows how a $K_p$-input cluster constructs the partition using Theorem 11. The full version [12] shows how a $K_p$-compatible cluster reorganizes itself into a $K_p$-input cluster.

**Theorem 25.** Given a $K_p$-input cluster $C = (V_C, E_C)$ and constants $p, p' \leq p, p' > 3$, there is a deterministic CONGEST algorithm on $C$ that, in $n^{1-2/p}(1)$ rounds, constructs a $(p', p')$-split $K_p$-partition tree $T$ for $V_1 = V_{C'}$, $V_2 = V \setminus V_{C'}$, $E_1 = E(V_C, V_{C'})$, $E_2 = E'$, and $E_{12} = \tilde{E}$ for $a, b = O(k^{1/p})$ so that all parts in $T$ are known to all of $V_{C'}$.

We show a partial-pass streaming algorithm constructing a layer of a $(p, p')$-split $K_p$ partition tree. We use auxiliary tokens, as for $p > 3, v \in V_{C'}$ can be responsible for edges in $\tilde{E} \cup E'$ that emanate from a large number of $u \in V \setminus V_{C'}$. To obtain a good runtime with Theorem 11, we need a low $t_{\text{max}}$, i.e., vertices hold few main tokens. If we create a main token for each $u \in V$, some $u \in V_{C'}$ responsible for edges from a large number of $u \in V$ might hold too many main tokens. Therefore, we aggregate the information about all $u \in V$ for which $v \in V_{C'}$ is responsible into a single main token and encode the information about individual $u$ in auxiliary tokens.

**Lemma 26.** There are constants $c_1, c_2$ s.t., given a split graph $G$, some $2 \leq p' < p$, $0 \leq i < p$ levels of a $(p, p')$-split $K_p$-partition tree $T_{G,k_{(p_i),p}}$ for $c_1$, $c_2$, and a part $U_{S,j}$ in the $i$-th level of $T$, $i = |S|$, there is a partial-pass streaming algorithm with $L = O(\text{polylog } n)$, $N_m = O(n)$, $N_{out} = O(n^{1/p})$, $B_{aux} = O(n^{1/p})$, $B_{write} = N_{out}$ and streams $S$ and $R$ where:

- $R$ has vertex interval endpoints defining a valid child partition of $U_{S,j}$ in a $(p, p')$-split $K_p$-partition tree for $c_1$, $c_2$.
- Let $W = V_2$ if $i + 1 < \pi$ and $W = V_1$ otherwise. Let $\sigma$ be an arbitrary partition of $W$ into $k = |V_{C'}|$ contiguously-numbered intervals $\sigma_1, \ldots, \sigma_k$. The input stream $S$ contains $k$ tokens $\sigma_i$, $1 \leq i \leq k$, where the token $\sigma_i$ encodes $\sum_{e \in E} |E(\sigma_i, V_{\sigma_i})|$, $\sum_{e \in E} |E(\sigma_i, V_{C'})|$, and $\sum_{e \in E} |E(\sigma_i, V_{U'})|$ for each $U' \in \text{ancec}(U_{S,j})$. The tokens $\sigma_1, \ldots, \sigma_k |_{\sigma_i}$ contain $|E(\sigma_i, V_{\sigma_i})|$, $|E(\sigma_i, V_{C'})|$, and $|E(\sigma_i, V_{U'})|$ for each $v \in \sigma_i$ in order of increasing vertex number.

**Proof.** We show a counter-based algorithm (Algorithm 2) with counters for constraints $\text{DEG}_2to2, \text{UP DEG}_2to2, \text{DEG}_2to1$ if $i + 1 < \pi$ and $\text{DEG}_1to1, \text{UP DEG}_1to1, \text{UP DEG}_1to2$ otherwise.

We show that this algorithm satisfies Lemma 26. Each token contains $O(\log n)$ numbers, each polynomial in $n$, so $L = O(\text{polylog } n)$. As partition $\sigma$ has $O(n)$ parts, and there is one main token for each, $N_m = O(n)$. Each output token is a part of a partition in a $(p, p')$-split $K_p$-partition tree, so by Definition 20, $N_{out} = O(n^{1/p})$. The
Algorithm 2 Build one layer of a $(p, p')$-split $K_p$-partition tree

1: Set each counter to 0 and set $l ← 0$
2: for $τ_i ∈ S$ do
3: READ $τ_i$
4: Add the sums from $τ_i$ and to the appropriate counters
5: if any counter exceeds its maximum value then
6: Restore previous states of counters
7: GET-AUX $a_l$
8: for $a_{l,m} ∈ a_l$ do
9: READ $a_{l,m}$
10: $r ←$ the vertex number of $a_{l,m}$
11: Add sums from $a_{l,m}$ to the appropriate counters
12: if any counter exceeds its maximum value then
13: WRITE $[l, r = 1]$
14: Reset all counters to 0
15: Add each sum to the appropriate counter
16: $l ← r$
17: WRITE $[r, \text{max } W]$

algorithm only performs GET-AUX when a new part is formed, thus $B_{\text{aux}} = O(n^2/p)$. Finally, the total number of WRITE operations performed is $N_{\text{out}}$, the number of WRITE operations between any two READ operations is also bounded by $B_{\text{write}} = N_{\text{out}}$.

This constructs a valid child partition of $U_S$. A vertex can always join a new part on its own without exceeding the maximal value of any counter, due to the ‘+k’ in the bounds of $\text{deg}_G \cdot t_0$, $\text{UP}_G \cdot \text{deg}_G \cdot t_0$ and the ‘+n’ in those of the other counters. Starting a new part each overlap guarantees properties 1–6 of Definition 20. It remains to bound the number of times a counter exceeds its maximum value, e.g., a part closed by counter $\text{deg}_G \cdot 2t_0$ has at least $c_1 m_2/b$ edges into $V_S$, of which there are at most $2m_2$ total (counting each endpoint twice). There are at most $2b/c_1$ such parts. Similar analyses for the other counters in the full version [12] gives that if there exist constants $c_1, c_2$ satisfying $2a/c_1 + (c_1 + 1)a/c_2 + (c_1 + 1)a/c_2 < a$ and $2b/c_1 + (c_1 + 1)b/c_2 + 2b/p < h$, this produces a valid partition. Indeed, $c_1 = 8, c_2 = 36$ satisfy this property.

Theorem 11 can simulate the algorithm described in Lemma 26 in $n^{1/2 + o(1)}$ rounds, as stated and proven in the full version [12]. We apply it $p$ times for a total round complexity of $n^{1/2 + o(1)} p = n^{1/2 + o(1)}$, which proves Theorem 25.

Acknowledgements: This project was partially supported by the European Union’s Horizon 2020 Research and Innovation Programme under grant agreement no. 775839.

REFERENCES

[1] Amir Abboud, Keren Censor-Hillel, Seri Khoury, and Christoph Lenzen. 2020. Fooling views: a new lower bound technique for distributed computations under congestion. Distributed Computing 33, 6 (2020), 545–559.

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

[3] Zvika Brakerski and Boaz Patt-Shamir. 2011. Distributed discovery of large near-cliques. Distributed Computing 24, 2 (2011), 79–89.

[4] Keren Censor-Hillel. 2021. Distributed Subgraph Finding: Progress and Challenges. In ICALP, Vol. 198. 3:1–3:14.

[5] Keren Censor-Hillel, Yi-Jun Chang, François Le Gall, and Dean Leitersdorf. 2021. Tight Distributed Listing of Cliques. In SODA. 2878 – 2891.

[6] Keren Censor-Hillel, Eldar Fischer, Gregory Schwartzman, and Yadu Vasudev. 2018. Fast distributed algorithms for testing graph properties. Distributed Computing 32, 1 (2018), 41–57.

[7] Keren Censor-Hillel, Orr Fischer, Tzlil Gonen, François Le Gall, Dean Leitersdorf, and Rotem Oshman. 2020. Fast Distributed Algorithms for Girth, Cycles and Small Subgraphs. In DCC. 33:1–33:17.

[8] Keren Censor-Hillel, Orr Fischer, François Le Gall, Dean Leitersdorf, and Rotem Oshman. 2022. Quantum Distributed Algorithms for Detection of Cliques. ITCS.

[9] Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. 2016. Algebraic methods in the congested clique. Distributed Computing 32, 6 (2018), 461–478.

[10] Keren Censor-Hillel, François Le Gall, and Dean Leitersdorf. 2020. On Distributed Listing of Cliques. In PODC. 474–482.

[11] Keren Censor-Hillel, Dean Leitersdorf, and Elia Turner. 2020. Sparse matrix multiplication and triangle listing in the Congested Clique model. Theoretical Computer Science 809 (2020), 45–60.

[12] Keren Censor-Hillel, Dean Leitersdorf, and David Vulakh. 2022. Deterministic Near-Optimal Distributed Listing of Cliques. https://arxiv.org/abs/2205.09245.

[13] Yi-Jun Chang, Seth Pettie, Thatchaphol Saranurak, and Hongjie Zhang. 2021. Near-Optimal Distributed Triangle Enumeration via Expander Decompositions. J. ACM 68, 3, Article 21 (2021), 36 pages.

[14] Yi-Jun Chang and Thatchaphol Saranurak. 2019. Improved Distributed Expander Decomposition and Nearly Optimal Triangle Enumeration. In PODC. 66–73.

[15] Yi-Jun Chang and Thatchaphol Saranurak. 2020. Deterministic Near-Optimal Expander Decomposition and Routing with Applications in Distributed Derandomization. In FOCS. 377–388.

[16] Artur Czumaj and Christian Konrad. 2020. Detecting Cliques in CONGEST Networks. Distributed Computing 33, 5 (2020), 533–543.

[17] Danny Dolev, Christoph Lenzen, and Shir Poded. 2012. ‘Tri,Tri Again’: Finding Triangles and Small Subgraphs in a Distributed Setting. In DISC. 195–209.

[18] Andrew Drucker, Fabian Kuhn, and Rotem Oshman. 2014. On the power of the congested clique model. In PODC.

[19] Talya Eden, Nimrod Fiat, Orr Fischer, Fabian Kuhn, and Rotem Oshman. 2021. Sublinear-time distributed algorithms for detecting small cliques and even cycles. Distributed Computing (2021).

[20] Guy Even, Orr Fischer, Pierre Fraignaud, Tzlil Gonen, Reut Levi, Moti Medina, Pedro Montealegre, Dennis Olivetti, Rotem Oshman, Ivan Rapaport, and Ioan Todinca. 2017. Three Notes on Distributed Property Testing. In DISC. 15:1–15:30.

[21] Orr Fischer, Tzlil Gonen, Fabian Kuhn, and Rotem Oshman. 2018. Possibilities and Impossibilities for Distributed Subgraph Detection. In SPAA. 153–162.

[22] Pierre Fraignaud and Dennis Olivetti. 2019. Distributed Detection of Cycles. ACM Trans. Parallel Comput. 6, 3, Article 12 (October 2019), 20 pages.

[23] Pierre Fraignaud, Ivan Rapaport, Ville Salo, and Ioan Todinca. 2016. Distributed Testing of Excluded Subgraphs. In Distributed Computing. 342–356.

[24] Mohsen Ghaffarif and Fabian Kuhn, and Hsin-Hao Su. 2017. Distributed MST and Routing in Almost Mixing Time. In PODC. 131–140.

[25] Mohsen Ghaffarif and Jason Li. 2018. New Distributed Algorithms in Almost Mixing Time via Transformations from Parallel Algorithms. In DISC. 31:1–31:16.

[26] Tzlil Gonen and Rotem Oshman. 2018. Lower Bounds for Subgraph Detection in the CONGEST Model. In PODC. 61:1–61:6.

[27] Juho Hirvenen, Joel Rybicki, Stefan Schmid, and Jukka Suomela. 2017. Large Cuts with Local Algorithms on Triangle-Free Graphs. The Electronic Journal of Combinatorics 24, 4 (2017).

[28] Dawei Huang, Seth Pettie, Yisixiang Zhang, and Zhijun Zhang. 2021. The Communication Complexity of Set Intersection and Multiple Equality Testing. SIAM J. Comput. 50, 2 (2021), 674–717.

[29] Taisuke Izumi and François Le Gall. 2017. Triangle Finding and Listing in CONGEST Networks. In PODC. 381–389.

[30] Taisuke Izumi, François Le Gall, and Frédéric Magniez. 2020. Near-Optimal Distributed Triangle Finding in the CONGEST Model. STACS.

[31] Mark Jerrum and Alistair Sinclair. 1989. Approximating the Permanent. SIAM J. Comput. 18, 6 (1989), 1149–1178.

[32] Janne H. Korhonen and Joel Rybicki. 2018. Deterministic Subgraph Detection in Broadcast CONGEST. In PODS. 4:1–4:16.

[33] J Ian Munro and Mike Paterson. 1980. Selection and Sorting with Limited Storage. Theor. Comput. Sci. 12 (1980), 315–323.

[34] S. Muthukrishnan. 2005. Data Streams: Algorithms and Applications. Found. Trends Theor. Comput. Sci. 1, 2 (2005).

[35] Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. 2021. On the Distributed Complexity of Large-Scale Graph Computations. ACM Trans. Parallel Comput. 8, 2 (2021), 7:1–7:28.

[36] Seth Pettie and Hsin-Hao Su. 2013. Fast Distributed Coloring Algorithms for Triangle-Free Graphs. In ICALP. 681–693.