Distributed Triangle Detection via Expander Decomposition*

Yi-Jun Chang† Seth Pettie‡ Hengjie Zhang§

Abstract

We present improved distributed algorithms for triangle detection and its variants in the CONGEST model. We show that Triangle Detection, Counting, and Enumeration can be solved in $\tilde{O}(n^{1/2})$ rounds. In contrast, the previous state-of-the-art bounds for Triangle Detection and Enumeration were $\tilde{O}(n^{2/3})$ and $\tilde{O}(n^{3/4})$, respectively, due to Izumi and LeGall (PODC 2017).

The main technical novelty in this work is a distributed graph partitioning algorithm. We show that in $\tilde{O}(n^{1-\delta})$ rounds we can partition the edge set of the network $G = (V, E)$ into three parts $E = E_m \cup E_s \cup E_r$ such that

- Each connected component induced by E_m has minimum degree $\Omega(n^\delta)$ and conductance $\Omega(1/\text{polylog}(n))$. As a consequence the mixing time of a random walk within the component is $O(\text{polylog}(n))$.
- The subgraph induced by E_s has arboricity at most n^δ.
- $|E_r| \le |E|/6$.

All of our algorithms are based on the following generic framework, which we believe is of interest beyond this work. Roughly, we deal with the set E_s by an algorithm that is efficient for low-arboricity graphs, and deal with the set E_r using recursive calls. For each connected component induced by E_m, we are able to simulate CONGESTED-CLIQUE algorithms with small overhead by applying a routing algorithm due to Ghaffari, Kuhn, and Su (PODC 2017) for high conductance graphs.

1 Introduction

We consider Triangle Detection problems in distributed networks. In the LOCAL model [36], which has no limit on bandwidth, all variants of Triangle Detection can be solved in exactly *one* round of communication: every vertex v simply announces its neighborhood $N(v)$ to all neighbors. However, in models that take bandwidth into account, e.g., CONGEST, Triangle Detection becomes significantly more complicated. Whereas many graph optimization problems studied in the CONGEST model are intrinsically "global" (i.e., require at least diameter time) [2, 11, 12, 15, 16, 21, 27], Triangle Detection is somewhat unusual in that it can, in principle, be solved using only locally available information.

The CONGEST Model. The underlying distributed network is represented as an undirected graph $G = (V, E)$, where each vertex corresponds to a computational device, and each edge corresponds to a bi-directional communication link. It is common in literature to assume that each $v \in V$ initially knows some global parameters such as $n = |V|$, $\Delta = \max_{v \in V} \deg(v)$, and $D = \text{diameter}(G)$. In this paper, we only require each vertex to know $n = |V|$. Each vertex v has a distinct $\Theta(\log n)$-bit identifier $\text{ID}(v)$. The computation proceeds according to synchronized *rounds*. In each round, each vertex v can perform unlimited local computation, and may send a distinct $O(\log n)$-bit message to each of its neighbors.

Throughout the paper we only consider the randomized variant of CONGEST. Each vertex is allowed to generate unlimited local random bits, but there is no global randomness.

The Congested Clique Model. The CONGESTED-CLIQUE model is a variant of CONGEST that allows all-to-all communication. Each vertex initially knows its adjacent edges and the set of vertex IDs, which we can assume w.l.o.g. is $\{1, \ldots, |V|\}$. In each round, each vertex transmits $n - 1$ $O(\log n)$-bit messages, one addressed to each vertex in the graph.

Intuitively, the CONGEST model captures two constraints in distributed computing: *locality* and *bandwidth*, whereas the CONGESTED-CLIQUE model only focuses on the *bandwidth* constraint. This difference makes the two models behave very differently. For instance, the *minimum spanning tree* (MST) problem can be solved in $O(1)$ rounds in the CONGESTED-CLIQUE [24], but its round complexity is $\tilde{\Theta}(D + \sqrt{n})$ in CONGEST [37, 39].

One of the main reasons that some problems can be solved efficiently in CONGESTED-CLIQUE is due to the routing algorithm of Lenzen [31]. As long as each vertex v is the source and the destination of at most $O(n)$ messages, we can deliver all messages in $O(1)$ rounds. Using this routing algorithm [31] as a communication primitive, many parallel algorithms can be transformed to efficient CONGESTED-CLIQUE algorithms [6]. For example, consider the distributed matrix multiplication problem, where the input matrices are distributed to the vertices such that the ith vertex

*This work is supported by NSF grants CCF-1514383, CCF-1637546, and CCF-1815316.

†University of Michigan.
‡University of Michigan.
§IIIS, Tsinghua University.

initially knows the ith row. The problem can be solved in the CONGESTED-CLIQUE model in $\tilde{O}(n^{1/3})$ rounds over semirings, or $\tilde{O}(n^{1-(2/\omega)+o(1)}) = o(n^{0.158})$ rounds over rings [6].

Distributed Routing in Almost Mixing Time. A *uniform lazy random walk* moves a token around an undirected graph by iteratively applying the following process for some number of steps: with probability 1/2 the token stays at the current vertex and otherwise it moves to a uniformly random neighbor. In a connected graph $G = (V, E)$, the stationary distribution of a lazy random walk is $\pi(u) = \deg(u)/(2|E|)$. Informally, the *mixing time* $\tau_{\text{mix}}(G)$ of a connected graph G is the minimum number of lazy random walk steps needed to get within a negligible distance of the stationary distribution. Formally:

DEFINITION 1.1. (MIXING TIME [17]) *Let $p_t^s(v)$ be the probability that after t steps of a lazy random walk starting at s, the walk lands at v. The mixing time $\tau_{\text{mix}}(G)$ is the minimum t such that for all $s \in V$ and $v \in V$, we have $|p_t^s(v) - \pi(v)| \leq \pi(v)/|V|$.*

Ghaffari, Kuhn, and Su [17] proved that if each vertex v is the source and the destination of at most $O(\deg(v))$ messages, then all messages can be routed to their destinations in $\tau_{\text{mix}}(G) \cdot 2^{O(\sqrt{\log n \log\log n})}$ rounds. The $2^{O(\sqrt{\log n \log\log n})}$ factor has recently been improved [18] to $2^{O(\sqrt{\log n})}$. The implication of this result is that many problems that can be solved efficiently in the CONGESTED-CLIQUE can also be solved efficiently in CONGEST, *but only if $\tau_{\text{mix}}(G)$ is small.* In particular, MST can be solved in $\tau_{\text{mix}}(G) \cdot 2^{O(\sqrt{\log n})}$ rounds in CONGEST [18]. This shows that the $\tilde{\Omega}(\sqrt{n})$ lower bound [37, 39] can be bypassed in networks with small $\tau_{\text{mix}}(G)$.

At this point, a natural question to ask is whether or not this line of research [17, 18] can be extended to a broader class of graphs (that may have high $\tau_{\text{mix}}(G)$), or even general graphs. The main contribution of this paper is to show that this is in fact doable, and based on this approach we improve the state-of-the-art algorithms for triangle detection, counting, and enumeration.

Graph Partitioning. It is well known that *any* graph can be decomposed into connected components of conductance $\Omega(\epsilon/\log n)$ (and hence $\text{poly}(\epsilon^{-1}, \log n)$ mixing time) after removing at most an ϵ-fraction of the edges [5, 35, 41, 43]. Moshkovitz and Shapira [32] showed that this bound is essentially tight. In particular, removing any constant fraction ϵ of the edges, the remaining components have conductance at most $O((\log\log n)^2/\log n)$.

A slightly weaker version of this graph partition can be constructed in near-linear time (for fixed ϵ) in the sequential computation model [41]. Their algorithm uses random walks to explore the graph locally to find a cut with edge sparsity $O(1/\log n)$. If the output cut is S, then the time spent is $\tilde{O}(\text{Vol}(S))$.[1] By iteratively finding a sparse cut and removing it from the graph, in $\tilde{O}(|E|)$ time a graph partition is obtained in which all components have $\Omega(1/\text{polylog}(n))$ conductance.

This graph partition and the idea of local graph exploration have found many applications, such as solving linear systems [41], unique games [5, 38, 43], analysis of personalized PageRank [3], minimum cut [25], dynamic algorithms [33], and property testing [19, 29].

In this work, we show that a variant of this graph partition can be constructed efficiently in the CONGEST model. The new twist is to partition the edge set in *three* parts, rather than two (i.e., removed and remaining edges).

Distributed Triangle Detection. Many variants of the triangle detection problem have been studied in the literature [6, 22].

Triangle Detection. Each vertex v reports a bit b_v, and $\bigvee_v b_v = 1$ if and only if the graph contains a triangle.

Triangle Counting. Each vertex v reports a number t_v, and $\sum_v t_v$ is exactly the total number of triangles in the graph.

Triangle Enumeration. Each vertex v reports a list L_v of triangles, and $\bigcup_v L_v$ contains exactly those triangles in the graph.

Local Triangle Enumeration. It may be desirable that every triangle be reported by one of the three participating vertices. It is required that L_v only contain triangles involving v.

Dolev, Lenzen, and Peled [9, Remark 1] showed that Triangle Enumeration can be solved deterministically in $O(n^{1/3}/\log n)$ time in the CONGESTED-CLIQUE. Censor-Hillel et al. [6] presented an algorithm for Triangle Detection and Counting in CONGESTED-CLIQUE that takes $\tilde{O}(n^{1-(2/\omega)+o(1)}) = o(n^{0.158})$ time via a reduction to matrix multiplication. Izumi and LeGall [22] showed that in CONGEST, the Detection and Enumeration problems can be solved in $\tilde{O}(n^{2/3})$ and $\tilde{O}(n^{3/4})$ time, respectively. They also proved that in both CONGEST and CONGESTED-CLIQUE, the Enumeration problem requires $\Omega(n^{1/3}/\log n)$ time, improving an earlier $\Omega(n^{1/3}/\log^3 n)$ bound of Pandurangan et al. [34].

[1] By definition, $\text{Vol}(S) = \sum_{v \in S} \deg(v)$.

Izumi and LeGall [22] proved a large separation between the complexity of the Enumeration and Local Enumeration problems. If triangles must be reported by a participating vertex, $\Omega(n/\log n)$ time is necessary (and sufficient) in CONGEST/CONGESTED-CLIQUE. More generally, the lower bound on Local Enumeration is $\Omega(\Delta/\log n)$ when the maximum degree is Δ.

In this paper, we show that Triangle Detection, Enumeration, and Counting can be solved in $\tilde{O}(n^{1/2})$ time in CONGEST. This result is achieved by a combination of our new distributed graph partition algorithm, the multi-commodity routing of [17,18], and a randomized version of the CONGESTED-CLIQUE algorithm for Triangle Enumeration of [6,9]. We also show that when the input graph has high conductance/low mixing time, that Triangle Enumeration can be solved even faster, in $O(\tau_{\mathrm{mix}}(G)n^{1/3+o(1)})$ time.

1.1 Technical Overview Consider a graph $G = (V, E)$. For a vertex subset S, we write $\mathrm{Vol}(S)$ to denote $\sum_{v\in S}\deg(v)$. Note that by default the degree is with respect to the original graph G. We write $\bar{S} = V\setminus S$, and let $\partial(S) = E(S, \bar{S})$ be the set of edges $e = \{u, v\}$ with $u \in S$ and $v \in \bar{S}$. The *sparsity* (or *conductance*) of a cut $(S, \bar{S})$ is defined as $\Phi(S) = |\partial(S)|/\min\{\mathrm{Vol}(S), \mathrm{Vol}(\bar{S})\}$. The *conductance* Φ_G of a graph G is the minimum value of $\Phi(S)$ over all vertex subsets S.

We have the following relation [23] between the mixing time $\tau_{\mathrm{mix}}(G)$ and conductance Φ_G:

$$\Theta\left(\frac{1}{\Phi_G}\right) \le \tau_{\mathrm{mix}}(G) \le \Theta\left(\frac{\log n}{\Phi_G^2}\right).$$

In particular, if the inverse of the conductance is $n^{o(1)}$, then the mixing time is also $n^{o(1)}$.

Our Graph Partition. We introduce a new, efficiently computable graph decomposition that partitions the edge set into *three* parts.

DEFINITION 1.2. *An n^δ-*decomposition *of a graph $G = (V, E)$ is a tripartition of the edge set $E = E_m \cup E_s \cup E_r$ satisfying the following conditions.*

(a) *Each connected component induced by E_m has $O(\mathrm{poly}\log n)$ mixing time, and each vertex in the component has $\Omega(n^\delta)$ incident edges in E_m. That is, for each vertex $v \in V$, either $\deg_{E_m}(v) = 0$ or $\deg_{E_m}(v) = \Omega(n^\delta)$.*

(b) *$E_s = \bigcup_{v\in V} E_{s,v}$, where $E_{s,v}$ is a subset of edges incident to v and $|E_{s,v}| \le n^\delta$. We view $E_{s,v}$ as* oriented *away from v. The overall orientation on E_s is* acyclic*, which certifies that E_s has arboricity*[2] *at most n^δ. Each vertex v knows $E_{s,v}$.*

(c) *$|E_r| \le |E|/6$.*

Throughout the paper we assume $\delta \in (0, 1)$ is a constant. The main difference between our graph partition and the ones in other works [41] is that we allow a set E_s that induces a low arboricity subgraph. The purpose of having the set E_s is to allow us to design an efficient CONGEST algorithm to construct the partition. In the sequential computation model, a common approach to find a graph partition is to iteratively find a vertex set S with small $\Phi(S) = O(1/\mathrm{polylog}(n))$, and then include the boundary edges $\partial(S)$ in the set E_r and remove them from the current graph. The number of iterations can be as high as $\tilde{\Theta}(n)$ since we could have $|S| = \tilde{O}(1)$.

To reduce the number of iterations to at most $O(n^{1-\delta})$, before we start to find S, we do a preprocessing step that removes low degree vertices in such a way that each vertex has degree at least $\Omega(n^\delta)$ in the remaining graph. This guarantees that $|S| = \Omega(n^\delta)$, and so the number of iterations can be upper bounded by $O(n^{1-\delta})$, since the total number of vertices is n.

1.2 Additional Related Works Drucker et al. [10] showed an $\Omega\left(\frac{n}{e^{\sqrt{\log n}}\log n}\right)$ lower bound for triangle detection in the *broadcast* CONGESTED-CLIQUE model, where each vertex can only broadcast one message to all other vertices in each round. In the CONGEST model, lower bounds for finding a triangles and other motifs (subgraphs) has been studied in [1, 10, 20, 26]. The problem of detecting a k-cycle has an $\tilde{\Omega}(\sqrt{n})$ lower bound, for any even number $k \ge 4$ [10,26]. Detecting a k-clique requires $\tilde{\Omega}(\sqrt{n})$ rounds for every $4 \le k \le \sqrt{n}$, and $\tilde{\Omega}(\sqrt{n}/k)$ rounds for every $k \ge \sqrt{n}$ [7].

Any <u>one-round</u> deterministic algorithm for the *triangle membership problem* (each vertex decides whether it belongs to a triangle) requires messages of size $\Omega(\Delta\log n)$ [1], which meets the trivial upper bound; for the randomized model, there is an $\Omega(\Delta)$ lower bound [14]. The distributed triangle detection problem has also been studied in the property testing setting in the CONGEST model [13].

Das Sarma et al. [40] first studied the distributed sparsest cut problem. Specifically, given two parameters b and ϕ, if there exists a cut of balance at least b and conductance at most ϕ, their algorithm outputs a cut of conductance at most $\tilde{O}(\sqrt{\phi})$ in $\tilde{O}((n+(1/\phi))/b)$ rounds. This result was later improved by Kuhn and Molla [28] to $\tilde{O}(D + 1/(b\phi))$.[3] Their algorithms are built upon techniques in [8].

[2] The arboricity of a graph is the minimum number α such that its edge set can be partitioned into α forests.

[3] Kuhn and Molla [28] further claimed that the output cut of

The local graph clustering algorithm of Spielman and Teng [41] has been improved, both in terms of running time and the quality of the cuts discovered; see, e.g., [3, 4, 30, 42].

1.3 Organization In Section 2 we present a new distributed algorithm for partitioning a graph into expanding subgraphs and a low-arboricity subgraph. A key subroutine for finding a sparse cut is described in Section 3. Section 4 presents Triangle Enumeration algorithms for both expanding graphs and general graphs. We conclude in Section 5 with a conjecture on the complexity of distributed graph partitioning.

2 Algorithm for Graph Partitioning

We first introduce some notation. Let $\deg_H(v)$ be the degree of v in the subgraph H, or in the graph induced by edge/vertex set H. Let $V(E^*)$ be the set of vertices induced by the edge set $E^* \subseteq E$. The *strong diameter* of a subgraph H of G is defined as $\max_{u,v\in H} \text{dist}_H(u,v)$ and the *weak diameter* of H is $\max_{u,v\in H} \text{dist}_G(u,v)$.

The goal of this section is to prove the following theorem.

THEOREM 2.1. *Given a graph $G = (V, E)$ with $n = |V|$, we can find, w.h.p., an n^δ-decomposition in $\tilde{O}(n^{1-\delta})$ rounds in the* CONGEST *model.*

The algorithm for Theorem 2.1 is based on repeated application of a black box algorithm $\mathcal{A}^*$, which is given a subgraph $G' = (V', E')$ of the original graph $G = (V, E)$, where $V' = V(E')$, $n' = |V'|$, and $m' = |E'|$. In $\mathcal{A}^*$, vertices may halt the algorithm at different times.

Specification of the Black Box. The goal of $\mathcal{A}^*$ is, given $G' = (V', E')$, to partition E' into $E' = E'_m \cup E'_s \cup E'_r$ satisfying some conditions. The edge set E'_m is partitioned into $E'_m = \bigcup_{i=1}^t \mathcal{E}_i$ for some t. We write $\mathcal{V}_i = V(\mathcal{E}_i)$ and $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$, and define $S = V' \setminus \left(\bigcup_{i=1}^t \mathcal{V}_i\right)$.

(C1) The vertex sets $\mathcal{V}_1, \ldots, \mathcal{V}_t, S$ are disjoint and partition V'.

(C2) The edge set E'_s can be decomposed as $E'_s = \bigcup_{v\in S} E'_{s,v}$, where $E'_{s,v}$ is a subset of edges incident to v, viewed as oriented away from v. This orientation is acyclic. For each vertex v such that $E'_{s,v} \neq \emptyset$, we have $|E'_{s,v}| + \deg_{E'_m}(v) \leq n^\delta$. Each vertex v knows the set $E'_{s,v}$.

(C3) Consider a subgraph $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$. Vertices in $\mathcal{V}_i$ halt after the same number of rounds, say K. Exactly one of the following subcases will be satisfied.

(C3-1) All vertices in $\mathcal{V}_i$ have degree $\Omega(n^\delta)$ in the subgraph $\mathcal{G}_i$, each connected component of $\mathcal{G}_i$ has $O(\text{poly}\log n)$ mixing time, and $K = O(\text{poly}\log n)$. Furthermore, every vertex in $\mathcal{V}_i$ knows that they are in this sub-case.

(C3-2) $|\mathcal{V}_i| \leq n' - \tilde{\Omega}(Kn^\delta)$, and every vertex in $\mathcal{V}_i$ knows they are in this subcase.

(C4) Each vertex $v \in S$ halts in $\tilde{O}(n'/n^\delta)$ rounds.

(C5) The following inequality is met:

$$E'_r \leq \Big(|E'|\log|E'| - \sum_{i=1}^t |\mathcal{E}_i|\log|\mathcal{E}_i|\Big)/(6\log m).$$

(C6) Each cluster $\mathcal{V}_i$ has a distinct identifier. When a vertex $v \in \mathcal{V}_i$ terminates, v knows the identifier of $\mathcal{V}_i$. If $v \in S$, v knows that it belongs to S.

We briefly explain the intuition behind these conditions. The algorithm $\mathcal{A}^*$ will be applied recursively to all subgraphs $\mathcal{G}_i$ that have yet to satisfy the minimum degree and mixing time requirements specified in Theorem 2.1 and Definition 1.2. Because vertices in different components halt at various times, they also may begin these recursive calls at different times.

The goal of (C2) is to make sure that once a vertex v has $E'_{s,v} \neq \emptyset$, the total number of edges added to $E_{s,v}$ cannot exceed n^δ. The goal of (C3) is to guarantee that the component size drops at a fast rate. The idea of (C5) is that the size of E'_r can be mostly charged to the number of the edges in the small-sized edge sets $\mathcal{E}_i$; this is used to bound the size of E_r in the final output of our graph partitioning algorithm.

Note that in general the strong diameter of a subgraph $\mathcal{G}_i$ can be much higher than the maximum running time of vertices in $\mathcal{G}_i$, and it could be possible that $\mathcal{G}_i$ is not even a connected subgraph of G. However, (C6) guarantees that each vertex $v \in \mathcal{V}_i$ still knows that it belongs to $\mathcal{V}_i$. This property allows us to recursively execute $\mathcal{A}^*$ on each subgraph $\mathcal{G}_i$.

LEMMA 2.1. *There is an algorithm $\mathcal{A}^*$ that finds a partition $E' = E'_m \cup E'_s \cup E'_r$ meeting the above specification in the* CONGEST *model, w.h.p.*

Assuming Lemma 2.1, we are now in a position to prove Theorem 2.1.

their algorithm has balance at least $b/2$, but this claim turns out to be incorrect (Anisur Rahaman Molla, personal communication, 2018).

Proof. [Proof of Theorem 2.1] Let $\mathcal{A}^*$ be the algorithm for Lemma 2.1. Initially, we apply $\mathcal{A}^*$ with $G' = G$, and this returns a partition $E' = E'_m \cup E'_s \cup E'_r$.

For each subgraph $\mathcal{G}_i$ in the partition output by an invocation of $\mathcal{A}^*$, do the following. If $\mathcal{G}_i$ satisfies (C3-1), by definition it must have $O(\text{poly}\log n)$ mixing time, and all vertices in $\mathcal{G}_i$ have degree $\Omega(n^\delta)$ in $\mathcal{G}_i$; we add the edge set $\mathcal{E}_i$ to the set E_m and all vertices in $\mathcal{V}_i$ halt. Otherwise we apply the algorithm recursively to $\mathcal{G}_i$, i.e., we begin by applying $\mathcal{A}^*$ to $G' = \mathcal{G}_i$ to further partition its edges. All recursive calls proceed in parallel, but may begin and end at different times. Conditions (C1) and (C6) guarantee that this is possible. (Note that if $\mathcal{G}_i$ is disconnected, then each connected component of $\mathcal{G}_i$ will execute the algorithm in isolation.)

Initially $E_r = \emptyset$ and $E_s = \emptyset$. After each invocation of $\mathcal{A}^*$, we update $E_r \leftarrow E_r \cup E'_r$, $E_s \leftarrow E_s \cup E'_s$, and $E_{s,u} \leftarrow E_{s,u} \cup E'_{s,u}$ for each vertex u.

Analysis. We verify that the three conditions of Definition 1.2 are satisfied. First of all, note that each connected component of E_m terminated in (C3-1) must have $O(\text{poly}\log n)$ mixing time, and all vertices in the component have degree $\Omega(n^\delta)$ within the component. Condition (a) of Definition 1.2 is met. Next, observe that Condition (b) of Definition 1.2 is met due to (C2). If the output of $\mathcal{A}^*$ satisfies that $E'_{s,v} \neq \emptyset$, then $|E_{s,v}|$ together with the number of remaining incident edges (i.e., the ones in E'_m) is less then n^δ. Therefore, $|E_{s,v}|$ cannot exceed n^δ, since only the edges in E'_m that are incident to v can be added to $E_{s,v}$ in future recursive calls. Lastly, we argue that (C5) implies that Condition (c) of Definition 1.2 is satisfied. Assume, inductively, that a recursive call on edge set $\mathcal{E}_i$ eventually contributes at most $|\mathcal{E}_i| \log |\mathcal{E}_i|/(6 \log m)$ edges to E_r. It follows from (C5) that the recursive call on edge set E' contributes $|E'| \log |E'|/(6 \log m)$ edges to E_r. We conclude that $|E_r| \leq |E| \log |E|/(6 \log |E|) = |E|/6$.

Now we analyze the round complexity. In one recursive call of $\mathcal{A}^*$, consider a component $\mathcal{G}_i$ in the output partition, and let K be the running time of vertices in $\mathcal{V}_i$. Due to (C3), there are two cases. If $\mathcal{G}_i$ satisfied (C3-1), it will halt in $K = O(\text{poly}\log n)$ rounds. Otherwise, (C3-2) is met, and we have $|\mathcal{V}_i| \leq n' - \tilde{\Omega}(Kn^\delta)$. Let $v \in V$ be any vertex, and let $K_1, \ldots, K_z$ be the running times of all calls to $\mathcal{A}^*$ that involve v. (Whenever v ends up in S or in a component satisfying (C3-1) it halts permanently, so $K_1, \ldots, K_{z-1}$ reflect executions that satisfy (C3-2) upon termination.) Then we must have $\sum_{i=1}^{z} K_i \leq \tilde{O}(n/n^\delta) + O(\text{poly}\log n) = \tilde{O}(n^{1-\delta})$. Thus, the whole algorithm stops within $\tilde{O}(n^{1-\delta})$ rounds. □

2.1 Subroutines Before proving Lemma 2.1, we first introduce some helpful subroutines. Lemma 2.3 shows that for subgraphs of sufficiently high strong diameter, we can find a sparse cut of the subgraph, with runtime proportional to the strong diameter. Lemma 2.4 offers a procedure that removes a set of edges in such a way that the vertices in the remaining graph have high degree, and the removed edges form a low arboricity subgraph. Lemma 2.5 shows that if a subgraph already has a low conductance cut, then we can efficiently find a cut of similar quality.

All these subroutines are applied to a connected subgraph $G^* = (V^*, E^*)$ of the underlying network $G = (V, E)$, and the computation does not involve vertices outside of G^*. In subsequent discussion in this section, the parameters n and m are always defined as $n = |V|$ and $m = |E|$, which are independent of the chosen subgraph G^*.

LEMMA 2.2. *Let m and D be two numbers. Let $(a_1, \ldots, a_D)$ be a sequence of positive integers such that $D \geq 48 \log^2 m$ and $\sum_{i=1}^{D} a_i \leq m$. Then there exists an index j such that $j \in [D/4, 3D/4]$ and*

$$a_j \leq \frac{1}{12 \log m} \cdot \min\left(\sum_{i=1}^{j-1} a_i,\ \sum_{i=j+1}^{D} a_i \right).$$

Proof. Define $S_k = \sum_{i=1}^{k} a_i$ to be the kth prefix sum. By symmetry, we may assume $S_{\lfloor D/2 \rfloor} \leq S_D - S_{\lfloor D/2 \rfloor}$, since otherwise we can reverse the sequence. Scan each index j from $D/4$ to $D/2$. If an index j does not satisfy $a_j \leq \frac{1}{12 \log m} \cdot S_{j-1}$, then this implies that $S_j > S_{j-1}\left(1 + \frac{1}{12 \log m}\right)$. If no index $j \in [D/4, D/2]$ satisfies this condition then $S_{\lfloor D/2 \rfloor}$ is larger than

$$\begin{aligned} & S_{\lfloor D/4 \rfloor} \cdot \left(1 + \frac{1}{12 \log m}\right)^{D/4} \\ & \geq S_{\lfloor D/4 \rfloor} \cdot \left(1 + \frac{1}{12 \log m}\right)^{12 \log^2 m} \\ & \geq S_{\lfloor D/4 \rfloor} \cdot m, \end{aligned}$$

which is impossible since $\sum_{i=1}^{D} a_i \leq m$. Therefore, there must exist an index $j \in [D/4, D/2]$ such that $a_j \leq \frac{1}{12 \log m} \cdot S_{j-1} = \frac{1}{12 \log m} \cdot \sum_{i=1}^{j-1} a_i$. By our assumption that $S_{\lfloor D/2 \rfloor} \leq S_D - S_{\lfloor D/2 \rfloor}$, we also have $a_j \leq \frac{1}{12 \log m} \cdot \min\left(\sum_{i=1}^{j-1} a_i,\ \sum_{i=j+1}^{D} a_i\right)$. □

In Lemma 2.3, the requirement that there are no edges linking two vertices in V_{low} implies that the strong diameter of $G^* = (V^*, E^*)$ is $O(n^{1-\delta})$, and so the runtime of Lemma 2.3 is always at most $O(n^{1-\delta})$.

LEMMA 2.3. (HIGH DIAMETER SUBROUTINE) *Let $G^* = (V^*, E^*)$ be a connected subgraph and $x \in V^*$ be a vertex for which $\tilde{D} = \max_{v \in V^*} \mathrm{dist}_{G^*}(x, v) \geq 48 \log^2 m$. Define $V_{\mathrm{low}} = \{v \in V^* \mid \deg_{G^*}(v) \leq n^\delta/2\}$. Suppose there are no edges linking two vertices in V_{low}. Then we can find a cut $(C, \bar{C})$ of G^* such that $\min(|C|, |\bar{C}|) \geq \frac{\tilde{D}}{32} n^\delta$ and $\partial(C) \leq \min(\mathrm{Vol}(C), \mathrm{Vol}(\bar{C}))/(12 \log m)$ in $O(\tilde{D})$ rounds deterministically in the* CONGEST *model. Each vertex in V^* knows whether or not it is in C.*

Proof. The algorithm is as follows. First, build a BFS tree of G^* rooted at $x \in V^*$ in $O(\tilde{D})$ rounds. Let L_i be the set of vertices of level i in the BFS tree, and let p_i be the number of edges $e = \{u, v\}$ such that $u \in L_i$ and $v \in L_{i+1}$. We write $L_{a..b} = \bigcup_{i=a}^{b} L_i$. In $O(\tilde{D})$ rounds we can let the root x learn the sequence $(p_1, \ldots, p_{\tilde{D}})$.

Note that in a BFS tree, edges do not connect two vertices in non-adjacent levels. By Lemma 2.2, there exists an index $j \in [\tilde{D}/4, 3\tilde{D}/4]$ such that

$$\begin{aligned} p_j &\leq \frac{1}{12 \log m} \cdot \min \left(\sum_{i=1}^{j-1} p_i, \sum_{i=j+1}^{\tilde{D}} p_i \right) \\ &\leq \frac{1}{12 \log m} \cdot \min \left(\mathrm{Vol}(L_{1..j}), \mathrm{Vol}(L_{j+1..\tilde{D}}) \right), \end{aligned}$$

and such an index j can be computed locally at the vertex x.

The cut is chosen to be $C = L_{1..j}$, so we have $\partial(C) \leq \min(\mathrm{Vol}(C), \mathrm{Vol}(\bar{C}))/(12 \log m)$. As for the second condition, due to our assumption in the statement of the lemma, for any two adjacent levels L_i, L_{i+1}, there must exist a vertex $v \in L_i \cup L_{i+1}$ such that $v \notin V_{\mathrm{low}}$. By definition of V_{low}, v has more than $n^\delta/2$ neighbors in G^*, and they are all within $L_{i-1..i+2}$. Thus, the number of vertices within any four consecutive levels must be greater than $n^\delta/2$. Since $j \in [\tilde{D}/4, 3\tilde{D}/4]$, we have

$$\min(|C|, |\bar{C}|) \geq \frac{\tilde{D}}{4}/4 \cdot n^\delta/2 \geq \frac{\tilde{D}}{32} n^\delta.$$

To let each vertex in V^* learn whether or not it is in C, the root x broadcasts the index j to all vertices in G^*. After that, each vertex in level smaller than or equal to j knows that it is in C; otherwise it is in $\bar{C}$. □

Intuitively, Lemma 2.4 says that after the removal of a subgraph of small arboricity (i.e., the edge set $E_s^\diamond$), the remaining graph (i.e., the edge set $E^\diamond$) has high minimum degree. The runtime is proportional to the number of removed vertices (i.e., $|V^*| - |V^\diamond|$) divided by the threshold n^δ. Note that the second condition of Lemma 2.4 implies that $E_{s,v}^\diamond = \emptyset$ for all $v \in V^\diamond$.

LEMMA 2.4. (LOW DEGREE SUBROUTINE) *Let $G^* = (V^*, E^*)$ be a connected subgraph with strong diameter D. We can partition $E^* = E^\diamond \cup E_s^\diamond$ meeting the following two conditions.*

1. *Let $V^\diamond$ be the set of vertices induced by $E^\diamond$. Each $v \in V^\diamond$ has more than $n^\delta/2$ incident edges in $E^\diamond$.*
2. *The edge set $E_s^\diamond$ is further partitioned as $E_s^\diamond = \bigcup_{v \in V^* \setminus V^\diamond} E_{s,v}^\diamond$, where $E_{s,v}^\diamond$ is a subset of incident edges of v, and $|E_{s,v}^\diamond| \leq n^\delta$. Each vertex v knows $E_{s,v}^\diamond$.*

This partition can be found in $O(D + (|V^*| - |V^\diamond|)/n^\delta)$ rounds deterministically in the CONGEST *model.*

Proof. To meet Condition 1, a naive approach is to iteratively "peel off" vertices that have degree at most $n^\delta/2$, i.e., put all their incident edges in E_s, so long as any such vertex exists. On some graphs this process requires $\Omega(n)$ peeling iterations.

We solve this issue by doing a batch deletion. First, build a BFS tree of G^* rooted at an arbitrary vertex $x \in V^*$. We use this BFS tree to let x count the number of vertices that have degree less than n^δ in the remaining subgraph in $O(D)$ rounds.

The algorithm proceeds in iterations. Initially we set $E^\diamond \leftarrow E^*$ and $E_s^\diamond \leftarrow \emptyset$. In each iteration, we identify the subset $Z \subseteq V^*$ whose vertices have at most n^δ incident edges in $E^\diamond$. We orient all the $E^\diamond$-edges touching Z away from Z, if one endpoint is in Z, or away from the endpoint with smaller ID, if both endpoints are in Z. Edges incident to v oriented away from v are added to $E_{s,v}^\diamond$ and removed from $E^\diamond$. The root x then counts the number $z = |Z|$ of such vertices via the BFS tree. If $z > n^\delta/2$, we proceed to the next iteration; otherwise we terminate the algorithm.

The termination condition ensures that each vertex has degree at least $(n^\delta + 1) - z > n^\delta/2$, and so Condition 1 is met. It is straightforward to see that the set $E_s^\diamond$ generated by the algorithm meets Condition 2, since for each v, we only add edges to $E_{s,v}^\diamond$ once, and it is guaranteed that $|E_{s,v}^\diamond| \leq n^\delta$. Tie-breaking according to vertex-ID ensures the orientation is acyclic.

Throughout the process, each time one vertex puts any edges into $E_s^\diamond$, it no longer stays in $V^\diamond$. Each iteration can be done in $O(D)$ time. We proceed to the next iteration only if there are more than $n^\delta/2$ vertices being removed from $V^\diamond$. A trivial implementation can lead to an algorithm taking $O(D \lceil (|V^*| - |V^\diamond|)/n^\delta \rceil)$ rounds. The round complexity can be further improved to $O(D + (|V^*| - |V^\diamond|)/n^\delta)$ by pipelining the iterations. At some point the root x detects that iteration i was the last iteration; in $O(D)$ time it broadcasts a message

to all nodes instructing them to roll back iterations $i+1, i+2, \ldots$, which have been executed speculatively. □

The proof of the following lemma is deferred to Section 3. It is a consequence of combining Lemmas 3.4 and 3.5.

LEMMA 2.5. (LOW CONDUCTANCE SUBROUTINE) *Let $G^* = (V^*, E^*)$ be a connected subgraph with strong diameter D. Let $\phi \leq 1/12$ be a number. Suppose that there exists a subset $S \subset V^*$ satisfying*

$$\mathrm{Vol}(S) \leq (2/3)\mathrm{Vol}(V^*) \text{ and } \Phi(S) \leq \frac{\phi^3}{19208 \ln^2(|E^*|e^4)}.$$

Assuming such an S exists, there is a CONGEST *algorithm that finds a cut $C \subset V^*$ such that $\Phi(C) \leq 12\phi$ in $O(D + \mathrm{poly}(\log|E^*|, 1/\phi))$ rounds, with failure probability $1/\mathrm{poly}(|E^*|)$. Each vertex in V^* knows whether or not it belongs to C.*

2.2 Proof of Lemma 2.1 We prove Lemma 2.1 by presenting and analyzing a specific distributed algorithm, which makes use of the subroutines specified in Lemmas 2.3, 2.4, and 2.5.

Recall that we are given a subgraph with edge set E' and must ultimately return a partition of it into $E'_m \cup E'_s \cup E'_r$. The algorithm initializes $E'_m \leftarrow E'$, $E'_s \leftarrow \emptyset$, and $E'_r \leftarrow \emptyset$. There are two types of special operations.

Remove. In an Remove operation, some edges are moved from E'_m to either E'_s or E'_r. For the sake of a clearer presentation, each such operation is tagged Remove-i, for some index i.

Split. Throughout the algorithm we maintain a partition of the current set E'_m. In a Split operation, the partition subdivided. Each such operation is tagged as Split-i, for some index i, such that Split-i occurs right after Remove-i.

Throughout the algorithm, we ensure that any part $E^\star$ of the partition of E'_m has an identifier that is known to all members of $V(E^\star)$. It is not required that each part forms a connected subgraph. The partition at the end of the algorithm, $E'_m = \bigcup_{i=1}^{t} \mathcal{E}_i$, is the output partition.

Notations. Since we treat E'_m as the "active" edge set and E'_s and E'_r as repositories of removed edges, $\deg(v)$ refers to the degree of v in the subgraph induced by the *current* E'_m. We write $V_{\mathrm{low}} = \{v \in V' \mid \deg(v) \leq n^\delta\}$.

Algorithm. In the first step of the algorithm, we move each edge $\{u, v\} \in E'_m$ in the subgraph induced by V_{low} to the set $E'_{s,u}$ (Remove-1), assuming $\mathrm{ID}(u) < \mathrm{ID}(v)$. Note that breaking ties by vertex-ID is critical to keep the orientation acyclic. This step only removes all edges in the subgraph induced by V_{low}; edges between V_{low} and $V' \setminus V_{\mathrm{low}}$ are left as is, so the identity of V_{low} is unchanged after this step.

After that, E'_m is divided into connected components. Assume these components are $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2), \ldots$, where $V_i = V(E_i)$. Let D_i be the depth of a BFS tree rooted at an arbitrary vertex in G_i. In $O(D_i)$ rounds, the subgraph G_i is assigned an identifier that is known to all vertices in V_i (Split-1). Note that this step is done in parallel for each G_i, and the time for this step is different for each G_i. From now on there will be no communication between different subgraphs in $\{G_1, G_2, \ldots\}$, and we focus on one specific subgraph G_i in the description of the algorithm.

Depending on how large D_i is, there are two cases. If $D_i \geq 48 \log^2 m$, we go to Case 1, otherwise we go to Case 2.

Case 1: In this case, we have $D_i \geq 48 \log^2 m$. Since there are no edges connecting two vertices in V_{low}, we can apply the High Diameter subroutine, Lemma 2.3, which finds a cut $(C, \bar{C})$ of G_i such that $\min(|C|, |V_i \setminus C|) \geq \frac{D_i}{32} n^\delta$ and $\partial(C) \leq \min(\mathrm{Vol}(C), \mathrm{Vol}(V_i \setminus C))/(12 \log m)$ in $O(D_i)$ rounds. Every vertex in V_i knows whether it is in C or not. All edges of the cut $(C, \bar{C})$ are put into E'_r (Remove-2). Then E_i splits into two parts according to the cut $(C, \bar{C})$ (Split-2). After that, all vertices in V_i terminate. (Observe that the part containing the BFS tree root is connected, but the other part is not necessarily connected.)

Case 2: In this case, we have $D_i \leq 48 \log^2 m$. Since $G_i = (V_i, E_i)$ is a small diameter graph, a vertex $v \in V_i$ is able broadcast a message to all vertices in V_i very fast. We apply the Low Degree subroutine, Lemma 2.4, to obtain a partition $E_i = E^\diamond \cup E^\diamond_s$. We add all edges in $E^\diamond_s$ to E'_s in such a way that $E'_{s,v} \leftarrow E'_{s,v} \cup E^\diamond_{s,v}$ for all $v \in V_i \setminus V^\diamond$, where $V^\diamond = V(E^\diamond)$ (Remove-3).

After removing these edges, the remaining edges of E_i are divided into several connected components, but all remaining vertices have degree larger than $n^\delta/2$. Assume these connected components are $G_{i,1} = (V_{i,1}, E_{i,1})$, $G_{i,2} = (V_{i,2}, E_{i,2}), \ldots$. Let $D_{i,j}$ be the depth of the BFS tree from an arbitrary root vertex in $G_{i,j}$. In $O(D_{i,j})$ rounds we compute such a BFS tree and assign an identifier that is known to all vertices in $V_{i,j}$ (Split-3). That is, the remaining edges in E_i are partitioned into $E_{i,1}$, $E_{1,2}$, ….

In what follows, we focus on one subgraph $G_{i,j}$ and proceed to Case 2-a or Case 2-b.

Case 2-a: In this case, $D_{i,j} \geq 48\log^2 m$. The input specification of the High Diameter subroutine (Lemma 2.3) is satisfied, since every vertex has degree larger than $n^\delta/2$. We apply the High Diameter subroutine to $G_{i,j}$. This takes $O(D_{i,j})$ rounds. This case is similar to Case 1, and we do the same thing as what we do in Case 1, i.e., remove the edges in the cut found by the subroutine (Remove-4), split the remaining edges (Split-4), and then all vertices in $V_{i,j}$ terminate.

Case 2-b: In this case, $D_{i,j} \leq 48\log^2 m$. Note that every vertex has degree larger than $n^\delta/2$, and $G_{i,j}$ has small diameter. What we do in this case is to test whether $G_{i,j}$ has any low conductance cut; if yes, we will split $E_{i,j}$ into two components. To do so, we apply the Low Conductance subroutine, Lemma 2.5, with $\phi = \frac{1}{144\log m}$. Based on the result, there are two cases.

Case 2-b-i: The subroutine finds a set of vertices C that $\Phi(C) \leq 12\phi = \frac{1}{12\log m}$, and every vertex knows whether it is in C or not. We move $\partial(C)$ to E'_r (Remove-5), and then split the remaining edges into two edge sets according to the cut $(C, \bar{C})$ (Split-5). After that, all vertices in $V_{i,j}$ terminate.

Case 2-b-ii: Otherwise, the subroutine does not return a subset C, and it means with probability at least $1 - 1/\text{poly}(|E_{i,j}|) = 1 - 1/\text{poly}(n)$, there is no cut $(S, \bar{S})$ with conductance less than $\frac{\phi^3}{19208\ln^2(|E_{i,j}|e^4)} = \Theta(\log^{-5} m)$. Recall the relation between the mixing time $\tau_{\text{mix}}(G_{i,j})$ and the conductance $\Phi = \Phi_{G_{i,j}}$: $\Theta(\frac{1}{\Phi}) \leq \tau_{\text{mix}}(G_{i,j}) \leq \Theta(\frac{\log|V_{i,j}|}{\Phi^2})$ [23]. Therefore, w.h.p., $G_{i,j}$ has $O(\text{poly}\log n)$ mixing time. All vertices in $V_{i,j}$ terminate without doing anything in this step.

Note that in the above calculation, we use the fact that every vertex in $V_{i,j}$ has degree larger than $n^\delta/2$ in $G_{i,j}$, and this implies that $|V_{i,j}| = \Omega(n^\delta)$ and $|E_{i,j}| = \Omega(n^{2\delta})$, and so $\Theta(\log m) = \Theta(\log n) = \Theta(\log|E_{i,j}|) = \Theta(\log|V_{i,j}|)$.

Analysis. We show that the output of $\mathcal{A}^*$ meets its specifications (C1)–(C6). Recall that $E'_m = \bigcup_{i=1}^t \mathcal{E}_i$ is the final partition of the edge set E'_m when all vertices terminate. Once an edge is moved from E'_m to either E'_r or E'_s, it remains there for the rest of the computation. Condition (C1) follows from the fact that each time we do a split operation, the induced vertex set of each part is disjoint. Condition (C6) follows from the fact that each vertex knows which part of E'_m it belongs to after each split operation. In the rest of this section, we prove that the remaining conditions are met.

CLAIM 2.1. *Condition (C2) is met.*

Proof. Note that only Remove-1 and Remove-3 involve E'_s. In Remove-1, any $E'_{s,u}$ that becomes non-empty must have had $u \in V_{\text{low}}$, so $\deg(u) \leq n^\delta$ before Remove-1, and therefore $|E'_{s,u}| + \deg(u) \leq n^\delta$ after Remove-1. In Remove-3, the Low Degree subroutine of Lemma 2.4 computes a partition $E_i = E^\diamond \cup E^\diamond_s$, and then we update $E'_{s,u} \leftarrow E'_{s,u} \cup E^\diamond_{s,u}$ for all $u \in V_i \setminus V^\diamond$. By Lemma 2.4, for any u such that $E^\diamond_{s,u} \neq \emptyset$, we have $|E^\diamond_{s,u}| \leq n^\delta$, and $u \notin V^\diamond$, where $V^\diamond$ is the vertex set induced by the remaining edge set $E^\diamond$. In other words, once u puts at least one edge into $E'_{s,u}$, we have $\deg(u) = 0$ after Remove-3. □

CLAIM 2.2. *Conditions (C3) and (C4) are met.*

Proof. We need to verify that in each part of the algorithm, we either spend only $O(\text{poly}\log n)$ rounds, or the size of the current component shrinks by $\tilde{\Omega}(n^\delta)$ vertices *per round*.

After removing all edges in the subgraph induced by V_{low}, the rest of E' is partitioned into connected components $\mathcal{E}_1, \mathcal{E}_2, \ldots$. Consider one such component $\mathcal{E}_i$, and suppose it goes to Case 1. We find a sparse cut $(C, \bar{C})$, and moving $\partial(C)$ to E'_r breaks $\mathcal{E}_i$ into $\mathcal{E}^1_i$ and $\mathcal{E}^2_i$. By Lemma 2.3, we have $\min(|C|, |\bar{C}|) \geq \frac{D_i}{32}n^\delta$, so the size of both $V(\mathcal{E}^1_i) = C$ and $V(\mathcal{E}^2_i) = \bar{C}$ are at most $|V(\mathcal{E}_i)| - \frac{D_i}{32}n^\delta \leq n' - \Omega(D_i)n^\delta$. Since the running time for each vertex in $V(\mathcal{E}^1_i)$ and $V(\mathcal{E}^2_i)$ is $O(D_i)$, the condition (C3-2) is met.

Now suppose that $\mathcal{E}_i$ goes to Case 2. Note that the total time spent before it reaches Case 2 is $O(D_i) = \text{poly}\log n$. In Case 2 we execute the Low Degree subroutine of Lemma 2.4, and let the time spent in this subroutine be τ. By Lemma 2.4, it is either the case that (i) $\tau = O(D_i)$ or (ii) the remaining vertex set $V^\diamond$ satisfies $|V(E_i)| - |V^\diamond| = \Omega(\tau n^\delta)$. In other words, if we spend too much time (i.e., $\omega(D_i)$) on this subroutine, we must lose $\Omega(n^\delta)$ vertices per round.

After that, $\mathcal{E}_i$ is split into $\mathcal{E}_{i,1}, \mathcal{E}_{i,2}, \ldots$. We consider the set $\mathcal{E}_{i,j}$. If $\mathcal{E}_{i,j}$ goes to Case 2-a, then the analysis is the same as that in Case 1, and so (C3-2) is met.

Now suppose that $\mathcal{E}_{i,j}$ goes to Case 2-b. Note that the time spent during the Low Conductance subroutine of Lemma 2.5 is $O(\text{poly}\log n)$. Suppose that a low conductance cut $(C, \bar{C})$ is found (Case 2-b-i). Since the cut has conductance less than $\frac{1}{12\log m}$, by the fact that every vertex has degree higher than $n^\delta/2$, we must have $\min(|C|, |\bar{C}|) = \Omega(n^\delta)$. Assume $\mathcal{E}_{i,j} \setminus \partial(C)$ is split into $\mathcal{E}^1_{i,j}$ and $\mathcal{E}^2_{i,j}$. The size of both $V(\mathcal{E}^1_{i,j})$ and $V(\mathcal{E}^2_{i,j})$ must be at most $|V(\mathcal{E}_{i,j})| - \Omega(n^\delta)$. Thus, (C3-2) holds for both parts $\mathcal{E}^1_{i,j}$ and $\mathcal{E}^2_{i,j}$.

Suppose that no cut $(C, \bar{C})$ is found (Case 2-b-ii). If the running time K among vertices in $V_{i,j}$ is $O(\text{poly}\log n)$, then (C3-1) holds. Otherwise, we must have $|V_{i,j}| \leq n' - \tilde{\Omega}(Kn^\delta)$ due to the Low Degree subroutine, and so (C3-2) holds.

Condition (C4) follows from the the above proof of (C3), since for each part of the algorithm, it is either the case that (i) this part takes $O(\text{poly}\log n)$ time, or (ii) the number of vertices in the current subgraph is reduced by $\tilde{\Omega}(n^\delta)$ *per round*. □

CLAIM 2.3. *Condition (C5) is met.*

Proof. Condition (C5) says that after the algorithm $\mathcal{A}^*$ completes, $|E'_r| \le f$, where

$$f = \left(|E'| \log |E'| - \sum_{i=1}^{t} |\mathcal{E}_i| \log |\mathcal{E}_i| \right) / (6 \log m).$$

We prove the stronger claim that this inequality holds at all times w.r.t. the current edge partition $\mathcal{E}_1 \cup \cdots \cup \mathcal{E}_t$ of E'_m. In the base case this is clearly true, since $t = 1$ and $E' = E'_m = \mathcal{E}_1$ and $E'_r = \emptyset$. Moving edges from E'_m to E'_s increases f and has no effect on E'_r, so we only have to consider the movement of edges from E'_m to E'_r. Note that this only occurs in Remove-i and Split-i, for $i \in \{2, 4, 5\}$, where in these operations we find a cut $(C, \bar{C})$ and split one of the parts $\mathcal{E}_j$ according to the cut. In all cases we have

$$|\partial(C)| \le \frac{\min(\text{Vol}(C), \text{Vol}(\bar{C}))}{12 \log m}.$$

Suppose that removing $\partial(C)$ splits $\mathcal{E}_j$ into $\mathcal{E}_j^1$ and $\mathcal{E}_j^2$, with $|\mathcal{E}_j^1| \le |\mathcal{E}_j^2|$ and $C = V(\mathcal{E}_j^1)$. We bound the change in $|E'_r|$ and f separately. Clearly

$$\Delta |E'_r| = |\partial(C)| \le \frac{2|\mathcal{E}_j^1| + \partial(C)}{12 \log m} \le \frac{|\mathcal{E}_j^1|}{6 \log m} + \frac{\partial(C)}{12 \log m}.$$

and

$$\begin{aligned} \Delta f &= \frac{1}{6 \log m} \cdot \left(|\mathcal{E}_j| \log |\mathcal{E}_j| - \sum_{k \in \{1,2\}} |\mathcal{E}_j^k| \log |\mathcal{E}_j^k| \right) \\ &\ge \frac{1}{6 \log m} \cdot \left(|\mathcal{E}_j^1| \log(|\mathcal{E}_j|/|\mathcal{E}_j^1|) + \partial(C) \log |\mathcal{E}_j| \right) \\ &> \Delta |E'_r|. \qquad \text{(Because } |\mathcal{E}_j^1| < |\mathcal{E}_j|/2\text{)} \end{aligned}$$

Thus, $|E'_r| \le f$ also holds after Remove-i and Split-i, for $i \in \{2, 4, 5\}$. □

3 Algorithm for Finding a Sparse Cut

Recall that in our decomposition routine, we search for a sparse cut in a *subgraph* $G^* = (V(E^*), E^*)$ of G. In this section, we do not care about anything outside of G^*, and so we slightly abuse the notation to write $G = (V, E)$ to denote the subgraph G^*, and we use $n = |V(G^*)|$ and $m = |E(G^*)|$ to be the number of vertices and edges in the subgraph. In this section we prove Lemma 2.5, which concerns an efficient distributed analogue of Spielman and Teng's [41, 42] `Nibble` routine.

Many existing works [3, 8, 28, 41] have shown that looking at the distribution of random walks is a good approach to finding a sparse cut. The basic idea is to first sample a source vertex s according to the degree distribution, i.e., the probability that v is sampled is $\deg(v)/(2m)$, and do a lazy random walk from s. Assume there is a sparse cut S with conductance $\Phi(S)$, and $\text{Vol}(S) \le \text{Vol}(V)/2$. If $s \in S$, then the probability distribution of the random walk will be mostly confined to S within the initial $t_0 = O(\frac{1}{\Phi(S)})$ steps. A common way to utilize this observation is to sort the vertices $(v_1, \ldots, v_n)$ in decreasing order of their random walk probability, and it is guaranteed that for some choice of j, the subset $C = \{v_1, \ldots, v_j\}$ is a sparse cut that is approximately as good as S.

The papers [28, 40] adapted this approach to the CONGEST model. If the cut S satisfies that $b \cdot 2|E| \le \text{Vol}(S)$ (*i.e., S has balance b*), then a cut C satisfying $\Phi(C) = O(\sqrt{\Phi(S) \log n})$ can be found in $\tilde{O}(D + 1/(b\Phi(S)))$ rounds. The algorithm is inefficient when $1/b = \Theta(|E|/\text{Vol}(S))$ is large. The main source of this inefficiency is that if we sample a vertex s according to the degree distribution, then the probability that $s \in S$ is only $O(b)$. This implies that we have to calculate many random walk distributions before we find a desired sparse cut. If we calculate these random walk distributions *simultaneously*, then *we may suffer from a huge congestion issue.*

Spielman and Teng [41] show that a *random walk* distribution with *truncation* (rounding a probability to zero when it becomes too small) can reveal a sparse cut, provided the starting vertex of the random walk is good. The main contribution of this section is a proof that the Spielman-Teng method for finding cuts of conductance roughly ϕ can be implemented in $\text{poly}(\phi^{-1}, \log n)$ time in the CONGEST model, i.e., with no dependence on the balance parameter b.

Terminology. We first review some definitions and results from Spielman and Teng [41]. Let A be the adjacency matrix of the graph G. We assume a 1-1 correspondence between $V(G)$ and $\{1, \ldots, n\}$. In a *lazy random walk*, the walk stays at the current vertex with probability 1/2 and otherwise moves to a random neighbor of the current vertex. The matrix realizing this walk can be expressed as $T = (AD^{-1} + I)/2$, where D is the diagonal matrix with $(d(1), \ldots, d(n))$ on the diagonal, and $d(i) = \deg(i)$.

Let p_t^v be the probability distribution of the lazy random walk that begins at v and walks for t steps. In

the limit, as $t \to \infty$, $p_t(x)$ approaches $d(x)/2m$, so it is natural to measure $p_t(x)$ *relative* to this baseline.

$$\rho_t(x) = p_t(x)/d(x),$$

Define π_t to be the permutation that sorts $V = \{1, \ldots, n\}$ in decreasing order of ρ_t-values, breaking ties by vertex ID. We never actually compute π_t. To implement our algorithms, it suffices that given $\rho_t(u), \rho_t(v), \mathrm{ID}(u), \mathrm{ID}(v)$, we can determine whether or not u precedes v according to π_t.

$$\rho_t(\pi_t(i)) \geq \rho_t(\pi_t(i+1)), \text{ for all } i.$$

Let p be a distribution on V. The truncation operation $[p]_\epsilon$ rounds $p(x)$ to zero if it falls below a threshold that depends on x.

$$[p]_\epsilon(x) = \begin{cases} p(x) & \text{if } p(x) \geq 2\epsilon d(x), \\ 0 & \text{otherwise.} \end{cases}$$

The truncated random walk starting at vertex v is defined as follows. In subsequent discussion we may omit v if it is known implicitly.

$$\tilde{p}_0^v(x) = \begin{cases} 1 & x = v \text{ and } 1 \geq 2\epsilon d(x), \\ 0 & \text{otherwise.} \end{cases}$$
$$\tilde{p}_t^v = [T\tilde{p}_{t-1}]_\epsilon.$$

The description of the algorithm `Nibble` and Lemma 3.1 in [41] implies the following lemma.[4]

LEMMA 3.1. ([41]) *For each $\phi \leq 1$, define the parameters*

$$t_0 = \frac{49\ln(me^4)}{\phi^2}$$
$$\textit{and} \quad \gamma = \frac{5\phi}{392\ln(me^4)}.$$

For each subset $S \subset V$ satisfying

$$\mathrm{Vol}(S) \leq \frac{2}{3} \cdot \mathrm{Vol}(V)$$
$$\textit{and} \quad \Phi(S) \leq \frac{\phi^3}{19208\ln^2(me^4)},$$

there exists a subset $S^g \subseteq S$ with the following properties. First, $\mathrm{Vol}(S^g) \geq \mathrm{Vol}(S)/2$. Second, S^g is partitioned into $S^g = \bigcup_{b=1}^{\log m} S_b^g$ such that if a random walk is initiated at any $v \in S_b^g$ with truncation parameter $\epsilon = \frac{\phi}{56\ln(me^4)t_0 2^b}$, then there exists a number $t \in [1, t_0]$ and an index j such that the following four conditions are met for the cut $C = \{\tilde{\pi}_t^v(1), \ldots, \tilde{\pi}_t^v(j))\}$.

(i) $\Phi(C) \leq \phi$,

(ii) $\tilde{\rho}_t(\tilde{\pi}_t(j)) \geq \gamma/\mathrm{Vol}(C)$,

(iii) $\mathrm{Vol}(C \cap S) \geq (4/7)2^{b-1}$,

(iv) $\mathrm{Vol}(C) \leq (5/6)\mathrm{Vol}(V)$.

In subsequent discussion, with respect to a given parameter $\phi \leq 1$, for any subset $S \subset V$ satisfying the condition of Lemma 3.1, we fix a subset $S^g \subseteq S$ and its decomposition $S^g = \bigcup_{b=1}^{\log m} S_b^g$ to be any choices satisfying Lemma 3.1.

3.1 Distributed Algorithm Now we give our algorithm `Distributed Nibble`. To simplify things, we present it as a sequential algorithm, and prove in Lemma 3.5 that it can be implemented efficiently in the CONGEST model. For any permutation π, we use the notation $\pi(i..j)$ to denote the set $\{\pi(i), \pi(i+1), \ldots, \pi(j)\}$.

Algorithm 1 `Distributed Nibble`

Input: ϕ.
for parameter $b = 1$ to $\lceil \log m \rceil$ **do**
  Set parameters $t_0 = 49\ln(me^4)/\phi^2$, and $\epsilon_b = \frac{\phi}{56\ln(me^4)t_0 2^b}$, as in Lemma 3.1.
  (1) Independently randomly sample $K = c\log m \cdot \frac{\mathrm{Vol}(V)}{2^b}$ vertices $v_1, ..., v_K$ proportional to their degrees, where c is a large enough constant.
  Initialize $\tilde{p}_0^{v_i}$.
  for $t = 1$ to t_0, for every v_i **do**
    (2) Calculate $\tilde{p}_t^{v_i} = [T\tilde{p}_{t-1}^{v_i}]_{\epsilon_b}$.
    Denote j_{max} as the largest index such that $\tilde{p}_t^{v_i}(\tilde{\pi}_t^{v_i}(j_{max})) > 0$.
    for $x = 0$ to $\log_{1+\phi}(5/6)\mathrm{Vol}(V)$ **do**
      (3) Set $j \leq j_{max}$ to be the largest index that $\mathrm{Vol}(\tilde{\pi}_t^{v_i}(1..j)) \leq (1+\phi)^x$.
      (4) If $\Phi(\tilde{\pi}_t^{v_i}(1..j)) \leq 12\phi$, output the sparse cut $C = \tilde{\pi}_t^{v_i}(1..j)$ and halt.
    end for
  end for
end for
Return *failed*.

From Lemma 3.1 we know that we can obtain a cut C with some good properties if we start the truncated random walk at a vertex $v \in S_b^g$ with parameter ϵ_b. Therefore, what we do in `Distributed Nibble` is to just sample sufficiently many vertices as the starting points of random walks so that with sufficiently high probability at least one them is in the set S_b^g. The danger here is that calculating all these random walk

[4] There are many versions of the paper [41] available; we refer to `https://arxiv.org/abs/cs/0310051v9`.

distributions simultaneously may be infeasible if any part of the graph becomes *too congested*.

In this section we analyze the behavior of `Distributed Nibble` (as a sequential algorithm) and prove that it operates correctly. In Section 3.2 we argue that `Distributed Nibble` can be implemented efficiently in the CONGEST model, in $\text{poly}(\log m, 1/\phi)$ time.

Roughly speaking, Lemma 3.2 shows that if the sets $\pi(1..j)$ and $\pi(1..j')$ have similar volume, then the cuts resulting from these two sets have similar sparsity. This justifies lines (3) and (4) of `Distributed Nibble` and allows us to examine a small number of prefixes of the permutation $\tilde{\pi}_t^{v_i}$.

LEMMA 3.2. *Let π be any permutation, and let $\phi \leq 1/12$. If, for some index j, $\Phi(\pi(1..j)) \leq \phi$ and $\text{Vol}(\pi(1..j)) \leq (5/6)\text{Vol}(V)$, then $\Phi(\pi(1..j')) \leq 12\phi$ for all indices $j' > j$ such that*

$$\text{Vol}(\pi(1..j')) \leq (1+\phi)\text{Vol}(\pi(1..j)).$$

Proof. Let $x = \text{Vol}(\pi(1..j))$ and $y = \text{Vol}(\pi(1..j'))$. Recall that $2m = 2|E| = \text{Vol}(V)$, and so $x \leq (5/6)\text{Vol}(V) = (5/6)2m$. We have $x \leq y \leq (1+\phi)x$. Since $x \leq (5/6)2m$ and $\phi \leq 1/12$, we have $\phi x \leq x/12 \leq (2m-x)/2$. Therefore,

$$2m - y \geq 2m - x - \phi x \geq (2m-x)/2.$$

We calculate an upper bound of $\Phi(\pi(1..j'))$ as follows.

$$\begin{aligned}
\Phi(\pi(1..j')) &= \frac{\partial(\pi(1..j'))}{\min(y, 2m-y)} \\
&\leq \frac{\partial(\pi(1..j)) + \sum_{i=j+1}^{j'} d(\pi(i))}{\min(x, (2m-x)/2)} \\
&\leq \frac{\partial(\pi(1..j)) + \phi x}{\min(x, (2m-x))/2} \\
&\leq 12\phi.
\end{aligned}$$

We explain the details of the derivation. The first inequality is due to $x \leq y$ and $(2m-x)/2 \leq 2m-y$, which follow from the above discussion. The second inequality is due to the fact that $\sum_{i=j+1}^{j'} d(\pi(i)) = \text{Vol}(\pi(1..j')) - \text{Vol}(\pi(1..j)) \leq \phi \cdot \text{Vol}(\pi(1..j)) = \phi x$. For the third inequality, note that $\frac{\partial(\pi(1..j))}{\min(x,(2m-x))} \leq \phi$ and $\frac{\phi x}{\min(x,(2m-x))} \leq 5\phi$, since $x \leq (5/6)2m$. □

LEMMA 3.3. *Let $S \subset V$ be any subset satisfying*

$$\text{Vol}(S) \leq (2/3)\text{Vol}(V) \quad \textit{and} \quad \Phi(S) \leq \frac{\phi^3}{19208 \ln^2(me^4)}.$$

Then there exists a number b such that $\text{Vol}(S_b^g) \geq 2^b/32$.

Proof. Denote $x = \text{Vol}(S)$. From Condition (iii) of Lemma 3.1 we deduce that if $S_b^g \neq \emptyset$, then there exists a set of vertices C such that $\text{Vol}(S) \geq \text{Vol}(C \cap S) \geq (4/7)2^{b-1}$. Thus, for all b such that $b \geq \lceil \log x \rceil + 2$, we must have $S_b^g = \emptyset$. If the statement of this lemma is false, i.e., $\text{Vol}(S_b^g) < 2^b/32$ for all b, then

$$\text{Vol}(S^g) \leq \sum_{b=1}^{\lceil \log x \rceil + 1} \frac{2^b}{32} < \frac{2^{\lceil \log x \rceil + 2}}{32} < x/4,$$

which contradicts the requirement $\text{Vol}(S^g) \geq \text{Vol}(S)/2$ specified in Lemma 3.1. □

LEMMA 3.4. (CORRECTNESS) *For any $\phi \leq 1/12$, if there exists a subset $S \subset V$ satisfying*

$$\text{Vol}(S) \leq (2/3)\text{Vol}(V) \quad \textit{and} \quad \Phi(S) \leq \frac{\phi^3}{19208 \ln^2(me^4)},$$

then `Distributed Nibble` outputs a set of vertices C such that $\Phi(C) \leq 12\phi$ with probability at least $1 - 1/\text{poly}(m)$.

Proof. From Lemma 3.3 we know there exists a number b such that $\text{Vol}(S_b^g) \geq 2^b/32$. Since we sample v_i proportional to the degree distribution,

$$\Pr[v_i \in S_b^g] = \frac{\text{Vol}(S_b^g)}{\text{Vol}(V)} \geq \frac{2^b}{32 \cdot \text{Vol}(V)}.$$

Since we sample $K = c \log m \cdot \frac{\text{Vol}(V)}{2^b}$ number of vertices,

$$\begin{aligned}
\Pr[\exists i \text{ s.t. } v_i \in S_b^g] &\geq 1 - \left(1 - \frac{2^b}{32\text{Vol}(V)}\right)^{c \log m \frac{\text{Vol}(V)}{2^b}} \\
&\geq 1 - m^{-\Omega(c)}.
\end{aligned}$$

Now we focus on the truncated random walk starting at this vertex $v_i \in S_b^g$. We fix two numbers $t \in [1, t_0]$ and j such that the four conditions in Lemma 3.1 are satisfied. In particular, Condition (i) and Condition (iv) in Lemma 3.1 say that

$$\begin{aligned}
\text{Vol}(\tilde{\pi}_t^{v_i}(1..j)) &\leq (5/6)\text{Vol}(V), \\
\Phi(\tilde{\pi}_t^{v_i}(1..j)) &\leq \phi.
\end{aligned}$$

Therefore, we are able to apply Lemma 3.2, and so we have $\Phi(\tilde{\pi}_t^{v_i}(1..j')) \leq 12\phi$ for all indices j' such that $\text{Vol}(\tilde{\pi}_t^{v_i}(1..j)) \leq \text{Vol}(\tilde{\pi}_t^{v_i}(1..j')) \leq (1+\phi)\text{Vol}(\tilde{\pi}_t^{v_i}(1..j))$.

In `Distributed Nibble`, we search for a cut with target volume $(1+\phi)^x$, for all possible integers x. Note that Condition (ii) in Lemma 3.1 implies $j \leq j_{max}$. Therefore, in Step (3) of `Distributed Nibble`, at least one index $j^\star$ picked by the algorithm satisfies

$$\text{Vol}(\tilde{\pi}_t^{v_i}(1..j)) \leq \text{Vol}(\tilde{\pi}_t^{v_i}(1..j^\star)) \leq (1+\phi)\text{Vol}(\tilde{\pi}_t^{v_i}(1..j)).$$

By Lemma 3.2, the cut $C = \tilde{\pi}_t^{v_i}(\{1, ..., j^\star\})$ associated with this index $j^\star$ found in Step (4) meets the requirement $\Phi(C) \leq 12\phi$ of the lemma. □

3.2 Implementation We show how to implement `Distributed Nibble` in the CONGEST model. The goal of this section is to prove Lemma 3.5. Note that Lemma 2.5 is a consequence of Lemmas 3.4 and 3.5.

LEMMA 3.5. *`Distributed Nibble` can be implemented in the* CONGEST *model using $O(D + \log^9 m/\phi^{10})$ rounds, with success probability $1 - 1/\mathrm{poly}(m)$, where D is the diameter of graph. If `Distributed Nibble` outputs a set C successfully, then each vertex knows whether or not it belongs to C.*

To prove this lemma, we shall analyze `Distributed Nibble` step by step.

LEMMA 3.6. (STEP (1)) *The samples for every level b (from 1 to $\lceil \log m \rceil$) can be generated in $O(D + \log m)$ time.*

Proof. We build a BFS tree rooted at an arbitrary vertex x. For each vertex v, define $s(v)$ as the sum of $d(u)$ for each u in the subtree rooted at v. In $O(D)$ rounds we can let each vertex v learn the number $s(v)$ by a bottom-up traversal of the BFS tree.

In the beginning, for each $b = 1, \ldots, \lceil \log m \rceil$, we generate $K_b = c \log m \frac{\mathrm{Vol}(V)}{2^b}$ number of b-tokens at the root x. Let $L = \Theta(D)$ be the number of layers in the BFS tree. For $i = 1, \ldots, L$, the vertices of layer i do the following. When a b-token arrives at v, the token disappears at v with probability $d(v)/s(v)$ and v includes itself in the bth sample; otherwise, v sends the token to a child u with probability $\frac{s(u)}{s(v)-d(v)}$. Note that v only needs to tell each child u how many b-tokens u gets. Thus, for each b, the process of choosing $K_b = c \log m \frac{\mathrm{Vol}(V)}{2^b}$ vertices from the degree distribution can be done in L rounds. By pipelining, we can do this for all b in $O(D + \log m)$ rounds.

This method has the virtue of selecting exactly K_b vertices in the bth sample. We can also select K_b vertices in expectation, in just $O(D)$ time, simply by computing $\mathrm{Vol}(V)$ with a BFS tree, disseminating it to all vertices, and letting each v join the sample independently with probability $K_b \deg(v)/\mathrm{Vol}(V)$. □

It is not obvious why Step (2) of `Distributed Nibble` should be efficiently implementable in the CONGEST model. Before analyzing it, we give some helpful lemmas about lazy random walks.

LEMMA 3.7. ([41]) *For all u, v, and t, $\rho_t^v(u) = \rho_t^u(v)$.*

Proof. This lemma was observed in [41] without proof. For the sake of completeness, we provide a short proof here. A sequence of vertices $W = (x_0, x_1, \ldots, x_t)$ is called a *walk* of length t if $x_{i+1} \in N(x_i) \cup \{x_i\}$ for each $i \in [0, t)$. We write $\Pr[W]$ to be the probability that the first t steps of a lazy random walk starting at x_0 tracks W. Let $W^R = (x_t, x_{t-1}, \ldots, x_0)$ be the reversal of W.

Let $\mathcal{W}_t^{u,v}$ be the set of walks of length t starting at u and ending at v. It is clear that $\rho_t^u(v) = \sum_{W \in \mathcal{W}_t^{u,v}} \Pr[W]/d(v)$ and $\rho_t^v(u) = \sum_{W \in \mathcal{W}_t^{v,u}} \Pr[W]/d(u)$. Since $\mathcal{W}_t^{v,u} = \{W^R \mid W \in \mathcal{W}_t^{u,v}\}$, to prove the lemma it suffices to show that $\Pr[W]/d(v) = \Pr[W^R]/d(u)$ for each $W \in \mathcal{W}_t^{u,v}$.

Fix any $W \in \mathcal{W}_t^{u,v}$ and let $W_* = (y_0, \ldots, y_s)$ be the subsequence of W resulting from splicing out immediate repetitions in W. It is clear that $\Pr[W] = 2^{-t} \cdot \prod_{i=0}^{s-1} 1/d(y_i)$, and so

$$\frac{\Pr[W]}{d(v)} = \frac{\Pr[W]}{d(y_s)} = 2^{-t} \cdot \prod_{i=0}^{s} \frac{1}{d(y_i)} = \frac{\Pr[W^R]}{d(y_0)} = \frac{\Pr[W]}{d(u)}.$$

□

LEMMA 3.8. *Fix the parameter b (which influences ϵ_b and hence the truncation operation of the random walk) and define*

$$Z_t(u) = \{v_i \mid v_i \text{ is in the } b\text{th sample and } \tilde{p}_{t-1}^{v_i}(u) > 0\}.$$

For every vertex u and every t, with high probability, $|Z_t(u)| \le O(\log^3 m/\phi^3)$.

Proof. Define $S = \{v \in V \mid \tilde{p}_{t-1}^v(u) > 0\}$. By definition $Z_t(u) = S \cap \{v_1, \ldots, v_{K_b}\}$. For each $v \in S$, we have $p_{t-1}^v(u) \ge \tilde{p}_{t-1}^v(u) \ge 2\epsilon_b d(u)$. Recall that p_{t-1} is the probability distribution obtained after $t-1$ steps of the lazy random walk without truncation. By Lemma 3.7,

$$p_{t-1}^u(v) = (p_{t-1}^v(u)/d(u))d(v) \ge 2\epsilon_b \cdot d(v).$$

Therefore, $2\epsilon_b \cdot \mathrm{Vol}(S) \le \sum_{v \in S} p_{t-1}^u(v) \le 1$, and so $\mathrm{Vol}(S) \le \frac{1}{2\epsilon_b}$, which implies

$$\Pr[v_i \in S] \le \frac{1}{2\epsilon_b \cdot \mathrm{Vol}(V)}.$$

Recall that $t_0 = \frac{49 \ln(me^4)}{\phi^2}$ and $\epsilon_b = \frac{\phi}{56 \ln(me^4) t_0 2^b}$. Rewrite the number $K_b = c \log m \frac{\mathrm{Vol}(V)}{2^b}$ as $K_b = \Theta(\epsilon_b \cdot \mathrm{Vol}(V) \cdot \log^3 m/\phi^3)$. Since each of $v_1, \ldots, v_{K_b}$ is chosen independently, using a Chernoff bound we conclude that there exists a constant $c' > 0$ depending on c such that

$$\Pr[|Z_t(u)| > c' \log^3 m/\phi^3] \le \exp(-\Omega(\log^3 m/\phi^3)).$$

□

LEMMA 3.9. (STEP (2)) *Fix the parameter b. Suppose each vertex v knows $\tilde{p}_{t-1}^{v_i}(v)$, for all v_i in the bth sample. Then with high probability, each vertex v can calculate $\tilde{p}_t^{v_i}(v)$, for all v_i, within $O(\log^3 m/\phi^3)$ rounds.*

Proof. The normal way to calculate $[T\tilde{p}_{t-1}(u)]_{\epsilon_b}$ is as follows. For each v_i, each vertex v broadcasts the number $\frac{\tilde{p}^{v_i}_{t-1}(v)}{2d(v)}$ to all its neighbors, and then v collects messages from neighbors. The vertex v can calculate $\tilde{p}^{v_i}_t(v)$ locally by adding $\tilde{p}^{v_i}_{t-1}(v)/2$ and all numbers received from its neighbors, then applying the truncation operation. Note that a straightforward analysis of this protocol leads to a terrible round complexity, since we have to do this for each v_i.

One crucial observation is that a vertex v does not need to care about those v_i with $\tilde{p}^{v_i}_{t-1}(v) = 0$ at time t. We modify this protocol a little bit in such a way that we never send a number if it is 0. Define $Z_t(u) = \{v_i \mid \tilde{p}^{v_i}_{t-1}(u) > 0\}$ as in Lemma 3.8, and so each vertex v only needs to spend $|Z_t(v)|$ rounds to simulate the time step t of the lazy random walk. By Lemma 3.8 and the discussion above, we have proved that Step (2) can be executed in $O(\log^3 m/\phi^3)$ time, for every v_i and any specific t. □

LEMMA 3.10. (STEPS (3,4)) *Fix parameters b, t and x. Steps (3) and (4) can be implemented in $O(\log^6 m/\phi^7)$ rounds for all v_i in the bth sample. For any sparse cut C found in Step (4), every vertex in C knows that it belongs to C.*

Proof. Now we focus on the random walk starting at v_i. Let $U = \{u \mid \exists t' \le t,\ \tilde{p}^{v_i}_{t'}(u) > 0\}$. We claim that U is a connected vertex set. Suppose U is disconnected. Let W be a connected subset of U such that $v_i \notin W$. Let t' be the minimum number such that there exists a vertex $u' \in W$ with $\tilde{p}^{v_i}_{t'}(u') > 0$. By our choice of u', there is no neighbor v' of u' such that $\tilde{p}^{v_i}_{t'-1}(v') > 0$, and this contradicts the fact that $\tilde{p}^{v_i}_{t'}(u') > 0$. Therefore, U must be a connected vertex set.

Obviously all $\tilde{\pi}^{v_i}_t(j)$ for $j \le j_{max}$ are in U. We build a BFS tree of U rooted at v_i, which has $t+1$ levels. We will execute Step (3) and Step (4) by sending requests from the root to all vertices in U, collecting information from U to the root, and making a decision locally at the root. Recall that each v_i has its own BFS tree, and in general a vertex u belongs to multiple BFS trees for different v_i. Luckily, each vertex u only belongs to the BFS tree of those $v_i \in \bigcup_{1\le t\le t_0+1} Z_t(u)$, so with only a $(t_0+1)\cdot \max_{u,t}|Z_t(u)| = O(\log^4 m/\phi^5)$ overhead of running time, we can do Step (3) and Step (4) for all v_i in parallel.

To find each index j specified in Step (3), we can do a "random binary search" on vertices in U. Let $\tilde{\pi} = \tilde{\pi}^{v_i}_t$ and $\tilde{\rho} = \tilde{\rho}^{v_i}_t$ be with respect to $\tilde{p}^{v_i}_t$. Note that by our choice of U we can assume U is a prefix set of $\tilde{\pi}$. We maintain two indices L and R that control the search space. Initially, $L \leftarrow 1$ and $R \leftarrow |U|$. In each iteration, we randomly pick one vertex $\tilde{\pi}(j)$ among $\tilde{\pi}(L..R)$ and calculate $\mathrm{Vol}(\tilde{\pi}(1,\cdots,j))$ by broadcasting $\tilde{\rho}(\tilde{\pi}(j))$ to all vertices in U and propagating information up the BFS tree.[5] If $\mathrm{Vol}(\tilde{\pi}(1,\cdots,j)) \le (1+\phi)^x$, we update $L \leftarrow j$; otherwise we let $R = j-1$. In each iteration, with probability $1/2$ we sample j in the middle half of $[L,R]$ and the size of search space $[L,R]$ shrinks by a factor of at least $3/4$. Therefore, w.h.p., after $O(\log m)$ iterations, we will have isolated $L = R = j$. Each iteration can be done in $O(t) = O(t_0)$ rounds. Due to the congestion overhead, Step (3) can be implemented in $O(\log m \cdot t_0 \cdot \log^4 m/\phi^5) = O(\log^6 m/\phi^7)$ rounds.

Step (4) can be done by simply collecting information about $\partial(\tilde{\pi}^{v_i}_t(1..j))$ and $\mathrm{Vol}(\tilde{\pi}^{v_i}_t(1..j))$; its round complexity is of a lower order than that of Step (3). If the root v_i finds a cut C with $\Phi(C) \le 12\phi$, it broadcasts $\tilde{\rho}(\tilde{\pi}(j))$ to all vertices in U to let the vertices in C know that they are in C. Note that for each vertex u in U, it can infer whether it is in C by comparing $\tilde{\rho}(u)$ and $\tilde{\rho}(\tilde{\pi}(j))$. □

Proof. [Proof of Lemma 3.5] Combining Lemmas 3.6, 3.9, and 3.10, the running time in Step (1) is $O(D+\log m)$, Step (2) is $O(\log^5 m/\phi^5)$, and Steps (3) and (4) are $O(\log^9 m/\phi^{10})$. The dominating term $O(\log^9 m/\phi^{10})$ comes from enumerating $\log m \cdot t_0 \cdot \log m/\phi = \Theta(\log^3 m/\phi^3)$ combinations of (b,t,x), spending $O(\log^6 m/\phi^7)$ rounds for each combination.

Whenever a vertex v_i finds a sparse cut C, it broadcasts a message to the entire graph saying that it has found a cut, and this takes $O(D)$ rounds. If multiple cuts are found by different vertices, we can select exactly one cut, breaking ties arbitrarily. A more opportunistic version of the algorithm could also take a maximal independent set of compatible cuts. □

4 Triangle Enumeration

We use the routing algorithm from [17,18]. Theorem 4.1 was first stated in [17, Theorem 1.2] with round complexity $\tau_{\mathrm{mix}}(G)\cdot 2^{O(\sqrt{\log n \log\log n})}$; this was recently improved to $\tau_{\mathrm{mix}}(G)\cdot 2^{O(\sqrt{\log n})}$ in [18].

THEOREM 4.1. ([17, 18]) *Consider a graph $G = (V,E)$ and a set of point-to-point routing requests, each given by the* IDs *of the corresponding source-destination pair. If each vertex v is the source and the destination of at most $\deg(v)\cdot 2^{O(\sqrt{\log n})}$ messages, there is a randomized distributed algorithm that delivers all messages in $\tau_{\mathrm{mix}}(G)\cdot 2^{O(\sqrt{\log n})}$ rounds, w.h.p., in the* CONGEST *model.*

[5] Each vertex $u \in U$ does not know the index j such that $u = \tilde{\pi}(j)$, so we cannot do the search deterministically.

REMARK 4.1. *The claim of Theorem 4.1 appears to be unproven for arbitrary* ID*-assignments (Hsin-Hao Su, personal communication, 2018), but is true for well-behaved* ID*-assignments, which we illustrate can be computed efficiently in* CONGEST*. In [17, 18] each vertex $v \in V$ simulates $\deg(v)$ virtual vertices in a random graph G_0 which is negligibly close to one drawn from the Erdős-Rényi distribution $\mathcal{G}(2m, p)$ for some p. Presumably the IDs of v's virtual vertices are $(\mathrm{ID}(v), 1), \ldots, (\mathrm{ID}(v), \deg(v))$. It is proven [17, 18] that effecting a set of routing requests in G_0 takes $2^{O(\sqrt{\log n})}$ time in G_0; however, to translate a routing request $\mathrm{ID}(x) \rightsquigarrow \mathrm{ID}(y)$ in G to G_0, it seems necessary to map it (probabilistically) to $(\mathrm{ID}(x), i) \rightsquigarrow (\mathrm{ID}(y), j)$, where i, j are chosen uniformly at random from $[1, \deg(x)]$ and $[1, \deg(y)]$, respectively. (This is important for the global congestion guarantee that y's virtual nodes receive roughly equal numbers of messages from all sources.) This seems to require that x know how to compute $\deg(y)$ or an approximation thereof based on $\mathrm{ID}(y)$. Arbitrary* ID*-assignments obviously do not betray this information.*

LEMMA 4.1. *In $O(D + \log n)$ time we can compute an* ID *assignment $\mathrm{ID} : V \to \{1, \ldots, |V|\}$ and other information such that $\mathrm{ID}(u) < \mathrm{ID}(v)$ implies $\lfloor \log \deg(u) \rfloor \leq \lfloor \log \deg(v) \rfloor$, and any vertex u can locally compute $\lfloor \log \deg(v) \rfloor$ for any v.*

Proof. Build a BFS tree from an arbitrary vertex x in $O(D)$ time. In a bottom-up fashion, each vertex in the BFS tree calculates the *number* of vertices v in its subtree having $\lfloor \log \deg(v) \rfloor = i$, for $i = 0, \ldots, \log n$. This takes $O(D + \log n)$ time by pipelining. At this point the root x has the counts $n_0, \ldots, n_{\log n}$ for each degree class, where $n = \sum_i n_i$. It partitions up the ID-space so that all vertices in class-0 get IDs from $[1, n_0]$, class-1 from $[n_0+1, n_0+n_1]$, and so on. The root broadcasts the numbers $n_0, \ldots, n_{\log n}$, and disseminates the IDs to all nodes according to their degrees. (In particular, the root gives each child $\log n$ intervals of the ID-space, which they further subdivide, sending $\log n$ intervals to the grandchildren, etc.) With pipelining this takes another $O(D + \log n)$ time. Clearly knowing $n_0, \ldots, n_{\log n}$ and $\mathrm{ID}(v)$ suffice to calculate $\lfloor \log \deg(v) \rfloor$. □

Lemma 4.1 gives us a good ID-assignment to apply Theorem 4.1. It is also useful in our triangle enumeration application. Roughly speaking, vertices with larger degrees also have more bandwidth in the CONGEST model, and therefore should be responsible for learning about larger subgraphs and enumerating more triangles.

Before we present our triangle enumeration algorithm for general graphs, we address the important special case of finding triangles with at least one edge in a component of high conductance.

4.1 Triangle Enumeration in High Conductance Graphs Recall that our graph decomposition routine returns a tripartition $E_m \cup E_s \cup E_r$. Triangles that intersect E_s will be enumerated separately. The purpose of this section is to provide a routine to enumerate triangles that intersect E_m but not E_s, i.e., they are (i) completely contained in E_m or (ii) have at least one edge in E_m and E_r.[6] Whereas each component of E_m has low mixing time, we can say nothing about the mixing time of a component of E_m plus all incident E_r edges.

Definitions. The underlying network is $G = (V, E)$. The input is a subgraph $G_{\mathrm{in}} = (V_{\mathrm{in}}, E_{\mathrm{in}})$ with low mixing time, together with some additional edges E_{out} joining vertices in V_{in} to V. Let $\deg_{\mathrm{in}}(v)$ and $\deg_{\mathrm{out}}(v)$ be the number of E_{in} and E_{out} edges incident to v. In this section we write $n = |V|$ and $m_{\mathrm{in}} = |E_{\mathrm{in}}|$. We assume $V = V(E_{\mathrm{in}} \cup E_{\mathrm{out}})$. Note that Condition (i) of Theorem 4.2 implies that $m_{\mathrm{in}} \leq |E_{\mathrm{in}} \cup E_{\mathrm{out}}| \leq 3m_{\mathrm{in}}$.

THEOREM 4.2. *Suppose that G_{in} and E_{out} meet the following conditions:*

(i) For each $v \in V_{\mathrm{in}}$, $\deg_{\mathrm{in}}(v) \geq \deg_{\mathrm{out}}(v)$.

(ii) $\tau_{\mathrm{mix}}(G_{\mathrm{in}}) = n^{o(1)}$.

In the CONGEST *model, all triangles in $E_{\mathrm{in}} \cup E_{\mathrm{out}}$ can be counted and enumerated, w.h.p., in $O(n^{1/3+o(1)})$ rounds.*

Note that Theorem 4.2 applies to the class of graphs with $n^{o(1)}$ mixing time by setting $E_{\mathrm{out}} = \emptyset$. We first describe the algorithm behind Theorem 4.2 and then analyze it in Lemmas 4.2–4.5.

The Easy Case. We first check whether any vertex $v^\star \in V(E_{\mathrm{in}} \cup E_{\mathrm{out}})$ has

$$\deg_{\mathrm{in}}(v^\star) + \deg_{\mathrm{out}}(v^\star) \geq m/(40n^{1/3} \log n) = \zeta.$$

If so, we apply Theorem 4.1 to the subgraph G^+_{in} induced by E_{in} and all edges E_{out} incident to $v^\star$, and have every vertex $u \in V_{\mathrm{in}}$ transmit to $v^\star$ all its incident edges in $E_{\mathrm{in}} \cup E_{\mathrm{out}}$.[7] Condition (i) of Theorem 4.2

[6] Our algorithm for constructing the tripartition $E_m \cup E_s \cup E_r$ satisfies the property that there is no triangle with two edges in E_m and one edge in E_r. However, all algorithms in Section 4 do not rely on this property.

[7] Note that when $v^\star \in V \setminus V_{\mathrm{in}}$, $\tau_{\mathrm{mix}}(G_{\mathrm{in}}) = n^{o(1)}$ implies that $\tau_{\mathrm{mix}}(G^+_{\mathrm{in}}) = n^{o(1)}$ as well. This follows from the observation that the conductance $\Phi_{G^+_{\mathrm{in}}}$ of G^+_{in} is at least 1/3 of the conductance Φ_G of G_{in}. Pick a sparsest cut $(S, \bar{S})$ of G^+_{in}, and let $(S', \bar{S}')$

implies that $|E_{\text{in}} \cup E_{\text{out}}| \leq 3m_{\text{in}}$, so the total volume of messages entering $v^\star$ is $O(m_{\text{in}})$. Therefore the routing takes $O(\tau_{\text{mix}}(G^+_{\text{in}}) \cdot 2^{O(\sqrt{\log n})} \cdot m/\zeta) = n^{1/3+o(1)}$, and thereafter, $v^\star$ can report all triangles in $E_{\text{in}} \cup E_{\text{out}}$. In the analysis of the following steps, we may assume that the maximum degree in the graph induced by $E_{\text{in}} \cup E_{\text{out}}$ is at most $m_{\text{in}}/(40n^{1/3} \log n)$.

Vertex Classes. Let $\delta = 2^{\lfloor \log(2m_{\text{in}}/n) \rfloor}$ be the average number of incident edges in E_{in} among all n vertices V, rounded down to the nearest power of 2. Write $\deg_{\text{in}}(v) = k_v \cdot \delta$, and call v a *class-0 vertex* if $k_v \in [0, 1/2)$ and a *class-i vertex* if $k_v \in [2^{i-2}, 2^{i-1})$. We use the fact that

$$\sum_{v \in V_{\text{in}} \,:\, k_v \geq 1/2} 2k_v \geq n.$$

By applying Lemma 4.1 to reassign IDs, we may assume that the ID-space of V_{in} is $\{1, \ldots, |V_{\text{in}}|\}$ and that any vertex can compute the class of v, given $\text{ID}(v)$.

Randomized Partition. Our algorithm is a randomized adaptation of the CONGESTED-CLIQUE algorithm of [9]. We partition the vertex set V into $V_1 \cup \cdots \cup V_{n^{1/3}}$ locally, without communication. Each vertex $v \in V$ selects an integer $r_v \in [1, n^{1/3}]$ uniformly at random, joins V_{r_v}, and transmits 'r_v' to its immediate neighbors in V_{in}. We allocate the (less than) n triads

$$\mathcal{T} = \left\{ (j_1, j_2, j_3) \mid 1 \leq j_1 \leq j_2 \leq j_3 \leq n^{1/3} \right\}$$

to the vertices in V_{in} in the following way. Enumerate the vertices in increasing order of ID. If v is class-0, then skip v. If v is class-i, $i \geq 1$, then $k_v < 2^{i-1}/\delta$. Allocate to v the next $2^i/\delta \geq 2k_v$ triads from $\mathcal{T}$, and stop whenever all triads are allocated.

We use Lemma 4.1 to generate the IDs of vertices in V_{in}. In view of how vertex class is defined, Lemma 4.1 guarantees that each vertex $v \in V_{\text{in}}$ knows the class of all vertices in V_{in}, and can therefore perform this allocation locally, without communication.

A vertex $v \in V$ that is assigned a triad (j_1, j_2, j_3) is responsible for learning the set of all edges $E(V_{j_1}, V_{j_2}) \cup E(V_{j_2}, V_{j_3}) \cup E(V_{j_1}, V_{j_3})$ and reporting/counting those triangles (x_1, x_2, x_3) with $x_k \in V_{j_k}$.[8]

Transmitting Edges. Every vertex $v \in V_{\text{in}}$ knows the IDs of all its neighbors in V and which part of the vertex partition they are in. For each $v \in V_{\text{in}}$, each incident edge $(v, u) \in E_{\text{in}} \cup E_{\text{out}}$, and each index $r^* \in [1, n^{1/3}]$, v transmits the message "$(v, u), r_v, r_u$" to the unique vertex x handling the triad on $\{r_u, r_v, r^*\}$. Observe that the total message volume is exactly $\Theta(m_{\text{in}} n^{1/3})$.

We analyze the behavior of this algorithm in the CONGEST model, where the last step is implemented by applying Theorem 4.1 to G_{in}. Recall from Condition (i) of Theorem 4.2 that the number of edges in the graph we consider, $\bar{m} = |E_{\text{in}} \cup E_{\text{out}}|$, is in the range $[m_{\text{in}}, 3m_{\text{in}}]$.

LEMMA 4.2. *Consider a graph with $\bar{m}$ edges and $\bar{n}$ vertices. We generate a subset S by letting each vertex join S independently with probability p. Suppose that the maximum degree is $\Delta \leq \bar{m}p/20 \log \bar{n}$ and $p^2\bar{m} \geq 400 \log^2 \bar{n}$. Then, with probability at least $1 - 10(\log \bar{n})/\bar{n}^5$, the number of edges in the subgraph induced by S is at most $6p^2\bar{m}$.*

Proof. For an edge e_i, define $x_i = 1$ if both two endpoints of edge e_i join S, otherwise $x_i = 0$. Then $X = \sum_{i=1}^{\bar{m}} x_i$ is the number of edges in the subgraph induced by S. We have $\mathbb{E}[X] = p^2\bar{m}$, and by Markov's inequality,

$$\Pr[X \geq 6\,\mathbb{E}[X]] = \Pr[X^c \geq (6\,\mathbb{E}[X])^c] \leq \frac{1}{6^c} \frac{\mathbb{E}[X^c]}{p^{2c}\bar{m}^c},$$

where $c = 5 \log \bar{n}$ is a parameter.

$$\begin{aligned} \mathbb{E}[X^c] &= \sum_{i_1, \ldots, i_c \in [1, \bar{m}]} \mathbb{E}\left[\prod_{j=1}^{c} x_{i_j} \right] \\ &= \sum_{k=2}^{2c} f_k \cdot p^k, \end{aligned}$$

where f_k is the number of choices $\{i_1, \ldots, i_c \in [1, \bar{m}]\}$ such that the number of distinct endpoints in the edge set $e_{i_1}, \ldots, e_{i_c}$ is k.

For any choice of $(i_1, \ldots, i_c \in [1, \bar{m}])$, we project it to a vector $\langle k_1, \ldots, k_c \rangle \in \{0, 1, 2\}^c$, where k_j indicates the number of endpoints of e_{i_j} that overlap with the endpoints of the edges $e_{i_1}, \ldots, e_{i_{j-1}}$. Note that $2c - \sum k_j$ is the number of distinct endpoints in the edge set $\{e_{i_1}, \ldots, e_{i_c}\}$. We fix a vector $\langle k_1, \ldots, k_c \rangle$ and count how many choices of $(i_1, \ldots, i_c)$ project to this vector.

Suppose that the edges $e_{i_1}, \ldots, e_{i_{j-1}}$ are fixed. We bound the number of choices of e_{i_j} as follows. If $k_j = 0$, the number of choices is clearly at most m. If $k_j = 1$, the number of choices is at most $(2c)(p\bar{m}/20 \log \bar{n})$, since one of its endpoints (which overlaps with the

be the corresponding cut of G_{in}. The sparsity $\Phi(S)$ of $(S, \bar{S})$ must be at least 1/3 of the sparsity $\Phi(S')$ of $(S', \bar{S'})$, because $\text{Vol}(S) \leq 2|S'| + \text{Vol}(S') \leq 3\text{Vol}(S')$. Therefore, $\Phi_{G^+_{\text{in}}} = \Phi(S) \geq \frac{1}{3}\Phi(S') \geq \frac{1}{3}\Phi_{G_{\text{in}}}$.

[8] In the Triangle Counting application, it is important that v not count every triangle it is aware of. For example, if v is assigned (j, j, j'), v knows about triangles in the subgraph induced by V_j but should not count them; these triangles will be counted only by the vertex u that is assigned (j, j, j).

endpoints of the edges $e_{i_1}, \ldots, e_{i_{j-1}}$) has at most $2c$ choices, and the other endpoint (which does not overlap with the endpoints of the edges $e_{i_1}, \ldots, e_{i_{j-1}}$) has at most $\Delta \leq \bar{m}p/20\log\bar{n}$ choices. If $k_j = 2$, the number of choices is at most $(2c)^2$.

Based on the above calculation, we upper bound f_k as follows. In the calculation, x is the number of indices j such that $k_j = 1$, and y is the number of indices j that $k_j = 2$. Note that $\binom{c}{x}\binom{c-x}{y}$ is the number of distinct vectors $\langle k_1, \ldots, k_c\rangle$ realizing the given parameters c, x, and y. The number f_k is at most

$$\begin{aligned}
&\sum_{\substack{x+y\leq c\\ 2c-x-2y=k}} \bar{m}^{c-x-y}\binom{c}{x}\binom{c-x}{y}\left(\frac{2cp\bar{m}}{20\log\bar{n}}\right)^x (4c^2)^y \\
&\leq \sum_{\substack{x+y\leq c\\ 2c-x-2y=k}} \bar{m}^c 3^c \left(\frac{2cp}{20\log\bar{n}}\right)^x \left(\frac{4c^2}{\bar{m}}\right)^y \\
&\leq \sum_{\substack{x+y\leq c\\ 2c-x-2y=k}} (3\bar{m})^c \left(\frac{2cp}{20\log\bar{n}}\right)^{x+2y} \\
&\leq c(3\bar{m})^c \left(\frac{2cp}{20\log\bar{n}}\right)^{2c-k}.
\end{aligned}$$

The third inequality is due to the fact $p^2\bar{m} \geq 400\log^2\bar{n}$, which implies $(2cp/20\log\bar{n})^2 \geq (4c^2/\bar{m})$. Using the fact that $\frac{2c}{20\log\bar{n}} \leq 1/2$, we upper bound $\mathbb{E}[X^c]$ as follows.

$$\begin{aligned}
\mathbb{E}[X^c] &\leq \sum_{k=2}^{2c} f_k \cdot p^k \\
&= c(3\bar{m})^c p^{2c} \sum_{k=2}^{2c} \left(\frac{2c}{20\log\bar{n}}\right)^{2c-k} \\
&< 2c(3\bar{m})^c p^{2c}.
\end{aligned}$$

Therefore,

$$\Pr[X \geq 6\,\mathbb{E}[X]] \leq \frac{1}{6^c}\frac{\mathbb{E}[X^c]}{p^{2c}\bar{m}^c} \leq \frac{2c3^c}{6^c} \leq \frac{10\log\bar{n}}{\bar{n}^5}.$$

Note that the probability can be amplified to $\bar{n}^{-t}$ for any constant t by setting $c = t\log\bar{n}$ and using different constants in the statement of the lemma. □

LEMMA 4.3. *With probability at least $1 - 1/n^4$, we have $|E(V_{j_1}, V_{j_2})| \leq 6|E_{\text{in}} \cup E_{\text{out}}|/n^{2/3}$ for all $j_1, j_2 \in [1, n^{1/3}]$.*

Proof. Recall that each $v \in V$ joins the set V_i with probability $1/n^{1/3}$. Thus, the probability that a vertex $v \in V$ is in $V_{j_1} \cup V_{j_2}$ is at most $p = 2n^{-1/3}$.[9]

[9]For the case of $j_1 = j_2$, the probability is $n^{-1/3}$.

We apply Lemma 4.2 to the subgraph induced by $E_{\text{in}} \cup E_{\text{out}}$ having $\bar{m} = |E_{\text{in}} \cup E_{\text{out}}|$ edges and $\bar{n} = n$ vertices, with sampling probability $p = 2n^{-1/3}$ and $S = V_{j_1} \cup V_{j_2}$. By assumption, the maximum degree (of the subgraph induced by $E_{\text{in}} \cup E_{\text{out}}$) is at most $m_{\text{in}}p/(20\log n) \leq \bar{m}p/(20\log\bar{n})$, since otherwise we go to the easy case. The maximum degree upper bound implies $\bar{n} \geq (20\log\bar{n})/p$, and $p^2\bar{m} \geq (p\bar{n})^2 \geq 400\log^2\bar{n}$. By Lemma 4.2, we conclude that $\Pr[|E(V_{j_1}, V_{j_2})| > 6\bar{m}/n^{2/3}] \leq \frac{10\log n}{n^5}$. Note that $|E(V_{j_1} \cup V_{j_2})| \geq |E(V_{j_1}, V_{j_2})|$.

By a union bound over all $n^{2/3}$ choices of j_1 and j_2, the stated upper bound holds everywhere, with probability at least $1 - 1/n^4$. □

LEMMA 4.4. *With high probability, each vertex $v \in V_{\text{in}}$ receives $O(\deg_{\text{in}}(v) \cdot n^{1/3})$ edges.*

Proof. Consider any vertex $v \in V_{\text{in}}$. If $k_v < 1/2$, then v receives no message; otherwise v is responsible for between $2k_v$ and $4k_v$ triads, and v collects the edge set $E(V_{j_1}, V_{j_2})$ for at most $12k_v$ pairs of V_{j_1}, V_{j_2}. By Lemma 4.3, w.h.p., $|E(V_{j_1}, V_{j_2})| = O(m_{\text{in}}/n^{2/3})$. Remember that our choice of k_v implies $k_v = \Theta(\deg_{\text{in}}(v) \cdot n/m_{\text{in}})$, and so v receives

$$O(m_{\text{in}}/n^{2/3}) \cdot 12k_v = O(\deg_{\text{in}}(v) \cdot n^{1/3})$$

messages, with high probability. □

LEMMA 4.5. *Each vertex $v \in V_{\text{in}}$ sends $O(\deg_{\text{in}}(v) \cdot n^{1/3})$ edges with probability 1.*

Proof. By Condition (i) of Theorem 4.2, $v \in V_{\text{in}}$ is responsible for $\deg_{\text{in}}(v) + \deg_{\text{out}}(v) \leq 2\deg_{\text{in}}(v)$ incident edges, and each is involved in exactly $n^{1/3}$ triads. □

Lemmas 4.2–4.5 show that the message volume in to/out of every vertex is close to its expectation. By applying Theorem 4.1 and Lemma 4.1, all messages can be routed in $n^{1/3+o(1)}$ time. This concludes the proof of Theorem 4.2. Corollary 4.1 is a simple consequence of Theorem 4.2.

COROLLARY 4.1. *Let G be a graph with $\tau_{\text{mix}}(G) = n^{o(1)}$. In the* CONGEST *model, Triangle Detection, Enumeration, and Counting can be solved on G, with high probability, in $n^{1/3+o(1)}$ time.*

4.2 Triangle Enumeration and Counting in General Graphs The algorithm for Theorem 4.3 is based on an $n^{1/2}$-decomposition. Since the connected components induced by E_m have low mixing time, we can solve Triangle Enumeration/Counting on them very efficiently using Theorem 4.2, in $n^{1/3+o(1)}$ time, i.e., much less than the time required to compute the $n^{1/2}$-decomposition.

THEOREM 4.3. *In the* CONGEST *model, Triangle Detection, Counting, and Enumeration can be solved, w.h.p., in* $\tilde{O}(n^{1/2})$ *rounds.*

Proof. The underlying graph is $G = (V, E)$. We set the parameter $\delta = 1/2$. By Theorem 2.1, we compute an n^δ-decomposition $E = E_m \cup E_s \cup E_r$ using $\tilde{O}(n^{1-\delta})$ rounds. We divide the task of enumerating triangles into three cases. By ensuring that every triangle is output by exactly one vertex, this algorithm also solves Triangle Counting.

The algorithm has three steps. In the first step, we list all triangles intersecting E_s. In the second step, we identify a subset $E_r^{\text{new}} \subseteq E_m$, and in this step we list all triangles intersecting $E_m \setminus E_r^{\text{new}}$. At this point, all remaining triangles that are not yet listed are contained in $E_r^{\text{new}} \cup E_r$, and they will be listed in the third step.

Case 1: All Triangles Intersecting E_s. We handle this case as follows. By Condition (b) of Definition 1.2, $E_s = \bigcup_{v \in V} E_{s,v}$, where $\{E_{s,v}\}$ defines an acyclic n^δ-orientation. We let each v announce $E_{s,v}$ to all its neighbors, in $O(n^\delta)$ time. For the Triangle Counting application it is important that every triangle $\{x, y, z\}$ intersecting E_s be reported by exactly *one* vertex. If (x, y) and (x, z) are oriented and $\text{ID}(y) < \text{ID}(z)$, then y detects and reports the triangle. If (x, z) is oriented, $\{x, y\}$ is unoriented, and $\{y, z\}$ is unoriented or oriented as (z, y), then y detects and reports the triangle. If $(x, z), (y, z)$ are oriented but $\{x, y\}$ is not, and $\text{ID}(y) < \text{ID}(x)$, y reports the triangle.[10]

Case 2: Some Triangles Intersecting E_m. Consider a single connected component $G_{\text{in}} = (V_{\text{in}}, E_{\text{in}})$ induced by E_m, which has mixing time $n^{o(1)}$. We classify vertices in V_{in} as *good* or *bad* depending on whether they naturally satisfy Condition (i) of Theorem 4.2. A vertex is good if $\deg_{\text{in}}(v) \geq \deg_{E_r}(v)$. Let E_{out} be the subset of E_r-edges incident to *good* vertices in V_{in}, and let E_r^{new} be the subset of E_m-edges incident to *bad* vertices in V_{in}. We now apply Theorem 4.2 to enumerate/count all triangles in the edge set $E_{\text{in}} \cup E_{\text{out}}$.

Because triangles completely contained in E_r^{new} will also be found in Case 3, the Triangle *Counting* algorithm should refrain from including these in the tally for Case 2.

Case 3: Triangles Contained in $E_r^{\text{new}} \cup E_r$. Since each edge in E_r^{new} can be charged to an endpoint of an edge in E_r, we have $|E_r^{\text{new}} \cup E_r| \leq 3|E_r| \leq |E|/2$. We apply the algorithm recursively to the graph induced by $E_r^{\text{new}} \cup E_r$. The depth of the recursion is obviously at most $\log m$.

[10] Most of these cases do not occur in the parital orientations produced by our algorithm; nonetheless, they can occur in arbitrary partial acyclic orientations.

Round Complexity. Computing an n^δ-decomposition $E = E_m \cup E_s \cup E_r$ takes $\tilde{O}(n^{1-\delta})$ rounds. The algorithm for Case 1 takes $O(n^\delta)$ rounds. The algorithm for Case 2 takes $O(n^{1/3+o(1)})$ rounds. The number of recursive calls (Case 3) is $\log m$. Thus, the overall round complexity is

$$\log m \cdot \left(O(n^\delta) + \tilde{O}(n^{1-\delta}) + O(n^{1/3+o(1)})\right) = \tilde{O}(n^{1/2}).$$

□

4.3 Subgraph Enumeration In this section we show that Corollary 4.1 can be extended to enumerating s-vertex subgraphs in $O(n^{(s-2)/s+o(1)})$ rounds. Note that the $\Omega(n^{1/3}/\log n)$ lower bound for triangle enumeration on Erdős-Rényi graphs $\mathcal{G}(n, 1/2)$ [22] can be generalized to an $\Omega(n^{(s-2)/s}/\log n)$ lower bound for enumerating s-vertex subgraphs. This implies that Theorem 4.4 is nearly optimal on $\mathcal{G}(n, 1/2)$.

THEOREM 4.4. *Let* $s = O(1)$ *be any constant. Given a graph* G *of* n *vertices with* $\tau_{\text{mix}}(G) = n^{o(1)}$*, we can list all* s*-vertex subgraphs of* G *in* $O(n^{(s-2)/s+o(1)})$ *rounds, w.h.p., in the* CONGEST *model.*

It has been shown in [9] that listing all s-vertex subgraphs of G can be done in $O(n^{(s-2)/s}/\log n)$ rounds in the deterministic CONGESTED-CLIQUE model. This result, together with the routing algorithm of Lemma 4.1, does not immediately imply Theorem 4.4, since $\deg(v)$ could be much less than n.

Theorem 4.4 is proved using a variant of Theorem 4.2 with $E_{\text{out}} = \emptyset$. The proof of Theorem 4.4 is almost the same as that of Theorem 4.2, and so in what follows we only highlight the difference.

Let $G = (V, E)$ and $m = |E|$. Similarly, we assume the maximum degree is $m/\left(40n^{1/s}\log n\right)$, since otherwise we are in the easy case, where we can apply Theorem 4.1 to have one vertex v learn the entire edge set E in $O(n^{1/s+o(1)}) \leq O(n^{(s-2)/s+o(1)})$ rounds, and we are done after that.

We partition V into $n^{1/s}$ subsets $V_1, \ldots, V_{n^{1/s}}$. Instead of considering triads, here we consider s-tuples: $\left\{(i_1, \ldots, i_s) \mid 1 \leq i_1 \leq \ldots \leq i_s \leq n^{1/s}\right\}$. After a vertex v learns the edge set $\bigcup_{j_1, j_2 \in [1,s]} E(V_{i_{j_1}}, V_{i_{j_2}})$, it has ability to list all s-vertex subgraphs in which the jth vertex is in V_{i_j}. We prove a variant of Lemma 4.3, as follows.

LEMMA 4.6. *W.h.p.,* $|E(V_i, V_j)| = O(m/n^{2/s})$ *for all* $i, j \in [1, n^{1/s}]$.

Proof. We set $p = 2n^{-1/s}$. The maximum degree is at most $m/\left(40n^{1/s}\log n\right) \leq mp/20\log n$, and $p^2 m \geq$

$400 \log^2 n$. By applying Lemma 4.2 and use the same analysis in Lemma 4.3, we conclude this lemma. □

Proof. [Proof of Theorem 4.4] Here we only consider the time complexity to deliver all messages. Consider a vertex v. If $k_v < 1/2$, then v receives no message. Otherwise v is responsible for between $2k_v$ and $4k_v$ s-tuples, and v collects $E(V_i, V_j)$ for at most $4s^2k_v$ pairs (V_i, V_j). By Lemma 4.6, w.h.p., $|E(V_i, V_j)| = O(m/n^{2/s})$ for all i, j. Hence the number of edges v received is at most $O(m/n^{2/s}) \cdot 4s^2k_v = O(\deg(v) \cdot n^{(s-2)/s})$.

Note that each vertex v sends at most $O(\deg(v)n^{(s-2)/s})$ messages since for each incident edge e of v, there are at most $O(n^{(s-2)/s})$ s-tuples involving e. By Theorem 4.1, the delivery of all messages can be done in $O(n^{(s-2)/s+o(1)})$ rounds, w.h.p. □

5 Conclusion

In this paper we have shown that all variants of Triangle Detection, Enumeration, and Counting can be solved in $\tilde{O}(n^{1/2})$ rounds in the CONGEST model. The bottleneck in our algorithm is not triangle-finding *per se*, but in the decomposition of the graph into expanding subgraphs. Our graph decomposition routine takes $\tilde{O}(n^{1/2})$ time and produces three edge sets, the third one inducing a subgraph with arboricity $O(n^{1/2})$. We believe that this third set is unnecessary, and that the running time can be improved substantially.

HYPOTHESIS 1. *In the* CONGEST *model, the edge set E can be partitioned into $E = E_m \cup E_r$ such that $|E_r| \leq |E|/6$ and connected components induced by E_m have conductance $\Omega(1/\text{polylog}(n))$ and hence $O(\text{polylog}(n))$ mixing time. The time required to compute (E_m, E_r) is at most or $n^{o(1)}$ time* (weak version)*, or* $\text{polylog}(n)$ *time* (strong version).

Observe that in the triangle enumeration application, an even weaker version of Hypothesis 1 is sufficient to improve our algorithm's running time to $n^{1/3+o(1)}$. It is enough to find a decomposition in $n^{1/3+o(1)}$ time achieving $n^{o(1)}$ mixing time in E_m. However, we suspect that the *weak* and *strong* variants of Hypothesis 1 will become more meaningful as this decomposition technique finds more applications in CONGEST.

Assuming Hypothesis 1 (either variant), the upper bound for triangle enumeration, detection, and counting as a function of n and Δ is:

$$\min\left\{ O(\Delta),\ n^{1/3+o(1)} \right\},$$

i.e., depending on the magnitude of Δ, we should execute one of two algorithms. Is there a *third* algorithm that is substantially better than these two for some triangle problem and some graph density?

References

[1] A. ABBOUD, K. CENSOR-HILLEL, S. KHOURY, AND C. LENZEN, *Fooling views: A new lower bound technique for distributed computations under congestion*, arXiv preprint arXiv:1711.01623, (2017).

[2] U. AGARWAL, V. RAMACHANDRAN, V. KING, AND M. PONTECORVI, *A deterministic distributed algorithm for exact weighted all-pairs shortest paths in $\tilde{O}(n^{3/2})$ rounds*, in Proceedings 38th ACM Symposium on Principles of Distributed Computing (PODC), 2018, pp. 199–205.

[3] R. ANDERSEN, F. R. K. CHUNG, AND K. J. LANG, *Local partitioning for directed graphs using PageRank*, Internet Mathematics, 5 (2008), pp. 3–22.

[4] R. ANDERSEN, S. O. GHARAN, Y. PERES, AND L. TREVISAN, *Almost optimal local graph clustering using evolving sets*, J. ACM, 63 (2016), pp. 15:1–15:31.

[5] S. ARORA, B. BARAK, AND D. STEURER, *Subexponential algorithms for unique games and related problems*, J. ACM, 62 (2015), pp. 42:1–42:25.

[6] K. CENSOR-HILLEL, P. KASKI, J. H. KORHONEN, C. LENZEN, A. PAZ, AND J. SUOMELA, *Algebraic methods in the congested clique*, Distributed Computing, (2016).

[7] A. CZUMAJ AND C. KONRAD, *Detecting Cliques in CONGEST Networks*, in Proceedings 32nd International Symposium on Distributed Computing (DISC), U. Schmid and J. Widder, eds., vol. 121 of Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2018, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 16:1–16:15.

[8] A. DAS SARMA, S. GOLLAPUDI, AND R. PANIGRAHY, *Sparse cut projections in graph streams*, in Proceedings 17th European Symposium on Algorithms (ESA), 2009, pp. 480–491.

[9] D. DOLEV, C. LENZEN, AND S. PELED, *"Tri, tri again": Finding triangles and small subgraphs in a distributed setting*, in Proceedings 26th International Symposium on Distributed Computing (DISC), 2012, pp. 195–209.

[10] A. DRUCKER, F. KUHN, AND R. OSHMAN, *On the power of the congested clique model*, in Proceedings 33rd ACM Symposium on Principles of Distributed Computing (PODC), 2014, pp. 367–376.

[11] M. ELKIN, *Distributed exact shortest paths in sublinear time*, in Proceedings 49th Annual ACM Symposium on Theory of Computing (STOC), 2017, pp. 757–770.

[12] M. ELKIN, *A simple deterministic distributed MST algorithm, with near-optimal time and message complexities*, in Proceedings 37th ACM Symposium on Principles of Distributed Computing (PODC), 2017, pp. 157–163.

[13] G. Even, O. Fischer, P. Fraigniaud, T. Gonen, R. Levi, M. Medina, P. Montealegre, D. Olivetti, R. Oshman, I. Rapaport, and I. Todinca, *Three Notes on Distributed Property Testing*, in Proceedings 31st International Symposium on Distributed Computing (DISC), vol. 91 of Leibniz International Proceedings in Informatics (LIPIcs), 2017, pp. 15:1–15:30.

[14] O. Fischer, T. Gonen, F. Kuhn, and R. Oshman, *Possibilities and impossibilities for distributed subgraph detection*, in Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures (SPAA), New York, NY, USA, 2018, ACM, pp. 153–162.

[15] M. Ghaffari and B. Haeupler, *Distributed algorithms for planar networks I: Planar embedding*, in Proceedings 36th ACM Symposium on Principles of Distributed Computing (PODC), 2016, pp. 29–38.

[16] M. Ghaffari and B. Haeupler, *Distributed algorithms for planar networks II: Low-congestion shortcuts, MST, and min-cut*, in Proceedings 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2016, pp. 202–219.

[17] M. Ghaffari, F. Kuhn, and H.-H. Su, *Distributed MST and routing in almost mixing time*, in Proceedings 37th ACM Symposium on Principles of Distributed Computing (PODC), 2017, pp. 131–140.

[18] M. Ghaffari and J. Li, *New distributed algorithms in almost mixing time via transformations from parallel algorithms*, in Proceedings 32nd International Symposium on Distributed Computing (DISC), U. Schmid and J. Widder, eds., vol. 121 of Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2018, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 31:1–31:16.

[19] O. Goldreich and D. Ron, *A sublinear bipartiteness tester for bounded degree graphs*, Combinatorica, 19 (1999), pp. 335–373.

[20] T. Gonen and R. Oshman, *Lower bounds for subgraph detection in the CONGEST model*, in Proceedings 21st International Conference on Principles of Distributed Systems (OPODIS), vol. 95 of Leibniz International Proceedings in Informatics (LIPIcs), 2018, pp. 6:1–6:16.

[21] C. Huang, D. Nanongkai, and T. Saranurak, *Distributed exact weighted all-pairs shortest paths in $\tilde{O}(n^{5/4})$ rounds*, in Proceedings 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2017, pp. 168–179.

[22] T. Izumi and F. Le Gall, *Triangle finding and listing in CONGEST networks*, in Proceedings 37th ACM Symposium on Principles of Distributed Computing (PODC), 2017, pp. 381–389.

[23] M. Jerrum and A. Sinclair, *Approximating the permanent*, SIAM Journal on Computing, 18 (1989), pp. 1149–1178.

[24] T. Jurdziński and K. Nowicki, *MST in $O(1)$ rounds of congested clique*, in Proceedings 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2018, pp. 2620–2632.

[25] K.-i. Kawarabayashi and M. Thorup, *Deterministic global minimum cut of a simple graph in near-linear time*, in Proceedings 47th Annual ACM Symposium on Theory of Computing (STOC), 2015, pp. 665–674.

[26] J. H. Korhonen and J. Rybicki, *Deterministic subgraph detection in broadcast CONGEST*, in Proceedings 21st International Conference on Principles of Distributed Systems (OPODIS), vol. 95 of Leibniz International Proceedings in Informatics (LIPIcs), 2018, pp. 4:1–4:16.

[27] S. Krinninger and D. Nanongkai, *A faster distributed single-source shortest paths algorithm*, in Proceedings 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2018, pp. 686–697.

[28] F. Kuhn and A. R. Molla, *Distributed sparse cut approximation*, in Proceedings 19th International Conference on Principles of Distributed Systems (OPODIS), 2015, pp. 10:1–10:14.

[29] A. Kumar, C. Seshadhri, and A. Stolman, *Finding forbidden minors in sublinear time: a $O(n^{1/2+o(1)})$-query one-sided tester for minor closed properties on bounded degree graphs*, in Proceedings 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2018, pp. 509–520.

[30] T. C. Kwok and L. C. Lau, *Finding small sparse cuts by random walk*, in Proceedings 15th International Workshop on Approximation, Randomization, and Combinatorial Optimization (APPROX/RANDOM), 2012, pp. 615–626.

[31] C. Lenzen, *Optimal deterministic routing and sorting on the congested clique*, in Proceedings 33rd ACM Symposium on Principles of Distributed Computing (PODC), 2013, pp. 42–50.

[32] G. Moshkovitz and A. Shapira, *Decomposing a graph into expanding subgraphs*, Random Struct. Algorithms, 52 (2018), pp. 158–178.

[33] D. Nanongkai, T. Saranurak, and C. Wulff-Nilsen, *Dynamic minimum spanning forest with subpolynomial worst-case update time*, in Proceedings of IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS), IEEE, 2017, pp. 950–961.

[34] G. Pandurangan, P. Robinson, and M. Scquizzato, *On the distributed complexity of large-scale graph computations*, in Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures (SPAA), New York, NY, USA, 2018, ACM, pp. 405–414.

[35] M. Pătraşcu and M. Thorup, *Planning for fast connectivity updates*, in Proceedings 48th IEEE Symposium on Foundations of Computer Science (FOCS), 2007, pp. 263–271.

[36] D. Peleg, *Distributed Computing: A Locality-Sensitive Approach*, SIAM, 2000.

[37] D. Peleg and V. Rubinovich, *A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction*, SIAM J. Comput., 30 (2000), pp. 1427–1442.

[38] P. Raghavendra and D. Steurer, *Graph expansion and the unique games conjecture*, in Proceedings 42nd ACM Symposium on Theory of Computing (STOC), 2010, pp. 755–764.
[39] A. D. Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer, *Distributed verification and hardness of distributed approximation*, SIAM J. Comput., 41 (2012), pp. 1235–1265.
[40] A. D. Sarma, A. R. Molla, and G. Pandurangan, *Distributed computation of sparse cuts via random walks*, in Proceedings 16th International Conference on Distributed Computing and Networking (ICDCN), 2015, pp. 6:1–6:10.
[41] D. A. Spielman and S.-H. Teng, *Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems*, in Proceedings 36th Annual ACM Symposium on Theory of Computing (STOC), 2004, pp. 81–90.
[42] D. A. Spielman and S.-H. Teng, *A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning*, SIAM J. Comput., 42 (2013), pp. 1–26.
[43] L. Trevisan, *Approximation algorithms for unique games*, Theory of Computing, 4 (2008), pp. 111–128.