Near-optimal Distributed Triangle Enumeration via Expander Decompositions

YI-JUN CHANG, ETH Zürich, Switzerland
SETH PETTIE, University of Michigan, USA
THATCHAPHOL SARANURAK, Toyota Technological Institute at Chicago, USA
HENGJIE ZHANG, Columbia University, USA

We present improved distributed algorithms for variants of the triangle finding problem in the CONGEST model. We show that triangle detection, counting, and enumeration can be solved in $\tilde{O}(n^{1/3})$ rounds using *expander decompositions*. This matches the triangle enumeration lower bound of $\tilde{\Omega}(n^{1/3})$ by Izumi and Le Gall [PODC'17] and Pandurangan, Robinson, and Scquizzato [SPAA'18], which holds even in the CONGESTED-CLIQUE model. The previous upper bounds for triangle detection and enumeration in CONGEST were $\tilde{O}(n^{2/3})$ and $\tilde{O}(n^{3/4})$, respectively, due to Izumi and Le Gall [PODC'17].

An (ϵ, ϕ)-expander decomposition of a graph $G = (V, E)$ is a clustering of the vertices $V = V_1 \cup \cdots \cup V_x$ such that (i) each cluster V_i induces a subgraph with conductance at least ϕ and (ii) the number of inter-cluster edges is at most $\epsilon|E|$. We show that an (ϵ, ϕ)-expander decomposition with $\phi = (\epsilon/\log n)^{2^{O(k)}}$ can be constructed in $O(n^{2/k} \cdot \text{poly}(1/\phi, \log n))$ rounds for any $\epsilon \in (0, 1)$ and positive integer k. For example, a $(1/n^{o(1)}, 1/n^{o(1)})$-expander decomposition only requires $n^{o(1)}$ rounds to compute, which is optimal up to subpolynomial factors, and a $(0.1, 1/\text{poly}\log n)$-expander decomposition can be computed in $O(n^{\gamma})$ rounds, for any arbitrarily small constant $\gamma > 0$.

Our triangle finding algorithms are based on the following generic framework using expander decompositions, which is of independent interest. We first construct an expander decomposition. For each cluster, we simulate CONGESTED-CLIQUE algorithms with small overhead by applying the *expander routing* algorithm due to Ghaffari, Kuhn, and Su [PODC'17] Finally, we deal with inter-cluster edges using recursive calls.

CCS Concepts: • **Theory of computation → Distributed algorithms**;

Additional Key Words and Phrases: CONGEST, distributed graph algorithms

This article synthesizes results from two conference papers: *Distributed Triangle Detection via Expander Decomposition*, in Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms (SODA 2019), San Diego, CA, USA [11] and *Improved Distributed Expander Decomposition and Nearly Optimal Triangle Enumeration*, in Proceedings of the 38th ACM Symposium on Principles of Distributed Computing (PODC 2019), Toronto, ON, Canada [12].

Supported by Dr. Max Rössler, by the Walter Haefner Foundation, and by the ETH Zürich Foundation.
Supported by NSF grants CCF-1514383, CCF-1637546, and CCF-1815316.

Authors' addresses: Y.-J. Chang, ETH Zürich, Institute for Theoretical Studies, Clausiusstrasse 47, 8092 Zürich, Switzerland; email: yi-jun.chang@eth-its.ethz.ch; S. Pettie, University of Michigan, Computer Science and Engineering, 2260 Hayward Street, Ann Arbor, MI 48109, USA; email: pettie@umich.edu; T. Saranurak, University of Michigan, Computer Science and Engineering, 2260 Hayward Street, Ann Arbor, MI 48109, USA; email: thsa@umich.edu; H. Zhang, Columbia University, Department of Computer Science, 500 W 120th St, New York, NY 10027, USA; email: hengjie.z@columbia.edu.

0004-5411/2021/05-ART21 $15.00
https://doi.org/10.1145/3446330

ACM Reference format:
Yi-jun Chang, Seth Pettie, Thatchaphol Saranurak, and Hengjie Zhang. 2021. Near-optimal Distributed Triangle Enumeration via Expander Decompositions. *J. ACM* 68, 3, Article 21 (May 2021), 36 pages.
https://doi.org/10.1145/3446330

1 INTRODUCTION

We consider triangle finding problems in distributed networks. We focus on the CONGEST model of distributed computing, where the underlying distributed network is represented as an undirected graph $G = (V, E)$. Each vertex corresponds to a computational device and each edge corresponds to a bi-directional communication link. It is common in the literature to assume that each vertex $v \in V$ initially knows some global parameters such as the number of vertices $n = |V|$, the maximum degree $\Delta = \max_{v \in V} \deg(v)$, and the diameter $D = \text{diameter}(G)$. In this article, 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 article, we only consider the randomized variant of CONGEST, where each vertex is allowed to generate unlimited local random bits, but there is no global randomness.

Many variants of the triangle finding problem have been studied in the literature [1, 9, 11, 17, 18, 24, 32, 52].

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.

All of these problems can be solved in exactly *one* round of communication if there is no no limit on bandwidth: every vertex v simply announces its neighborhood $N(v)$ to all neighbors. It is the bandwidth constraint of CONGEST that makes these problems non-trivial.

Whereas many graph optimization problems studied in the CONGEST model are intrinsically "global" (i.e., require at least diameter rounds) [2, 21, 22, 25, 26, 29, 38], the triangle finding problem is somewhat unusual in that it can, in principle, be solved using only locally available information.

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 CONGESTED-CLIQUE [34, 49], but its round complexity is $\tilde{\Theta}(D + \sqrt{n})$ in CONGEST [20, 41, 54, 58].[1]

[1]The notations $\tilde{O}(\cdot)$, $\tilde{\Theta}(\cdot)$, and $\tilde{\Omega}(\cdot)$ hide any factor polylogarithmic in n.

One of the main reasons that some problems can be solved efficiently in CONGESTED-CLIQUE is the routing algorithm of Lenzen [42]. 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 [42] as a communication primitive, many parallel algorithms can be transformed into efficient CONGESTED-CLIQUE algorithms [9]. For example, consider the distributed matrix multiplication problem, where the input matrices are distributed to the vertices such that the ith vertex initially knows the ith row. The problem can be solved in CONGESTED-CLIQUE in $\tilde{O}(n^{1/3})$ rounds over semirings, or $n^{1-(2/\omega)+o(1)} = o(n^{0.158})$ rounds over rings [9], where $\omega < 2.373$ is the exponent for matrix multiplication. As a consequence, Triangle Detection can be solved in $n^{1-(2/\omega)+o(1)} = o(n^{0.158})$ rounds in CONGESTED-CLIQUE [9].

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.

Definition 1 (Mixing Time [27]). 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 [27] proved that if each vertex v is the source and the destination of at most $\deg(v)$ 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 in CONGEST. The $2^{O(\sqrt{\log n \log\log n})}$ factor has recently been improved by Ghaffari and Li [28] to $2^{O(\sqrt{\log n})}$. This implies 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 [28], bypassing the $\tilde{\Omega}(\sqrt{n})$ lower bound for general graphs [54].

At this point, a natural question to ask is whether or not this line of research [27, 28] can be extended to a broader class of graphs *without* any conductance guarantee (e.g., the general graphs). The main contribution of this article is to show that this is in fact doable through the use of *distributed expander decompositions.*

Informally, an expander decomposition removes a small fraction of the edges so that the remaining connected components have high conductance. In this article, we present the first $n^{o(1)}$-round distributed expander decomposition algorithm in CONGEST. Based on this tool, we show that the Triangle Enumeration problem can be solved in near-optimal round complexity $\tilde{O}(n^{1/3})$, as follows. We first construct an expander decomposition. For each high conductance component of the decomposition, we simulate a modified version of the CONGESTED-CLIQUE Triangle Enumeration algorithm of Dolev et al. [17] with small overhead by applying the routing algorithm due to Ghaffari, Kuhn, and Su [27], and then we deal with triangles consisting solely of inter-cluster edges using recursion.

1.1 Distributed Triangle Finding

In this article, we show that Triangle Detection, Enumeration, and Counting can be solved in $\tilde{O}(n^{1/3})$ rounds in CONGEST, matching the $\Omega(n^{1/3}/\log n)$ lower bound [32, 52] up to a polylogarithmic factor. To our knowledge, this is the *only* non-trivial problem whose CONGEST and CONGESTED-CLIQUE complexities coincide, up to $\tilde{O}(1)$ factors.

Theorem 1. *Triangle Detection, Enumeration, and Counting can be solved in $\tilde{O}(n^{1/3})$ rounds in* CONGEST*, w.h.p.*

This result is achieved by a combination of several techniques, including our new distributed expander decomposition algorithm, a variant of the multi-commodity routing scheme of References [27, 28], and an adaptation of the CONGESTED-CLIQUE Triangle Enumeration algorithm of Dolev et al. [17] to CONGEST networks with small mixing time.

Prior Work. Dolev, Lenzen, and Peled [17, Remark 1] showed that Triangle Enumeration can be solved deterministically in $O(n^{1/3}/\log n)$ rounds in CONGESTED-CLIQUE. Censor-Hillel et al. [9] presented an algorithm for Triangle Detection and Counting in CONGESTED-CLIQUE that takes $n^{1-(2/\omega)+o(1)} = o(n^{0.158})$ rounds via a reduction to matrix multiplication. Izumi and Le Gall [32] showed that in CONGEST, the Detection and Enumeration problems can be solved in $\tilde{O}(n^{2/3})$ and $\tilde{O}(n^{3/4})$ rounds, respectively. They also proved that in both CONGEST and CONGESTED-CLIQUE, the Enumeration problem requires $\Omega(n^{1/3}/\log n)$ rounds, improving an earlier $\Omega(n^{1/3}/\log^3 n)$ bound of Pandurangan et al. [52].

Izumi and Le Gall [32] proved a large separation between the complexity of the Enumeration and Local Enumeration problems. If triangles must be reported by a participating vertex, then $\Omega(n/\log n)$ 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. However, very little is known about the complexity of triangle *detection.* Abboud et al. [1] proved that deterministic *1-round* triangle detection algorithms must transmit $\Omega(\Delta \log n)$ bits, whereas Fischer et al. [24] proved that randomized 1-round algorithms must transmit $\Omega(\Delta)$ bits. Neither result precludes a 2-round algorithm transmitting $\tilde{O}(1)$ bits, independent of Δ.

1.2 Distributed Expander Decompositions

Before we proceed, we review some graph terminology. For each vertex v, let $N(v)$ denote the set of neighbors of v. We also write $N^k(v) = \{u \in V \mid \operatorname{dist}(u,v) \le k\}$. Note that $N^1(v) = N(v) \cup \{v\}$. The terms $\operatorname{dist}(u,v)$, $N(v)$, and $N^k(v)$ depend on the underlying graph G, which appears subscripted if not clear from context. Throughout the article, $m = |E|$ denotes the number of edges in the *original* communication network $G = (V,E)$.

Conductance. Consider a graph $G = (V,E)$. For a vertex subset S, we write $\operatorname{Vol}(S)$ to denote $\sum_{v \in S} \deg(v)$. 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 *conductance* of a cut $(S,\bar{S})$ is defined as $\Phi(S) = |\partial(S)|/\min\{\operatorname{Vol}(S),\operatorname{Vol}(\bar{S})\}$. For the special case of $S = \emptyset$ and $S = V$, we set $\Phi(S) = 0$. A cut $(S,\bar{S})$ of conductance ϕ is also called a *ϕ-sparse cut.* The *conductance* Φ_G of a graph G is the minimum value of $\Phi(S)$ over all vertex subsets S with $0 < |S| < |V|$. We have the following relation [33] between the mixing time $\tau_{\text{mix}}(G)$ and conductance Φ_G:

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

Balance. Define the *balance* $\operatorname{bal}(S)$ of a cut S by $\operatorname{bal}(S) = \min\{\operatorname{Vol}(S),\operatorname{Vol}(\bar{S})\}/\operatorname{Vol}(V)$. We say that S is a *most-balanced* cut of G of conductance at most ϕ if $\operatorname{bal}(S)$ is maximized among all cuts of G with conductance at most ϕ.

Subgraph Notation. Let S be a vertex set. Denote by $E(S)$ the set of all edges with both endpoints in S. We write $G[S]$ to denote the subgraph induced by S, and we write $G\{S\}$ to denote the graph resulting from adding $\deg_V(v) - \deg_S(v)$ self-loops to each vertex v in $G[S]$. As in Reference [61], self-loops are regarded as contributing one to the degree, not two, and hence every $v \in S$ has the same degree in both G and $G\{S\}$. Observe that we always have

$$\Phi(G\{S\}) \leq \Phi(G[S]).$$

Expander Decompositions. An (ϵ, ϕ)-*expander decomposition* of a graph $G = (V, E)$ is defined as a partition of the vertex set $V = V_1 \cup \cdots \cup V_x$ satisfying the following conditions:

- For each cluster V_i, we have $\Phi(G\{V_i\}) \geq \phi$.
- The number of inter-cluster edges $(|\partial(V_1)| + \cdots + |\partial(V_x)|)/2$ is at most $\epsilon|E|$.

It is well known that *any* graph can be decomposed into connected components of conductance $\Omega(\epsilon/\log n)$ after removing at most an ϵ-fraction of the edges [4, 35, 53, 62, 66], and this bound is *tight*: after removing any constant fraction of the edges in a hypercube, some remaining component must have conductance at most $O(1/\log n)$ [3].

The expander decomposition has a wide range of applications, and it has been applied to solving linear systems [65], unique games [4, 55, 66], minimum cut [36], and dynamic algorithms [48].

A major contribution of this article is the following result.

Theorem 2. *Let $\epsilon \in (0, 1)$ and k be a positive integer. An (ϵ, ϕ)-expander decomposition with $\phi = (\frac{\log n}{\epsilon})^{2^{O(k)}}$ can be constructed in $n^{2/k} \cdot (\frac{\log n}{\epsilon})^{2^{O(k)}} = n^{2/k} \cdot \text{poly}(\frac{1}{\phi}, \log n)$ rounds, w.h.p.*

We emphasize that the number of rounds does not depend on the diameter of G. There is a trade-off between the two parameters ϵ and ϕ. For example, an (ϵ, ϕ)-expander decomposition with $\epsilon = 2^{-\log^{1/3} n}$ and $\phi = 2^{-\log^{2/3} n}$ can be constructed in $n^{O(1/\log\log n)}$ rounds by setting $k = O(\log\log n)$ in Theorem 2. If we are allowed to have $\epsilon = 0.1$ and spend $O(n^{0.1})$ rounds, then we can achieve $\phi = \Omega(1/\text{poly}\log n)$.

Prior Work. In the centralized setting, the first polynomial time algorithm for constructing an (ϵ, ϕ)-expander decomposition is given by Kannan, Vempala, and Vetta [35], where $\epsilon = \tilde{O}(\phi)$, and was further studied in many other subsequent works [5, 50, 51, 53, 57, 64, 66].

Spielman and Teng [63, 64] improved the runtime to $\tilde{O}(m/\text{poly}(\phi))$, for a weaker variant of the expander decompositions with $\epsilon = 1/\text{poly}(\phi, \log n)$. In their variant, each part V_i may not induce an expander, but it is guaranteed to be *contained* in some *unknown* expander in the sense that there is some $W_i \supseteq V_i$ for which $\Phi_{G\{W_i\}} \geq \phi$. Although this guarantee suffices for some applications [15, 37], others [14, 48] crucially require that each part of the decomposition induces an expander.

Nanongkai and Saranurak [47] and Wulff-Nilsen [67] independently gave a fast decomposition algorithm without weakening any guarantees, as in References [63, 64]. The algorithm of Reference [47] finds a $(\phi \log^{O(k)} n, \phi)$-expander decomposition in time $\tilde{O}(m^{1+1/k})$. Although the trade-off is worse in Reference [67], their high-level approaches are in fact the same. They gave the same black-box reduction from constructing an expander decomposition to finding a nearly most balanced sparse cut. Our distributed algorithm for Theorem 2 also follows this high-level approach.

Most recently, Saranurak and Wang [57] gave a $(\tilde{O}(\phi), \phi)$-expander decomposition algorithm with running time $\tilde{O}(m/\phi)$. This is optimal up to a polylogarithmic factor when $\phi \geq 1/\text{poly}\log(n)$. We do not use their approach, as their *trimming step* appears to be inherently sequential and challenging to parallelize or make distributed.

2 DISTRIBUTED TRIANGLE ENUMERATION

The goal of this section is to prove Theorem 1 using Theorem 2. The proof of Theorem 1 is in Section 2.2, and it uses Theorem 4.

Theorem 1. *Triangle Detection, Enumeration, and Counting can be solved in $\tilde{O}(n^{1/3})$ rounds in* CONGEST, *w.h.p.*

In Section 2.1, we review the routing algorithm of Ghaffari, Kuhn, and Su [27] and describe the adjustments that we need. In Section 2.2, we show how to adapt the randomized variant of the Triangle Enumeration CONGESTED-CLIQUE algorithm of Dolev, Lenzen, and Peled [17] to high conductance graphs in CONGEST. Combined with our distributed expander decomposition algorithm, this proves Theorem 1. In Section 2.3, we extend the result of Section 2.2 to enumeration of general subgraphs.

2.1 Routing in High Conductance Graphs

Consider the following multi-commodity routing problem. Given a set of routing requests where each vertex v is a source or a destination for at most $\deg(v)$ messages of $O(\log n)$ bits, the goal is to deliver all messages to their destinations. Ghaffari, Khun, and Su [27] showed that this routing problem can be solved in $2^{O(\sqrt{\log n \log\log n})} \cdot O(\tau_{\text{mix}})$ rounds. This was later improved to $2^{O(\sqrt{\log n})} \cdot O(\tau_{\text{mix}})$ by Ghaffari and Li [28].

Theorem 3 ([27, 28]). *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* $\deg(v)$ *messages, then there is a randomized distributed algorithm that delivers all messages in $\tau_{\text{mix}}(G) \cdot 2^{O(\sqrt{\log n})}$ rounds, w.h.p., in the* CONGEST *model.*

Whether the $2^{O(\sqrt{\log n})}$ factor of Theorem 3 can be improved is an intriguing open question. A straightforward generalization of Theorem 3 shows that when v is the source/destination of at most $M \cdot \deg(v)$ messages, all can be delivered in $M \cdot \tau_{\text{mix}}(G) \cdot 2^{O(\sqrt{\log n})}$ rounds.

Here, we observe that when M is polynomial in n, it *is* actually possible to reduce the $n^{o(1)}$ overhead to $\tilde{O}(1)$ by modifying the Ghaffari-Kuhn-Su [27] algorithm. This below discussion does not seem to apply to the Ghaffari-Li algorithm [28].

The routing algorithm of Reference [27] has two parts: a *pre-processing* routine that builds a hierarchical routing structure, and a *routing* routine that routes an ensemble of messages. The overall round complexity of $2^{O(\sqrt{\log n \log\log n})} \cdot O(\tau_{\text{mix}})$ is a result of balancing these two parts when $M = O(1)$, as follows.

Parameters: The parameter k is a positive integer that specifies the depth of the hierarchical routing structure. Define $\beta = m^{1/k}$, where m is the total number of edges.

Pre-processing Time: The algorithm for building the hierarchical routing structure consists of two parts, having complexities $k\beta \cdot (\log n)^{O(k)} \cdot \tau_{\text{mix}}$ [27, Lemma 3.2] and $O(k\beta^2 \log n \cdot \tau_{\text{mix}})$ [27, Lemma 3.3], respectively.

Routing Time: After building the hierarchical routing structure, a routing task can be solved in $(\log n)^{O(k)} \cdot \tau_{\text{mix}}$ rounds [27, Lemma 3.4]. More generally, it takes $M \cdot (\log n)^{O(k)} \cdot \tau_{\text{mix}}$ rounds if each vertex v is a source or a destination of at most $M \cdot \deg(v)$ messages.

The parameter k can be chosen as any positive integer. In Reference [27] they used $k = \Theta(\sqrt{\log n / \log\log n})$ to balance the pre-processing time and the routing time to show that the routing task can be solved in $2^{O(\sqrt{\log n \log\log n})} \cdot O(\tau_{\text{mix}})$ rounds. This round complexity was later improved to $2^{O(\sqrt{\log n})} \cdot O(\tau_{\text{mix}})$ in Reference [28], but unlike Reference [27], the algorithm of Reference [28] does not admit a trade-off as above, due to their special treatment of the base layer G_0 of the hierarchical routing structure. In Reference [28], G_0 is a random graph with degree $2^{O(\sqrt{\log n})}$, and simulating one round in G_0 already costs $2^{O(\sqrt{\log n})} \cdot O(\tau_{\text{mix}})$ rounds in the original graph G.

In our Triangle Enumeration application the parameter $M = \Theta(n^{1/3})$ is polynomial in n, which means it is advantageous to set $k = 12$ and $\beta = m^{1/12} < n^{1/6}$. Under this parameterization both the pre-processing and routing stages take $\tilde{O}(n^{1/3} \cdot \tau_{\text{mix}})$ rounds. This will lead to a Triangle Enumeration algorithm taking $\tilde{O}(n^{1/3})$ rounds rather than $n^{1/3+o(1)}$ rounds.

Remark 1. The claim of Theorem 3 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 References [27, 28] 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 $(\text{ID}(v), 1), \ldots, (\text{ID}(v), \deg(v))$. It is proven [27, 28] that effecting a set of routing requests in G_0 takes $2^{O(\sqrt{\log n})}$ rounds in G_0; however, to translate a routing request $\text{ID}(x) \rightsquigarrow \text{ID}(y)$ in G to G_0, it seems necessary to map it (probabilistically) to $(\text{ID}(x), i) \rightsquigarrow (\text{ID}(y), j)$, 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 vertices 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 $\text{ID}(y)$. Arbitrary ID-assignments obviously do not betray this information.

Lemma 1. *In $O(D + \log n)$ rounds we can compute an* ID*-assignment $V \to \{1, \ldots, |V|\}$ and other information such that* $\text{ID}(u) < \text{ID}(v)$ *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 $O(D)$ rounds. 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 $i = 0, \ldots, \lfloor \log(n-1) \rfloor$. This takes $O(D + \log n)$ rounds by pipelining. At this point the root x has the counts $n_0, \ldots, n_{\lfloor \log(n-1) \rfloor}$ 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 $n_0, \ldots, n_{\lfloor \log(n-1) \rfloor}$, and disseminates the IDs to all vertices 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 $O(D + \log n)$ rounds. Clearly knowing $n_0, \ldots, n_{\lfloor \log(n-1) \rfloor}$ and $\text{ID}(v)$ suffice to calculate $\lfloor \log \deg(v) \rfloor$. □

Lemma 1 gives us a well-behaved ID-assignment to apply the routing algorithm of Reference [27]. 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.

Theorem 1 is proved by running the Triangle Enumeration/Counting algorithm of Theorem 4 on the clusters of $O(\text{poly} \log n)$ mixing time in an expander decomposition. As the diameter D of a graph is at most the mixing time of the graph, the term $O(D)$ in the complexity of of Lemma 1 does not appear in the complexity of Theorems 1 and 4.

2.2 Triangle Enumeration

In this section, we prove Theorem 4. Because of the way we add self-loops in the definition of $G^* = G\{S\}$, we have $\deg_{G^*}(v) = \deg_G(v)$ for every $v \in S$ in Theorem 4. In particular, the total number of edges in G^* is equal to the number of edges in G with at least one endpoint in S. For the special case of $S = V$, we have $G^* = G$.

Theorem 4. *Let $G^* = G\{S\}$ for some vertex subset S. In the* CONGEST *model, all triangles in G having at least two vertices in S can be counted and enumerated, w.h.p., in $\tau_{\text{mix}}(G^*) \cdot \tilde{O}(n^{1/3})$ rounds. The algorithm only sends messages along the edges incident to S in G.*

We first show how to prove Theorem 1 by combining our distributed expander decomposition algorithm with Theorem 4.

Proof of Theorem 1. Applying our distributed expander decomposition algorithm (Theorem 2), we construct an (ϵ, ϕ)-expander decomposition $V = V_1 \cup \cdots \cup V_x$ with $\epsilon = 1/2$ and $\phi = 1/\Theta(\text{poly} \log n)$ in $\tilde{O}(n^{1/3})$ rounds by selecting $k = 6$. The mixing time τ_{mix} of each cluster $G\{V_i\}$ is at most $O(\frac{\log n}{\phi^2}) = O(\text{poly} \log n)$. We then apply the Triangle Enumeration/Counting algorithm of Theorem 4 to each cluster $G\{V_i\}$, in parallel, taking $\tilde{O}(n^{1/3}) \cdot O(\tau_{\text{mix}}) = \tilde{O}(n^{1/3})$ rounds. At this point all triangles have been enumerated, except for those contained entirely in E', the set of inter-cluster edges. We enumerate the remaining triangles by calling the algorithm recursively on E'. Since $|E'| \le \epsilon m = m/2$, the total complexity of all recursive calls is $\tilde{O}(n^{1/3}) \cdot \log m = \tilde{O}(n^{1/3})$. □

Overview. The remainder of this section constitutes a proof of Theorem 4. We begin with an overview of the Triangle Enumeration algorithm of Dolev, Lenzen, and Peled [17] for the CONGESTED-CLIQUE. Partition the vertex set V into $n^{1/3}$ parts of equal size $V_1 \cup \cdots \cup V_{n^{1/3}}$. For each triangle, there exist three indices $1 \le i \le j \le k \le n^{1/3}$ such that the triangle belongs to the subgraph induced by $V_i \cup V_j \cup V_k$. Since there are less than n such triples (i, j, k), we can associate each triple (i, j, k) with a vertex v, and such a vertex v will be responsible for listing all triangles in $V_i \cup V_j \cup V_k$. The number of edges in this subgraph induced by $V_i \cup V_j \cup V_k$ is at most $\binom{|V_i \cup V_j \cup V_k|}{2} = O(n^{4/3})$. With Lenzen's routing algorithm [42], it takes only $O(n^{1/3})$ rounds for each vertex to learn all the information it needs.

To adapt this algorithm to high conductance CONGEST networks, we have to replace Lenzen's routing algorithm [42] with the routing algorithm of Ghaffari, Kuhn, and Su [27], which has a more stringent requirement that the number of messages sent from/received by v is $O(\deg(v))$. For comparison, Lenzen's routing algorithm allows each vertex to be a source and a destination for $O(n)$ messages.

Other than one easy case, we show that if we partition the vertices randomly, then the number of edges in the graph induced by any $V_i \cup V_j \cup V_k$ can be upper bounded by $O(m/n^{2/3})$ with high probability. By having the workload assigned to a vertex proportional to its degree, we show that all triangles can be listed by invoking the routing algorithm of Ghaffari, Kuhn, and Su [27] with a message load of $O(n^{1/3}) \cdot \deg(v)$ on each vertex v.

We first describe the algorithm behind Theorem 4 and then analyze it in Lemmas 2–5. To solve Triangle Enumeration and Counting simultaneously, our algorithm ensures that each triangle with at least two vertices in S is reported by *exactly one* vertex in V.

Low Degree Vertices. The first step of the algorithm for Theorem 4 is to deal with the low degree vertices. Recall that $\deg_{G^*}(v) = \deg_G(v)$ for each $v \in S$. Define

$$S^* = \left\{ v \in S \,\middle|\, \deg_{G^*}(v) \ge n^{1/3} \log^2 n \right\}.$$

Each $v \in S \setminus S^*$ can list all triangles involving v by learning all edges in $E(N(v))$ as follows: v transmits $N(v)$ to all its neighbors, and then each neighbor $u \in N(v)$ transmits $N(u) \cap N(v)$ to v. This process clearly finishes in $\tilde{O}(n^{1/3})$ rounds, and it can be executed in parallel for all $v \in S \setminus S^*$. To ensure that each triangle $T = \{x, y, z\}$ is reported by exactly one vertex, we let the vertex $v \in T \cap S$ with the minimum $\text{ID}(v)$ be the one responsible for reporting T.

In the subsequent discussion, we focus on the set of vertices S^* and the set of edges

$$E^* = \{e = \{u, v\} \in E \mid \{u, v\} \cap S^* \neq \emptyset\}.$$

That is, E^* is the set of edges with at least one endpoint in S^*, and all remaining triangles yet to be listed are in E^*. To avoid double counting, triangles that are completely within E^* are not reported in the algorithm above.

We write $m^* = |E^*|$. Note that the parameter $n = |V|$ still denotes the number of vertices in the underlying network G.

The Easy Case. We check whether any vertex $v^\star \in V$ has

$$\deg_{E^*}(v^\star) \geq \zeta \overset{\text{def}}{=} \frac{m^*}{10n^{1/3}\log n}.$$

If so, then we apply the routing algorithm of Ghaffari, Kuhn, and Su [27] on $G^* \cup \{v^\star\}$ to send all m^* edges E^* to $v^\star$. Observe that if $v^\star$ is adjacent to S, but not in S, then the mixing time of $G^* \cup \{v^\star\}$ differs from that of G^* by at most a constant factor. If $v^\star \in S$, then $\deg_{G^*}(v^\star) \geq \deg_{E^*}(v) \geq \zeta$. If $v^\star \notin S$, then the degree of $v^\star$ in $G^* \cup \{v^\star\}$ is still at least $\deg_{E^*}(v) \geq \zeta$. Therefore, we can send these m^* edges E^* to $v^\star$ by applying the routing algorithm of Reference [27], which takes $\tilde{O}(\tau_{\text{mix}}(G^* \cup \{v^\star\}) \cdot m^*/\zeta) = \tilde{O}(\tau_{\text{mix}}(G^*) \cdot n^{1/3})$ rounds. Thereafter, $v^\star$ can report all triangles in E^*.

Thus, in the analysis of the following steps, we may assume that the maximum degree in E^* among all vertices in V is at most $m^*/(10n^{1/3}\log n)$.

Vertex Classes. Define $\delta \overset{\text{def}}{=} 2^{\lfloor \log(m^*/n) \rfloor}$. For each $v \in S^*$, define k_v to be the real number such that $\deg_G(v) = \deg_{G^*}(v) = k_v \cdot \delta$. 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

$$\begin{aligned}
\sum_{v \in S^* : k_v \geq 1/2} 2k_v &= \sum_{v \in S^*} 2k_v - \sum_{v \in S^* : k_v < 1/2} 2k_v \\
&> \sum_{v \in S^*} 2k_v - n, && \text{because } |S^*| \leq |V| = n, \\
&\geq \frac{n}{m^*} \sum_{v \in S^*} \deg_{G^*}(v) - n, && \text{because } \delta \leq \frac{m^*}{n}, \\
&\geq 2n - n, && \text{because } \deg_{G^*}(v) \geq \deg_{E^*}(v), \\
&= n.
\end{aligned}$$

By applying Lemma 1 to reassign IDs, we may assume that the ID-space of S^* is $\{1, \ldots, |S^*|\}$ 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 Reference [17] discussed above. 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 S^*. 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 S^* 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.

In view of how vertex classes are defined, Lemma 1 guarantees that each vertex $v \in S^*$ knows the class of all vertices in S^*, 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 in $\big(E(V_{j_1}, V_{j_2}) \cup E(V_{j_2}, V_{j_3}) \cup E(V_{j_1}, V_{j_3})\big) \cap E^*$ and reporting those triangles (x_1, x_2, x_3) with $x_k \in V_{j_k}$ with these edges. It is clear that each triangle in E^* is reported by exactly one vertex.

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

We analyze the behavior of this algorithm in the CONGEST model, where the last step is implemented by applying the routing algorithm of Reference [27] to G^*, modified as in Section 2.1.

Lemma 2. *Consider a graph with $\bar{m}$ edges and $\bar{n}$ vertices. We generate a subset W by letting each vertex join W independently with probability p. Suppose that the maximum degree is $\Delta \le \bar{m}p/20 \log \bar{n}$ and $p^2\bar{m} \ge 400 \log^2 \bar{n}$. Then, with probability at least $1 - 10(\log \bar{n})/\bar{n}^5$, the number of edges in the subgraph induced by W 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 W, otherwise $x_i = 0$. Then $X = \sum_{i=1}^{\bar{m}} x_i$ is the number of edges in the subgraph induced by W. We have $\mathbf{E}[X] = p^2\bar{m}$, and by Markov's inequality,

$$\Pr[X \ge 6\mathbf{E}[X]] = \Pr[X^c \ge (6\mathbf{E}[X])^c] \le \frac{1}{6^c} \frac{\mathbf{E}[X^c]}{p^{2c}\bar{m}^c},$$

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

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

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 exactly k.

We project $(i_1, \ldots, i_c)$ 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 $k_j = 0$, then the number of choices is clearly at most m. If $k_j = 1$, then the number of choices is at most $(2c)(p\bar{m}/20 \log \bar{n})$, since one of its endpoints (which overlaps with the 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 $\Delta \le \bar{m}p/20 \log \bar{n}$ choices. If $k_j = 2$, then 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 $\binom{c}{x}\binom{c-x}{y} < 3^c$ is the number of distinct vectors $\langle k_1, \ldots, k_c \rangle$ realizing the given parameters c, x, and y. We bound

f_k as follows:

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

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

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

Therefore, since $c = 5\log\bar{n}$,

$$\Pr[X\ge 6\mathbf{E}[X]] \le \frac{1}{6^c}\frac{\mathbf{E}[X^c]}{p^{2c}\bar{m}^c} \le \frac{2c3^c}{6^c} \le \frac{10\log\bar{n}}{\bar{n}^5}.$$

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

We apply Lemma 2 to bound the size of the subgraphs induced by two parts of the vertex partition.

Lemma 3. *With probability at least* $1-1/n^4$, *we have* $|E(V_{j_1},V_{j_2})\cap E^*| \le 6\max\{m^*/n^{2/3}, 100\log^2 n\}$ *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 $p = 2n^{-1/3}$. Note that for the case of $j_1 = j_2$, the probability is $n^{-1/3}$, which is less then p.

We apply Lemma 2 to the subgraph induced by E^*. We use the parameters $\bar{m} = \max\{m^*, 100n^{2/3}\log^2 n\} \ge m^*$ and $\bar{n} = n$ with sampling probability $p = 2n^{-1/3}$ and $W = V_{j_1}\cup V_{j_2}$. By assumption, the maximum degree of the subgraph induced by E^* is at most $m^*/(10n^{1/3}\log n) \le \bar{m}p/(20\log\bar{n})$, since otherwise we go to the easy case.

For the case $m^* < 100n^{2/3}\log^2 n$, in the analysis, we can add virtual edges to the graph so that the total number of edges equals $\bar{m}$. This is possible without violating the maximum degree constraint $\Delta \le \bar{m}p/(20\log\bar{n})$.

Our choice of parameters satisfies the inequality $p^2\bar{m} \ge 400\log^2\bar{n}$ needed in Lemma 2, and hence we have $\Pr[|E(V_{j_1},V_{j_2})\cap E^*| > 6\bar{m}/n^{2/3}] \le \frac{10\log n}{n^5}$. Note that $|E(V_{j_1}\cup V_{j_2})\cap E^*| \ge |E(V_{j_1},V_{j_2})\cap E^*|$.

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. *With probability at least $1 - 1/n^4$, each vertex $v \in S^*$ receives $O(\deg_{G^*}(v) \cdot n^{1/3})$ edges.*

Proof. Consider any vertex $v \in S^*$. 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} and V_{j_2}. By Lemma 3, $|E(V_{j_1}, V_{j_2}) \cap E^*| = O(\max\{m^*/n^{2/3}, \log^2 n\})$ for all $j_1, j_2 \in [1, n^{1/3}]$ with probability at least $1 - 1/n^4$.

Note that m^* is a trivial upper bound on the number of edges v receives. If $m^* < n^{1/3} \deg_{G^*}(v)$, then we are done already. In the subsequent discussion, we assume $m^* \geq n^{1/3} \deg_{G^*}(v)$. Our choice of k_v implies $k_v = \Theta(\deg_{G^*}(v) \cdot n/m^*)$, and so v receives

$$\begin{aligned}
O\left(\max\left\{\frac{m^*}{n^{2/3}}, \log^2 n\right\}\right) \cdot 12k_v &= O\left(\max\left\{\deg_{G^*}(v) \cdot n^{1/3}, \deg_{G^*}(v) \cdot \frac{n}{m^*} \cdot \log^2 n\right\}\right)\\
&\leq O\left(\max\left\{\deg_{G^*}(v) \cdot n^{1/3}, n^{2/3} \log^2 n\right\}\right)\\
&\leq O\left(\deg_{G^*}(v) \cdot n^{1/3}\right).
\end{aligned}$$

messages, w.h.p. The first inequality is due to the assumption $m^* \geq n^{1/3} \deg_{G^*}(v)$, and the second inequality is due to the fact that $\deg_{G^*}(v) \geq n^{1/3} \log^2 n$ for each $v \in S^*$. □

Lemma 5. *Each vertex $v \in S^*$ sends $O(\deg_{G^*}(v) \cdot n^{1/3})$ edges with probability 1.*

Proof. Each $v \in S^*$ is responsible for sending its $\deg_{E^*}(v) \leq \deg_G(v) = \deg_{G^*}(v)$ incident edges in G, and each is involved in exactly $n^{1/3}$ triads. □

Lemmas 2–5 show that the message volume sent to/from every vertex is close to its expectation. By applying the routing algorithm of Reference [27] and Lemma 1, all messages can be routed in $\tilde{O}(n^{1/3})$ rounds. This concludes the proof of Theorem 4.

2.3 Subgraph Enumeration

In this section, we show that Theorem 4 can be extended to enumerating all s-vertex subgraphs in $\tilde{O}(n^{(s-2)/s})$ rounds. Note that the $\Omega(n^{1/3}/\log n)$ lower bound for Triangle Enumeration on Erdős-Rényi graphs $\mathcal{G}(n, 1/2)$ [32] can be generalized to an $\Omega(n^{(s-2)/s}/\log n)$ lower bound for enumerating s-vertex cliques; see Reference [24]. This implies that Theorem 5 is nearly optimal on $\mathcal{G}(n, 1/2)$ for enumerating s-vertex cliques. We note that Theorem 5 is obviously not optimal for certain s-vertex subgraphs. For example, all star graphs can be listed trivially without communication.

Theorem 5. *Let $s = O(1)$ be any constant. Given a connected graph G of n vertices, we can list all s-vertex subgraphs of G in $\tau_{\text{mix}}(G) \cdot \tilde{O}(n^{(s-2)/s})$ rounds, w.h.p., in the* CONGEST *model.*

It has been shown in Reference [17] 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 Ghaffari, Kuhn, and Su [27], does not immediately imply Theorem 5, since $\deg(v)$ could be much less than n.

Theorem 5 is proved in such a way that is almost the same as that of Theorem 4, and so we only highlight the difference. Let $G = (V, E)$, $n = |V|$, and $m = |E|$.

Similarly, we assume the maximum degree is $\Delta \leq m/(10n^{1/s} \log n)$, since otherwise we are in the easy case, where we can route all the information to one vertex $v^\star$ in $\tilde{O}(n^{1/s}) \leq \tilde{O}(n^{(s-2)/s})$ 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: $\{(i_1, \ldots, i_s) \mid 1 \le i_1 \le \cdots \le i_s \le n^{1/s}\}$. After 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 3, as follows.

Lemma 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 $p = 2n^{-1/s}$. Recall that the maximum degree is at most $m/\left(10n^{1/s}\log n\right) \le mp/20\log n$, and we also have $p^2 m = \Omega(n^{1-2s}) \gg 400\log^2 n$, as $m \ge n-1$. The lemma follows by applying Lemma 2 and using the same analysis in the proof of Lemma 3. □

Proof of Theorem 5. Here, we only consider the round complexity to deliver all messages. Consider a vertex $v \in V$. 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 $4s^2 k_v$ pairs (V_i, V_j). By Lemma 6, w.h.p., $|E(V_i, V_j)| = O(m/n^{2/s})$ for all i, j. Hence, the number of edges v receives is at most $O(m/n^{2/s}) \cdot 4s^2 k_v = O(\deg(v) \cdot n^{(s-2)/s})$, since $s = O(1)$ is a constant and $k_v = \Theta(\deg(v) \cdot n/m)$.

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 applying the routing algorithm of Ghaffari, Kuhn, and Su [27], all messages can be delivered in $\tilde{O}(\tau_{\text{mix}}(G) \cdot n^{(s-2)/s})$ rounds, w.h.p. □

3 DISTRIBUTED EXPANDER DECOMPOSITION

Our Triangle Enumeration algorithm depends on being able to compute a $(1/2, 1/\text{poly}\log n)$-expander decomposition in $\tilde{O}(n^{1/3})$ rounds. The goal of this section is to prove Theorem 2 (restated below), which offers a tradeoff between (ϵ, ϕ) and round complexity.

Theorem 2. *Let $\epsilon \in (0,1)$ and k be a positive integer. An (ϵ, ϕ)-expander decomposition with $\phi = (\frac{\log n}{\epsilon})^{2^{O(k)}}$ can be constructed in $n^{2/k} \cdot (\frac{\log n}{\epsilon})^{2^{O(k)}} = n^{2/k} \cdot \text{poly}(\frac{1}{\phi}, \log n)$ rounds, w.h.p.*

The most straightforward approach for constructing an expander decomposition of a graph $G = (V, E)$ is as follows. Find a ϕ-sparse cut S. If such a cut S does not exist, then return V as a part in the partition. Otherwise, recurse on both sides $G\{S\}$ and $G\{V - S\}$, and so the edges in $E(S, V - S)$ become inter-cluster edges. To see the correctness, once the recursion stops at $G\{U\}$ for some U, we know that $\Phi_{G\{U\}} \ge \phi$. Also, the total number of inter-cluster edges is at most $O(m\phi \log n)$, because (i) each inter-cluster edge can be charged to edges in the smaller side of some ϕ-sparse cut and (ii) each edge can be on the smaller side of the cut at most $\log m$ times.

This straightforward approach has two efficiency issues: (i) checking whether a ϕ-sparse cut exists is not known to be solvable by a fast distributed algorithm and is in fact NP-hard [45], (ii) a ϕ-sparse cut S can be very unbalanced and hence the recursion depth can be as high as $\tilde{\Omega}(n)$. Thus, even if we ignore the time spent on finding cuts, the round complexity due to the recursion depth is already too high.

To handle these two issues, an approach taken by the authors of References [47, 57, 62, 67] is as follows. First, they use *approximate sparse cut algorithms*, which either find some ϕ'-sparse cut or certify that there is no ϕ-sparse cut where $\phi' \gg \phi$. Second, they find a cut with some guarantee about the balance of the cut, i.e., the smaller side of the cut should be sufficiently large.

The proof of Theorem 2 is based on the following approach, also taken in References [47, 67], which requires an efficient algorithm for computing a *nearly most balanced* sparse cut. Intuitively, given that we can find a nearly most balanced sparse cut efficiently, the recursion depth should be made very small, since a major source of inefficiency in the approach above is that the cuts found can be very imbalanced. This intuition can be made formal using the ideas in the centralized setting from Nanongkai and Saranurak [47] and Wulff-Nilsen [67].

Our main technical contribution is twofold. First, we exhibit the first distributed algorithm for computing a nearly most balanced sparse cut, which is the key algorithmic tool underlying the proof of Theorem 2. Second, to obtain a fast distributed algorithm, we must modify the centralized approach of References [47, 67] to construct an expander decomposition. In particular, we need to run a *low diameter decomposition* [43, 46] whenever we encounter a graph with high diameter, as our distributed algorithm for finding a nearly most balanced sparse cut is fast only on graphs with low diameter.

Remark 2. Kuhn and Molla [40, Theorem 3] previously claimed that their sparse cut algorithm also outputs a nearly most balanced sparse cut, but this claim turns out to be incorrect (Anisur Rahaman Molla, personal communication, 2018). The high level reason is as follows. They showed a distributed implementation for finding a single sweep cut. However, this cut does not guarantee any balancedness. This issue was known in the centralized setting. Indeed, it is shown since the '90s that a sweep cut defined by the second eigenvector of the Laplacian gives an approximate sparsest cut with no balance guarantee and the cut can be computed in near-linear time.[2] Algorithms for finding approximately most balanced sparsest cuts in near-linear time was not known until the nibble algorithm by Spielman and Teng [62], which outputs a union of many sweep cuts. Intuitively, a reason that one needs to take a union of several cuts is that the most balanced sparsest cut might be a disjoint union of many small cuts.

3.1 Sparse Cut Computation

Our distributed nearly most balanced sparse cut algorithm is a distributed implementation of a modified version of the sequential algorithm of Spielman and Teng [62]. The algorithm of Reference [62] involves $\tilde{O}(m)$ *sequential* iterations of `Nibble` with a random starting vertex on the remaining subgraph. Roughly speaking, the procedure `Nibble` aims at finding a sparse cut by simulating a bounded-length random walk. The idea is that if the starting vertex v belongs to some sparse cut S, then it is likely that most of the probability mass will be trapped inside S. We show that $\tilde{O}(m)$ simultaneous iterations of an approximate version of `Nibble` with a random starting vertex can be implemented efficiently in CONGEST in $O(\text{poly}(1/\phi, \log n))$ rounds, where ϕ is the target conductance.

Theorem 6 (Nearly Most Balanced Sparse Cut). *Given a parameter $\phi = O(1/\log^5 n)$, there is an $O(D \cdot \text{poly}(\log n, 1/\phi))$-round algorithm $\mathcal{A}$ that achieves the following w.h.p.*

- *Suppose $\Phi(G) \leq \phi$ and define b to be $\text{bal}(S)$, where S is the most-balanced cut of G with conductance at most ϕ. The algorithm $\mathcal{A}$ is guaranteed to return a cut C with balance $\text{bal}(C) \geq \min\{b/2, 1/48\}$ and conductance $\Phi(C) = O(\phi^{1/3} \log^{5/3} n)$.*
- *Whenever $\Phi(G) > \phi$, the algorithm $\mathcal{A}$ either returns $C = \emptyset$ or returns a cut C with conductance $\Phi(C) = O(\phi^{1/3} \log^{5/3} n)$, but with no guarantee on $\text{bal}(C)$.*

The proof of Theorem 6 is deferred to Section 4. The problem of finding a sparse cut in the distributed setting has been studied prior to this work. Given that there is a ϕ-sparse cut and balance b, the algorithm of Das Sarma, Molla, and Pandurangan [59] finds a cut of conductance at most $\tilde{O}(\sqrt{\phi})$ in $\tilde{O}((n + (1/\phi))/b)$ rounds in CONGEST. The round complexity was later improved to $\tilde{O}(D + 1/(b\phi))$ by Kuhn and Molla [40]. These prior works have the following drawbacks:

[2]The power method for computing an approximate second eigenvector is shown in Reference [39]. The analysis that the sweep cut based on the second eigenvector is an approximate sparsest cut is implicit in Reference [60] and explicit in Reference [35].

(i) their running time depends on b, which can be as small as $O(1/n)$ and (ii) their output cuts are not guaranteed to be nearly most balanced.

3.2 Low Diameter Decomposition

The runtime of our distributed sparse cut algorithm (Theorem 6) is proportional to the diameter. To avoid running this algorithm on a high diameter graph, we employ a *low diameter decomposition* to cluster the current graph into components of small diameter.

Algorithm of Miller, Peng, and Xu. The low diameter decomposition algorithm of Miller, Peng, and Xu [46] can be implemented in CONGEST efficiently. Given a parameter $\beta \in (0,1)$, their algorithm decomposes the graph into clusters of diameter $O(\frac{\log n}{\beta})$ in $O(\frac{\log n}{\beta})$ rounds such that the number of inter-cluster edges is at most $\beta|E|$. Roughly, their algorithm is to let each vertex v sample $\delta_v \sim \text{Exponential}(\beta)$, $\beta \in (0,1)$, and then v is assigned to the cluster of u that minimizes $\text{dist}(u,v) - \delta_u$. A similar approach has been applied to construct a *network decomposition* [6, 23, 43].

An issue of the above algorithm is that the guarantee on the number of inter-cluster edges holds only *in expectation*. In sequential or parallel computation models, we can simply repeat the procedure several times and keep the best clustering. However, there is no way to choose the globally best execution in CONGEST without taking diameter time. It is possible to to achieve this guarantee *with high probability* at the cost of increasing the round complexity and the diameter of the clusters from $O(\frac{\log n}{\beta})$ to $O(\text{poly}(\log n, 1/\beta))$; see Reference [12].

Algorithm of Rozhon and Ghaffari. For our application, the sole purpose of the diameter requirement for clusters is to enable efficient communication within each cluster, and so it suffices to consider the following relaxed requirement: for each cluster V_i, there is a *Steiner tree* T_i of diameter $O\,(\text{poly}(\log n, 1/\beta))$ containing all vertices in V_i and possibly other vertices such that each edge in the graph belongs to at most $O\,(\text{poly}(\log n, 1/\beta))$ Steiner trees.

This relaxed version of low diameter decomposition can be constructed *deterministically* in $O\,(\text{poly}(\log n, 1/\beta))$ rounds using the approach of Rozhon and Ghaffari [56], with minor modifications. The difference is that in the decomposition of Reference [56] the clusters are required to be *non-adjacent*. To separate the clusters, they allow a small constant fraction of vertices to be removed. In our setting, it is the edges, not the vertices, that are removed to separate the clusters.

We present a brief description of the algorithm of Reference [56], with the small modifications that we need. Suppose each vertex is initially equipped with a distinct ID of $b = O(\log n)$ bits. At the beginning, each vertex v hosts a cluster $C = \{v\}$ whose identifier $\text{ID}(C)$ is initialized to $\text{ID}(v)$. This trivial clustering already satisfies the diameter requirement, but it does not meet the requirement on the number of inter-cluster edges.

The algorithm works in b phases. In each phase, the clustering will be updated, and during the process at most β/b fraction of edges will be removed from the graph, and so the total number of removed edges is at most $\beta|E|$. The induction hypothesis specifies that at the end of the ith phase, for any two clusters C_1 and C_2 such that $\text{ID}(C_1)$ and $\text{ID}(C_2)$ have different i-bit suffix, there is no edge connecting C_1 and C_2.

Thus, the goal of the ith phase is to achieve the following. For any *fixed* $(i-1)$-bit suffix Y, separate all clusters whose ID is of the form $(\cdots 0Y)$ (called *blue* clusters) from those whose ID of the form $(\cdots 1Y)$ (called *red* clusters). The ith phase of the algorithm consists of $k = (b/\beta) \cdot O(\log n)$ iterations in which blue clusters may acquire new members and red clusters lose members.

In each such iteration, each vertex in a red cluster that is adjacent to one or more blue clusters requests to join any one of the blue clusters. Now consider a blue cluster C. Define E_1 to be the set of edges inside C and let E_2 be the set of new edges that would be added to C if *all* join requests were accepted. There are two cases:

(1) if $|E_1| = 0$ or $|E_2|/|E_1| > \beta/b$, then all of C's join requests are accepted;
(2) otherwise, all edges in E_2 are removed from the graph.

After $k = \log_{1+\beta/b} m = O((b/\beta) \cdot \log n)$ iterations, all red clusters are separated from all blue clusters, since it is impossible for a blue cluster to enter Case 1 for all k iterations.

Whenever a blue cluster C enters Case 1, each new member v of C is attached to C's Steiner tree by including an edge $\{v, u\} \in E_2$ joining it to an existing member u of C. Therefore, the diameter of the final Steiner tree will be $O(kb) = O(\frac{\log^3 n}{\beta})$, and each edge belongs to at most $b = O(\log n)$ Steiner trees. The algorithm can be implemented in $O(\text{poly}(\log n, 1/\beta))$ rounds.

We summarize the discussion as a theorem.

Theorem 7 (Low-diameter Decomposition [56]). *Given a parameter $\beta \in (0, 1)$, there is a deterministic algorithm that decomposes the vertex set V into clusters $V = V_1 \cup \cdots \cup V_x$ in $O(\text{poly}(\log n, 1/\beta))$ rounds satisfying the following conditions.*

- *The number of inter-cluster edges is at most $\beta|E|$.*
- *Each cluster V_i is associated with a Steiner tree T_i such that the leaf vertices of T_i is V_i. The diameter of T_i is $O\left(\frac{\log^3 n}{\beta}\right)$. Each edge $e \in E$ belongs to at most $O(\log n)$ Steiner trees.*

We employ Theorem 7 in our distributed expander decomposition algorithm. We can also use the algorithm of Miller, Peng, and Xu [46], which is more efficient and offers a better guarantee on the diameter of clusters. However, as the guarantee on the number of inter-cluster edges of the algorithm of Reference [46] only holds in expectation, if we use this algorithm instead of Theorem 7 in our distributed expander decomposition algorithm, then the guarantee on the number of inter-cluster edges of the output expander decomposition will also be in expectation.

3.3 Main Algorithm

We are now in a position to describe the main algorithm for distributed expander decomposition, which is based on the approach of References [47, 67] with the aforementioned sparse cut (Theorem 6) and low diameter decomposition (Theorem 7) algorithms.

For the sake of convenience, we let

$$h(\theta) = \Theta\left(\theta^{1/3} \log^{5/3} n\right)$$

be an increasing function associated with Theorem 6 such that when we run the nearly most balanced sparse cut algorithm of Theorem 6 with conductance parameter θ, if the output subset C is non-empty, then it has $\Phi(C) \le h(\theta)$. We note that

$$h^{-1}(\theta) = \Theta\left(\theta^3 / \log^5 n\right).$$

Let $\epsilon \in (0, 1)$ and $k \ge 1$ be the parameters specified in Theorem 2. We define the following parameters that are used in our algorithm for Theorem 2.

Nearly Most Balanced Sparse Cut: We define $\phi_0 = O(\epsilon^3 / \log^8 n)$ in such a way that when we run the nearly most balanced sparse cut algorithm with this conductance parameter, any non-empty output C must satisfy $\Phi(C) \le h(\phi_0) = \frac{\epsilon/6}{\log\binom{n}{2}}$. For each $1 \le i \le k$, we define $\phi_i = h^{-1}(\phi_{i-1})$.

Low Diameter Decomposition: The parameter $\beta = O(\epsilon^2/\log n)$ for the low diameter decomposition is chosen as follows. Set $d = O(\epsilon^{-1}\log n)$ as the smallest integer such that $(1-\epsilon/12)^d \cdot 2\binom{n}{2} < 1$. Then, we define $\beta = (\epsilon/3)/d$.

We show that an (ϵ, ϕ)-expander decomposition can be constructed in $O(n^{2/k} \cdot \text{poly}(1/\phi, \log n))$ rounds, with conductance parameter $\phi = \phi_k = (\epsilon/\log n)^{2^{O(k)}}$. We will later see that $\phi = \phi_k$ is the smallest conductance parameter we ever use for applying the nearly most balanced sparse cut algorithm.

Algorithm. Our algorithm has two phases. In the algorithm there are three places where we remove edges from the graph, and they are tagged with Remove-j, for $1 \le j \le 3$, for convenience. Whenever we remove an edge $e = \{u, v\}$, we add a self-loop at both u and v, and so the degree of a vertex never changes throughout the algorithm. We never remove self-loops.

At the end of the algorithm, V is partitioned into connected components $V_1, \dots, V_x$ induced by the remaining edges. To prove the correctness of the algorithm, we will show that the number of removed edges is at most $\epsilon|E|$, and $\Phi_{G\{V_i\}} \ge \phi$ for each component V_i.

Phase 1.
The input graph is $G = (V, E)$.

(1) Run the low diameter decomposition algorithm (Theorem 7) with parameter β on G. Remove all inter-cluster edges (Remove-1).
(2) For each cluster U of the low diameter decomposition $V = V_1 \cup \cdots \cup V_x$, run the nearly most balanced sparse cut algorithm (Theorem 6) with parameter ϕ_0 on $G\{U\}$. Let C be the output subset.
   (a) If $C = \emptyset$, then the subgraph $G^* = G\{U\}$ quits Phase 1.
   (b) If $C \neq \emptyset$ and $\text{Vol}(C) \le (\epsilon/12)\,\text{Vol}(U)$, then the subgraph $G^* = G\{U\}$ quits Phase 1 and enters Phase 2.
   (c) Otherwise, remove the cut edges $E(C, U \setminus C)$ (Remove-2), and then we recurse on both sides $G\{C\}$ and $G\{U \setminus C\}$ of the cut.

We emphasize that we do not remove the cut edges in Step 2b of Phase 1.

Lemma 7. *The depth of the recursion of Phase 1 is at most d.*

Proof. Suppose there is still a cluster U entering the depth $d+1$ of the recursion of Phase 1. Then according to the threshold for $\text{Vol}(C)$ specified in Step 2b, we infer that $\text{Vol}(U) \le (1-\epsilon/12)^d\,\text{Vol}(V) < 1$ by our choice of d, which is impossible. □

Phase 2.
The input graph is $G^* = G\{U\}$. Define $\tau \overset{\text{def}}{=} ((\epsilon/6)\cdot \text{Vol}(U))^{1/k}$. Define the sequence: $m_1 \overset{\text{def}}{=} (\epsilon/6)\cdot\text{Vol}(U)$, and $m_i \overset{\text{def}}{=} m_{i-1}/\tau$, for each $1 < i \le k+1$. Initialize $L \leftarrow 1$ and $U' \leftarrow U$. Repeatedly do the following procedure.

- Run the nearly most balanced sparse cut algorithm (Theorem 6) with parameter ϕ_L on $G\{U'\}$. Let C be the output subset. Note that $\Phi_{G\{U'\}}(C) \le \phi_{L-1}$.
  —If $C = \emptyset$, then the subgraph $G\{U'\}$ quits Phase 2.
  —If $C \neq \emptyset$ and $\text{Vol}(C) \le m_L/(2\tau)$, then update $L \leftarrow L+1$.
  —Otherwise, update $U' \leftarrow U' \setminus C$, and remove all edges with one or both endpoints in C (Remove-3).

Intuitively, in Phase 2, we keep calling the nearly most balanced sparse cut algorithm to find a cut C and remove it. If we find a cut C that has volume greater than $m_L/(2\tau)$, then we are making good progress. If $\mathrm{Vol}(C) \leq m_L/(2\tau)$, then we learn that the volume of the most balanced sparse cut of conductance at most ϕ_L is at most $2 \cdot m_L/(2\tau) = m_L/\tau = m_{L+1}$ by Theorem 6, and so we move on to the next level by setting $L \leftarrow L + 1$.

The maximum possible level L is k. Since by definition $m_k/(2\tau) = 1/2 < 1$, there is no possibility to increase L to $k + 1$. Once we reach $L = k$, we will repeatedly run the nearly most balanced sparse cut algorithm until we get $C = \emptyset$ and quit.

When we remove a cut $C \neq \emptyset$ in Phase 2, each $u \in C$ becomes an isolated vertex with $\deg(u)$ self-loops, as all edges incident to u have been removed, and so in the final decomposition $V = V_1 \cup \cdots \cup V_x$, we can have $V_i = \{u\}$ for some i. We emphasize that we only do the edge removal when $\mathrm{Vol}(C) > m_L/(2\tau)$. Lemma 8 bounds the volume of the cuts found during Phase 2.

Lemma 8. *For each* $1 \leq i \leq k$, *define* C_i *as the union of all subsets* C *found in Phase 2 when* $L \geq i$. *Then* $\mathrm{Vol}(C_i) \leq m_i$.

Proof. We first consider the case of $i = 1$. Observe that the graph $G^* = G\{U\}$ satisfies the property that the most balanced sparse cut of conductance at most ϕ_0 has balance at most $2(\epsilon/12) = \epsilon/6$, since otherwise it does not meet the condition for entering Phase 2. Note that all cuts we find during Phase 2 have conductance at most ϕ_0, and so the union of them C_1 is also a cut of G^* with conductance at most ϕ_0. This implies that $\mathrm{Vol}(C_1) \leq (\epsilon/6)\,\mathrm{Vol}(U) = m_1$.

The proof for the case of $2 \leq i \leq k$ is exactly the same, as the condition for increasing L is to have $\mathrm{Vol}(C) \leq m_L/(2\tau)$. Let $G' = G\{U'\}$ be the graph considered in the iteration when we increase L from $i - 1$ to i. The existence of such a cut C of G' implies that the most balanced sparse cut of conductance at most ϕ_{i-1} of G' has volume at most $2\,\mathrm{Vol}(C) \leq m_{i-1}/\tau = m_i$. Similarly, note that all cuts we find when $L \geq i$ have conductance at most ϕ_{i-1}, and so the union of them C_i is also a cut of G' with conductance at most ϕ_{i-1}. This implies that $\mathrm{Vol}(C_i) \leq m_i$. □

Conductance of Remaining Components. For each $u \in V$, there are two possible ways for u to end the algorithm:

- During Phase 1 or Phase 2, the output of the nearly most balanced sparse cut algorithm on the cluster $G\{U\}$ that u belongs to is $C = \emptyset$. In this case, U becomes a component V_i in the final decomposition $V = V_1 \cup \cdots \cup V_x$. If ϕ' is the conductance parameter used in the nearly most balanced sparse cut algorithm, then $\Phi(G\{V_i\}) \geq \phi'$. Note that $\phi' \geq \phi_k = \phi$.
- During Phase 2, suppose $u \in C$ and $\mathrm{Vol}(C) > m_L/(2\tau)$, where C is the output of the nearly most balanced sparse cut algorithm. In this case, u itself becomes a component $V_i = \{u\}$ in the final decomposition $V = V_1 \cup \cdots \cup V_x$. Trivially, we have $\Phi(G\{V_i\}) \geq \phi$.

Therefore, we conclude that each component V_i in the final decomposition $V = V_1 \cup \cdots \cup V_x$ satisfies that $\Phi(G\{V_i\}) \geq \phi$.

Number of Removed Edges. There are three places in the algorithm where we remove edges. We show that, for each $1 \leq j \leq 3$, the number of edges removed due to Remove-j is at most $(\epsilon/3)|E|$, and so the total number of inter-cluster edges in the final decomposition $V = V_1 \cup \cdots \cup V_x$ is at most $\epsilon|E|$.

(1) By Lemma 7, the depth of recursion of Phase 1 is at most d. For each $i \in [1, d]$ the number of edges removed due to the low diameter decomposition algorithm during depth i of the recursion is at most $\beta|E|$, according to Theorem 7. By our choice of β, the number of edges removed due to Remove-1 is at most $d \cdot \beta|E| \leq (\epsilon/3)|E|$.

(2) For each edge $e \in E(C, U \setminus C)$ removed due to the nearly most balanced sparse cut algorithm in Phase 1, we charge the cost of the edge removal to some pairs (v, e) in the following way. If $\mathrm{Vol}(C) < \mathrm{Vol}(U \setminus C)$, for each $v \in C$, and for each edge e incident to v, then we charge the amount $|E(C, U \setminus C)| / \mathrm{Vol}(C)$ to (v, e); otherwise, for each $v \in U \setminus C$, and for each edge e incident to v, we charge the amount $|E(C, U \setminus C)| / \mathrm{Vol}(U \setminus C)$ to (v, e). Note that each pair (v, e) is charged at most $\log |E|$ times throughout the algorithm, and the amount per charging is at most $h(\phi_0)$. Therefore, the number of edges removed due to Remove-2 is at most $(\log |E|) \cdot h(\phi_0) \cdot 2|E| \le (\epsilon/3)|E|$ by our choice of ϕ_0.

(3) By Lemma 8, the summation of $\mathrm{Vol}(C)$ over all cuts C in $G^* = G\{U\}$ that are found and removed during Phase 2 due to Remove-3 is at most $m_1 = (\epsilon/6) \mathrm{Vol}(U)$. Therefore, the number of edges removed due to Remove-3 is at most $(\epsilon/6) \mathrm{Vol}(V) = (\epsilon/3)|E|$.

Round Complexity. During Phase 1, each vertex participates in at most $d = O(\epsilon^{-1} \log n)$ invocations of the nearly most balanced sparse cut algorithm and the low diameter decomposition algorithm. By our choice of parameters $\beta = O(\epsilon^2 / \log n)$ and $\phi_0 = O(\epsilon^3 / \log^8 n)$, the round complexity of both algorithms are $O(\mathrm{poly}(1/\epsilon, \log n))$, by Theorems 6 and 7.

Note that the low diameter decomposition algorithm of 7 guarantees that whenever we run the nearly most balanced sparse cut algorithm on $G\{U\}$, the diameter D of the Steiner tree associated with U is at most $(\frac{\log^3 n}{\beta}) = O(\mathrm{poly}(1/\epsilon, \log n))$, and each edge $e \in E$ belongs to at most $O(\log n)$ Steiner trees.

For Phase 2, Lemma 8 guarantees that for each $1 \le i \le k$ the algorithm can stay at $L = i$ for at most 2τ iterations. If we neither increase L nor quit Phase 2 for 2τ iterations, then we have $\mathrm{Vol}(C_L) > m_L$, contradicting Lemma 8. Therefore, the round complexity for Phase 2 can be upper bounded by

$$2\tau \sum_{i=1}^{k} O(\mathrm{poly}(1/\phi_i, \log n)) \le O\left(n^{2/k} \cdot \mathrm{poly}(1/\phi, \log n)\right).$$

During Phase 2, it is possible that the graph $G\{U'\}$ becomes disconnected or has large diameter. This does not cause any problems, since we can use the Steiner tree associated with $G^* = G\{U\}$ for communication during a sparse cut computation, and its diameter is at most $D = (\frac{\log^3 n}{\beta}) = O(\mathrm{poly}(1/\epsilon, \log n))$.

4 DISTRIBUTED NEARLY MOST BALANCED SPARSE CUT

The goal of this section is the prove Theorem 6, which we restate below for convenience. Although the parameter D in the theorem refers to the diameter of the graph $G = (V, E)$ under consideration, the theorem applies to the setting where there is a Steiner tree T of diameter D that contains all vertices in V and possibly some vertices outside of G, and all edges in T can be used for communication.

Theorem 6 (Nearly Most Balanced Sparse Cut). *Given a parameter $\phi = O(1/\log^5 n)$, there is an $O(D \cdot \mathrm{poly}(\log n, 1/\phi))$-round algorithm $\mathcal{A}$ that achieves the following w.h.p.*

- *Suppose $\Phi(G) \le \phi$ and define b to be $\mathrm{bal}(S)$, where S is the most-balanced cut of G with conductance at most ϕ. The algorithm $\mathcal{A}$ is guaranteed to return a cut C with balance $\mathrm{bal}(C) \ge \min\{b/2, 1/48\}$ and conductance $\Phi(C) = O(\phi^{1/3} \log^{5/3} n)$.*
- *Whenever $\Phi(G) > \phi$, the algorithm $\mathcal{A}$ either returns $C = \emptyset$ or returns a cut C with conductance $\Phi(C) = O(\phi^{1/3} \log^{5/3} n)$, but with no guarantee on $\mathrm{bal}(C)$.*

Overview. We will prove Theorem 6 by adapting the nearly most balanced sparse cut algorithm of Spielman and Teng [62] to CONGEST in a white-box manner.[3]

The main idea of Spielman and Teng [62], which is based on the work of Lovász and Simonovits [44], is as follows. Let S be a most-balanced sparse cut, then if we start a lazy random walk from a random vertex v in S for a small number of steps, then it is likely that most of the probably mass is confined in S. Moreover, if we arrange the vertices $v_1, v_2, \dots, v_n$ in the order of decreasing normalized probability mass $p(v)/\deg(v)$, then one of $C = \{v_1, v_2, \dots, v_i\}$ will be a sparse cut. This idea continues to work even if we truncate the random walk with a threshold $1/2^b$ that is approximately $1/\operatorname{Vol}(C)$. The procedure `Nibble` of Reference [62] is a realization of this idea.

The cut C returned by `Nibble` might be much smaller than S, as S itself might be a disjoint union of several small sparse cuts. Spielman and Teng [62] showed that a sparse cut C with $\operatorname{Vol}(C) = \Omega(\operatorname{Vol}(S))$ can be found by running `Nibble` sequentially for $\tilde{O}(|E)$ iterations with a random starting vertex v and a random truncation threshold b on the remaining part of the graph. The final cut C is the union of all cuts found.

The procedure `Nibble` of Reference [62] itself is not suitable for a distributed implementation, so we consider an approximate version of `Nibble` (Section 4.2) that can be efficiently implemented in CONGEST (Section 4.5). The nearly most balanced sparse cut algorithm of Spielman and Teng [62] involves doing $\tilde{O}(|E|)$ iterations of `Nibble` with a random starting vertex on the *remaining subgraph*. We will show that this sequential process can be parallelized at the cost of worsening the conductance guarantee by a polylogarithmic factor (Section 4.4).

Terminology. Given a parameter $\phi \in (0, 1)$, We define the following functions as in Reference [62]:

$$\begin{aligned}
\ell &\overset{\text{def}}{=} \lceil \log |E| \rceil, \\
t_0 &\overset{\text{def}}{=} 49 \ln(|E|e^2)/\phi^2, \\
f(\phi) &\overset{\text{def}}{=} \frac{\phi^3}{14^4 \ln^2(|E|e^4)}, \\
\gamma &\overset{\text{def}}{=} \frac{5\phi}{7 \cdot 7 \cdot 8 \cdot \ln(|E|e^4)}, \\
\epsilon_b &\overset{\text{def}}{=} \frac{\phi}{7 \cdot 8 \cdot \ln(|E|e^4) t_0 2^b}.
\end{aligned}$$

Let A be the adjacency matrix of the graph $G = (V, E)$. We assume a 1-1 correspondence between V and $\{1, \dots, n\}$. 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 $M = (AD^{-1} + I)/2$, where D is the diagonal matrix with $(\deg(1), \dots, \deg(n))$ on the diagonal.

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 $t \to \infty$, $p_t^v(x)$ approaches $\deg(x)/(2|E|)$, so it is natural to measure $p_t^v(x)$ *relative* to this baseline:

$$\varrho_t^v(x) = \frac{p_t^v(x)}{\deg(x)},$$

[3]There are many versions of Reference [62]; we refer to https://arxiv.org/abs/cs/0310051v9.

Let $f : V \mapsto [0,1]$ be any function. The truncation operation $[f]_\epsilon$ rounds $f(x)$ to zero if it falls below a threshold that depends on $\deg(x)$:

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

As in Reference [62], for any vertex set S, we define the vector χ_S by $\chi_S(u) = 1$ if $u \in S$ and $\chi_S(u) = 0$ if $u \notin S$, and we define the vector ψ_S by $\psi_S(u) = \deg(u)/\operatorname{Vol}(S)$ if $u \in S$ and $\psi_S(u) = 0$ if $u \notin S$. In particular, χ_v is a probability distribution on V that has all its probability mass on the vertex v, and ψ_V is the degree distribution of V. That is, $\mathbf{Pr}_{x \sim \psi_V}[x = v] = \deg(v)/\operatorname{Vol}(V)$.

4.1 Nibble

We first review the `Nibble` algorithm of Reference [62], which computes the following sequence of vectors with truncation parameter ϵ_b:

$$\tilde{p}_t^v = \begin{cases} \chi_v & \text{if } t = 0, \\ [M\tilde{p}_{t-1}^v]_{\epsilon_b} & \text{otherwise.} \end{cases}$$

We define $\tilde{\varrho}_t^v(u) = \tilde{p}_t^v(u)/\deg(u)$ as the normalized probability mass at u at time t. Due to truncation, for all $u \in V$ and $t \geq 0$, we have $p_t^v(u) \geq \tilde{p}_t^v(u)$ and $\varrho_t^v(u) \geq \tilde{\varrho}_t^v(u)$.

We define $\tilde{\pi}_t^v$ as a permutation of V such that

$$\tilde{\varrho}_t^v(\tilde{\pi}_t^v(1)) \geq \tilde{\varrho}_t^v(\tilde{\pi}_t^v(2)) \geq \cdots \geq \tilde{\varrho}_t^v(\tilde{\pi}_t^v(|V|)).$$

That is, we order the vertices by their $\tilde{\varrho}_t^v$-value, breaking ties arbitrarily, e.g., by comparing IDs. We write $\tilde{\pi}_t^v(i \ldots j)$ to denote the set of vertices $\tilde{\pi}_t^v(x)$ with $i \leq x \leq j$. For example, $\tilde{\pi}_t^v(1 \ldots j)$ is the set of the top j vertices with the highest $\tilde{\varrho}_t^v$-value.

```
Algorithm Nibble($G, v, \phi, b$)
For $t = 1$ to $t_0$, if there exists an index $1 \leq j \leq |V|$ meeting the following conditions
    (C.1) $\Phi(\tilde{\pi}_t^v(1 \ldots j)) \leq \phi$.
    (C.2) $\tilde{\varrho}_t^v(\tilde{\pi}_t^v(j)) \geq \gamma / \operatorname{Vol}(\tilde{\pi}_t^v(1 \ldots j))$.
    (C.3) $(5/6)\operatorname{Vol}(V) \geq \operatorname{Vol}(\tilde{\pi}_t^v(1 \ldots j)) \geq (5/7)2^{b-1}$.
then return $C = \tilde{\pi}_t^v(1 \ldots j)$ and quit. Otherwise return $C = \emptyset$.
```

Note that the definition of `Nibble`(G, v, ϕ, b) is exactly the same as the one presented in Reference [62].

Definition 2. Define $Z_{u,\phi,b}$ as the subset of V such that if we start the lazy random walk from $v \in Z_{u,\phi,b}$, then $\varrho_t^v(u) \geq 2\epsilon_b$ for at least one of $t \in [0, t_0]$. For any edge $e = \{u_1, u_2\}$, define $Z_{e,\phi,b} = Z_{u_1,\phi,b} \cup Z_{u_2,\phi,b}$.

Intuitively, if $v \notin Z_{e,\phi,b}$, then e does not participate in `Nibble`(G, v, ϕ, b) and both endpoints of e are not in the output C of `Nibble`(G, v, ϕ, b). In particular, $v \in Z_{e,\phi,b}$ is a necessary condition for $e \in E(C)$, The following auxiliary lemma establishes upper bounds on $\operatorname{Vol}(Z_{u,\phi,b})$ and $\operatorname{Vol}(Z_{e,\phi,b})$. This lemma will be applied to bound the amount of congestion when we execute multiple `Nibble` in parallel. Intuitively, if $\operatorname{Vol}(Z_{e,\phi,b})$ is small, then we can afford to run many instances `Nibble`(G, v, ϕ, b) in parallel for random starting vertices v sampled from the degree distribution ψ_V.

Lemma 9. *The following formulas hold for each vertex u and each edge e.*

$$\mathrm{Vol}(Z_{u,\phi,b}) \le (t_0+1)/(2\epsilon_b),$$
$$\mathrm{Vol}(Z_{e,\phi,b}) \le (t_0+1)/\epsilon_b.$$

In particular, these two quantities are both upper bounded by $O(\phi^{-5}2^b \log^3 |E|)$.

Proof. Recall that superscript is used to indicate the starting vertex of the lazy random walk. We write $Z_{u,\phi,b,t} = \{v \in V \mid \varrho_t^v(u) \ge 2\epsilon_b\}$. Then $\mathrm{Vol}(Z_{u,\phi,b}) \le \sum_{t=0}^{t_0} \mathrm{Vol}(Z_{u,\phi,b,t})$. Thus, to prove the lemma, if suffices to show that $\mathrm{Vol}(Z_{u,\phi,b,t}) \le 1/(2\epsilon_b)$. This inequality follows from the fact that $\varrho_t^v(u) = \varrho_t^u(v)$, as follows:

$$\begin{aligned} 1 &= \sum_{v\in V} p_t^u(v) \\ &\ge \sum_{v\in V\,:\,\varrho_t^u(v)\ge 2\epsilon_b} p_t^u(v) \\ &\ge \sum_{v\in V\,:\,\varrho_t^u(v)\ge 2\epsilon_b} 2\epsilon_b \cdot \deg(v) \\ &= \sum_{v\in V\,:\,\varrho_t^v(u)\ge 2\epsilon_b} 2\epsilon_b \cdot \deg(v) \\ &= 2\epsilon_b \cdot \mathrm{Vol}(Z_{u,\phi,b,t}). \end{aligned}$$

The fact that $\varrho_t^v(u) = \varrho_t^u(v)$ has been observed in Reference [62] without a proof. For the sake of completeness, we will show a proof of this fact. An alternative proof can be found in Reference [11, Lemma 3.7]. In the following calculation, we use the fact that $D^{-1}MD = D^{-1}(AD^{-1}+I)D/2 = (D^{-1}A+I)/2 = M^\top$:

$$\begin{aligned} \varrho_t^v(u) &= \chi_u^\top D^{-1} M^t \chi_v \\ &= \chi_u^\top (D^{-1}MD)^t (D^{-1}\chi_v) \\ &= \chi_u^\top (M^\top)^t (D^{-1}\chi_v) \\ &= (D^{-1}\chi_v)^\top M^t \chi_u \\ &= \chi_v^\top D^{-1} M^t \chi_u \\ &= \varrho_t^u(v). \end{aligned}$$

Finally, recall that $\epsilon_b = \Theta(\frac{\phi}{t_0 2^b \ln |E|})$ and $t_0 = \Theta(\frac{\ln |E|}{\phi^2})$, and so

$$\mathrm{Vol}(Z_{e,\phi,b}) \le 2(t_0+1)/(2\epsilon_b) = O(\phi^{-5}2^b \log^3 |E|).$$

□

Lemma 10 lists some crucial properties of `Nibble`. In subsequent discussion, for any given subset $S \subset V$, the subset $S^g \subseteq S$ and the partition $S^g = \bigcup_{b=1}^{\ell} S_b^g$ are defined according to Lemma 10.

Lemma 10 (Analysis of `Nibble`). *For each $\phi \in (0,1]$, and for each subset $S \subset V$ satisfying*

$$\mathrm{Vol}(S) \le \frac{2}{3}\cdot \mathrm{Vol}(V) \quad \text{and} \quad \Phi(S) \le 2f(\phi),$$

there exists a subset $S^g \subseteq S$ with the following properties. First, $\mathrm{Vol}(S^g) \ge \mathrm{Vol}(S)/2$. *Second, S^g is partitioned into $S^g = \bigcup_{b=1}^{\ell} S_b^g$ such that if a lazy random walk is initiated at any $v \in S_b^g$ with truncation parameter ϵ_b, the following are true.*

(1) *The set C returned by* `Nibble`(G, v, ϕ, b) *is non-empty.*
(2) *Let $\lambda > 1/5$ be a number. For any $1 \le t \le t_0$ and j satisfying $\tilde{\varrho}_t^v(\tilde{\pi}_t^v(j)) \ge \lambda\gamma / \mathrm{Vol}(\tilde{\pi}_t^v(1 \dots j))$, we have $\mathrm{Vol}(\tilde{\pi}_t^v(1 \dots j) \cap S) \ge (1 - \frac{1}{5\lambda}) \mathrm{Vol}(\tilde{\pi}_t^v(1 \dots j))$. In particular, the set C returned by* `Nibble`(G, v, ϕ, b) *satisfies* $\mathrm{Vol}(C \cap S) \ge (4/7)2^{b-1}$.

Proof. The first condition follows from Reference [62, Lemma 3.1]. The second condition follows from the proof of Reference [62, Lemma 3.14]. To see that the set C returned by `Nibble`(G, v, ϕ, b) satisfies $\mathrm{Vol}(C \cap S) \ge (4/7)2^{b-1}$, observe that by (C.3), the set C satisfies $\mathrm{Vol}(C) \ge (5/7)2^{b-1}$. Setting $\lambda = 1$, (C.2) implies that $\mathrm{Vol}(C \cap S) \ge (1 - \frac{1}{5}) \mathrm{Vol}(C) \ge (4/5)(5/7)2^{b-1} = (4/7)2^{b-1}$. □

To put it another way, Lemma 10(1) says that there exist $1 \le t \le t_0$ and $1 \le j \le |V|$ such that (C.1)–(C.3) are met; Lemma 10(2) says that if t and j satisfy (C.2), then the set $\tilde{\pi}_t^v(1 \dots j)$ has high overlap with S. The reason that we allow a general value of λ in Lemma 10(2) is that we will use it with $\lambda \ne 1$ to analyze a modified version of `Nibble` in Section 4.2.

Intuitively, the set S^g represents the "core" of S in the sense that `Nibble`(G, v, ϕ, b) is guaranteed to return a sparse cut C if $v \in S_b^g$. Recall that (C.1) and (C.3) in the description of `Nibble`(G, v, ϕ, b) guarantees that the cut C has conductance at most ϕ and has volume at most $(5/6)\,\mathrm{Vol}(V)$.

4.2 Approximate Nibble

The algorithm `Nibble` is not suitable for a distributed implementation, since it has to go over all possible j. We provide a modified version of `Nibble` that only considers $O(\phi^{-1} \log(\mathrm{Vol}(V)))$ choices of j for each t. The cost of doing so is that we have to relax the conditions slightly.

For any fixed t, we define the sequence (j_i) as follows. Define $j_{\max}$ to be the largest index w.r.t. $\tilde{\pi}_t^v$ for which $\tilde{p}_t^v(j_{\max}) > 0$. In the base case $j_1 = 1$. Once $j_1, \dots, j_{i-1}$ are defined, if $j_{i-1} < j_{\max}$, then j_i is defined to be

$$j_i = \max\left\{j_{i-1} + 1,\ \ \arg\max_{1 \le j \le j_{\max}} \left(\mathrm{Vol}(\tilde{\pi}_t^v(1 \dots j)) \le (1 + \phi)\,\mathrm{Vol}(\tilde{\pi}_t^v(1 \dots j_{i-1}))\right)\right\}.$$

Algorithm `ApproximateNibble`(G, v, ϕ, b)
For $t = 1$ to t_0, we go over all $O(\phi^{-1} \log(\mathrm{Vol}(V)))$ candidates j in the sequence (j_x). If $j_x = 1$ or $j_x = j_{x-1} + 1$, then we test whether (C.1), (C.2), and (C.3) are met. Otherwise, we test whether the following modified conditions are met.

(C.1*) $\Phi(\tilde{\pi}_t^v(1 \dots j_x)) \le 12\phi$.
(C.2*) $\tilde{\varrho}_t^v(\tilde{\pi}_t^v(j_{x-1})) \ge \gamma / \mathrm{Vol}(\tilde{\pi}_t^v(1 \dots j_x))$.
(C.3*) $(11/12)\,\mathrm{Vol}(V) \ge \mathrm{Vol}(\tilde{\pi}_t^v(1 \dots j_x)) \ge (5/7)2^{b-1}$.

If some j_x passes the test, then return $C = \tilde{\pi}_t^v(1 \dots j_x)$ and quit. Otherwise return $C = \emptyset$.

Definition 3. Consider `ApproximateNibble`(G, v, ϕ, b). Define P^* as the set of edges e such that there exist at least one endpoint u of e and at least one number $t \in [0, t_0]$ with $\tilde{p}_t^v(u) > 0$.

Intuitively, P^* is the set of edges that participate in `ApproximateNibble`(G, v, ϕ, b). This notation will be used in the analysis of the complexity of our distributed implementation.

Lemma 11 shows an additional property of the output C of `ApproximateNibble`(G, v, ϕ, b) when v is appropriately chosen. Note that if C is non-empty, it must have conductance at most 12ϕ and volume at most $(11/12)\,\mathrm{Vol}(V)$ in view of (C.1*) and (C.3*).

Lemma 11 (Analysis of `ApproximateNibble`). *For each $0 < \phi \le 1/12$, and for each subset $S \subset V$, satisfying*

$$\mathrm{Vol}(S) \le \frac{2}{3} \cdot \mathrm{Vol}(V) \quad \textit{and} \quad \Phi(S) \le 2f(\phi),$$

the output C of `ApproximateNibble`(G, v, ϕ, b) *for any $v \in S_g^b$ is non-empty and it satisfies*

$$\mathrm{Vol}(C \cap S) \ge 2^{b-2}.$$

Proof. We pick (t, j) as the indices that satisfy (C.1)–(C.3), whose existence is guaranteed by Lemma 10(1). Let $v \in S_g^b$. We select x in such a way that $j_{x-1} < j \le j_x$. We will show that j_x will pass the test in `ApproximateNibble`(G, v, ϕ, b), and the output $C = \tilde{\pi}_t^v(1 \ldots j_x)$ satisfies $\mathrm{Vol}(C \cap S) \ge 2^{b-2}$.

For the easy special case that $j = j_i$ for some i, the index j_i is guaranteed to pass the test in `ApproximateNibble`(G, v, ϕ, b), and we have $\mathrm{Vol}(C \cap S) \ge (4/7)2^{b-1} > 2^{b-2}$ by Lemma 10.

Otherwise, the three indices $j_{x-1} < j < j_x$ satisfy the following relation:

$$\mathrm{Vol}(\tilde{\pi}_t^v(1 \ldots j_{x-1})) \le \mathrm{Vol}(\tilde{\pi}_t^v(1 \ldots j)) \le \mathrm{Vol}(\tilde{\pi}_t^v(1 \ldots j_x)) \le (1 + \phi)\,\mathrm{Vol}(\tilde{\pi}_t^v(1 \ldots j_{x-1})).$$

We first show that j_x satisfies the three conditions (C.1*), (C.2*), (C.3*), and so it will pass the test in `ApproximateNibble`(G, v, ϕ, b), and then we show that the output C satisfies $\mathrm{Vol}(C \cap S) \ge 2^{b-2}$.

Condition (C.1):* We divide the analysis into two cases.

- Consider the case $\mathrm{Vol}(\tilde{\pi}_t^v(1 \ldots j_x)) \le \mathrm{Vol}(V)/2$. We have $|\partial(\tilde{\pi}_t^v(1 \ldots j_x))| \le |\partial(\tilde{\pi}_t^v(1 \ldots j))| + \phi\,\mathrm{Vol}(\tilde{\pi}_t^v(1 \ldots j_x)) \le 2\phi\,\mathrm{Vol}(\tilde{\pi}_t^v(1 \ldots j)) \le 2\phi\,\mathrm{Vol}(\tilde{\pi}_t^v(1 \ldots j_x))$. Hence,
$$\Phi(\tilde{\pi}_t^v(1 \ldots j_x)) = |\partial(\tilde{\pi}_t^v(1 \ldots j_x))| / \mathrm{Vol}(\tilde{\pi}_t^v(1 \ldots j_x)) \le 2\phi,$$
and so (C.1*) is met. In the above calculation, we use the fact that $|\partial(\tilde{\pi}_t^v(1 \ldots j))| \le \phi\,\mathrm{Vol}(\tilde{\pi}_t^v(1 \ldots j))$, which is due to the assumption that (t, j) satisfies (C.1).
- Consider the case $\mathrm{Vol}(\tilde{\pi}_t^v(1 \ldots j_x)) > \mathrm{Vol}(V)/2$. The last inequality in the following calculation uses the fact that $\mathrm{Vol}(V \setminus \tilde{\pi}_t^v(1 \ldots j)) \ge (1/6)\,\mathrm{Vol}(V)$, which is due to the assumption that (t, j) satisfies (C.2).
$$\begin{aligned}
\mathrm{Vol}(V \setminus \tilde{\pi}_t^v(1 \ldots j_x)) &\ge \mathrm{Vol}(V \setminus \tilde{\pi}_t^v(1 \ldots j)) - \phi\,\mathrm{Vol}(\tilde{\pi}_t^v(1 \ldots j_x)) \\
&\ge \mathrm{Vol}(V \setminus \tilde{\pi}_t^v(1 \ldots j)) - (1/12)\,\mathrm{Vol}(\tilde{\pi}_t^v(1 \ldots j_x)) && \phi \ge 1/12 \\
&\ge \mathrm{Vol}(V \setminus \tilde{\pi}_t^v(1 \ldots j)) - (1/12)\,\mathrm{Vol}(V) \\
&\ge \mathrm{Vol}(V \setminus \tilde{\pi}_t^v(1 \ldots j))/2. && (*)
\end{aligned}$$
We are ready to show that $\Phi(\tilde{\pi}_t^v(1 \ldots j_x)) \le 12\phi$.

$$\begin{aligned}
\Phi(\tilde{\pi}_t^v(1 \ldots j_x)) &= |\partial(\tilde{\pi}_t^v(1 \ldots j_x))| / \mathrm{Vol}(V \setminus \tilde{\pi}_t^v(1 \ldots j_x)) \\
&\le \frac{\phi\,\mathrm{Vol}(V \setminus \tilde{\pi}_t^v(1 \ldots j)) + \phi\,\mathrm{Vol}(\tilde{\pi}_t^v(1 \ldots j))}{\mathrm{Vol}(V \setminus \tilde{\pi}_t^v(1 \ldots j_x))} \\
&\le \frac{6\phi\,\mathrm{Vol}(V \setminus \tilde{\pi}_t^v(1 \ldots j))}{\mathrm{Vol}(V \setminus \tilde{\pi}_t^v(1 \ldots j_x))} && \mathrm{Vol}(\tilde{\pi}_t^v(1 \ldots j)) \le (5/6)\,\mathrm{Vol}(V) \\
&\le 12\phi. && \text{use } (*)
\end{aligned}$$

Condition (C.2):*

$$\begin{aligned}
\tilde{\varrho}_t^v(\tilde{\pi}_t^v(j_{x-1})) &\ge \tilde{\varrho}_t^v(\tilde{\pi}_t^v(j)) && j_{x-1} \le j \\
&\ge \gamma / \mathrm{Vol}(\tilde{\pi}_t^v(1 \ldots j)) && (t, j) \text{ satisfies (C.2)} \\
&\ge \gamma / \mathrm{Vol}(\tilde{\pi}_t^v(1 \ldots j_x)). && j \le j_x
\end{aligned}$$

Condition (C.3):*

$$\begin{aligned}
(11/12)\operatorname{Vol}(V) &> (5/6)(1+\phi)\operatorname{Vol}(V) && \phi \le 1/12\\
&\ge (1+\phi)\operatorname{Vol}(\tilde{\pi}^v_t(1\dots j)) && (t,j) \text{ satisfies (C.3)}\\
&\ge \operatorname{Vol}(\tilde{\pi}^v_t(1\dots j_x))\\
&\ge \operatorname{Vol}(\tilde{\pi}^v_t(1\dots j)) && j \le j_x\\
&\ge (5/7)2^{b-1}. && (t,j) \text{ satisfies (C.3)}
\end{aligned}$$

Lower Bound of $\operatorname{Vol}(C \cap S)$. First, observe that (C.2*) implies

$$\tilde{\varrho}^v_t(\tilde{\pi}^v_t(j_{x-1})) \ge \frac{\gamma}{\operatorname{Vol}(\tilde{\pi}^v_t(1\dots j_x))} \ge \frac{\gamma}{(1+\phi)\operatorname{Vol}(\tilde{\pi}^v_t(1\dots j_{x-1}))} = \frac{(12/13)\gamma}{\operatorname{Vol}(\tilde{\pi}^v_t(1\dots j_{x-1}))}.$$

By Lemma 10, we can lower bound $\operatorname{Vol}(C \cap S)$ as follows:

$$\begin{aligned}
\operatorname{Vol}(C\cap S) &= \operatorname{Vol}(\tilde{\pi}^v_t(1\dots j_x)\cap S)\\
&> \operatorname{Vol}(\tilde{\pi}^v_t(1\dots j_{x-1})\cap S)\\
&\ge \left(1-\frac{13}{5\cdot 12}\right)\operatorname{Vol}(\tilde{\pi}^v_t(1\dots j_{x-1})) && \text{Lemma 10}\\
&\ge \left(1-\frac{13}{60}\right)\operatorname{Vol}(\tilde{\pi}^v_t(1\dots j_x))/(1+\phi)\\
&\ge \left(1-\frac{13}{60}\right)(5/7)2^{b-1}/(1+\phi) && \text{(C.3*)}\\
&> 2^{b-2}. && \phi \le 1/12
\end{aligned}$$

□

Recall that our goal is to design a distributed algorithm that finds a nearly most balanced sparse cut, so finding a cut C with low conductance is not enough. Following the approach of Reference [62], to find a nearly most balanced sparse cut, we will need to take the union of the output of multiple instances of ApproximateNibble, and the goal of the analysis is to show that the resulting vertex set has volume at least $\operatorname{Vol}(S)/2$. This explains the reason why we not only need to show that $C \neq \emptyset$ but also need to show a lower bound on $\operatorname{Vol}(C \cap S)$ in Lemma 11.

4.3 Random Nibble

Next, we consider the algorithm RandomNibble which executes ApproximateNibble with a random starting vertex v and a random parameter b. The definition of RandomNibble is exactly the same as the corresponding one in Reference [62] except that we use ApproximateNibble instead of Nibble.

Algorithm RandomNibble(G, ϕ)
Sample a starting vertex $v \sim \psi_V$ according to the degree distribution. Choose a number $b \in [1, \ell]$ with $\mathbf{Pr}[b = i] = 2^{-i}/(1-2^{-\ell})$. Execute ApproximateNibble(G, v, ϕ, b), and return the result C.

Recall that P^* is the set of edges participating in the subroutine ApproximateNibble(G, v, ϕ, b), as defined in Definition 3. Note that $E(C) \subseteq P^*$, where C is the output of RandomNibble(G, ϕ).

Lemma 12 (Analysis of RandomNibble). *For each* $\phi \in (0, 1/12]$, *the output* C *of* RandomNibble(G, ϕ) *satisfies the following:*

(1) $\mathbf{Pr}[e \in E(C)] \leq \mathbf{Pr}[e \in P^*] \leq \ell(t_0+1)\epsilon_0^{-1}/\operatorname{Vol}(V)$ *for each* $e \in E$.
(2) $\mathbf{E}[\operatorname{Vol}(C \cap S)] \geq \frac{\operatorname{Vol}(S)}{8\operatorname{Vol}(V)}$ *for each subset* $S \subset V$ *satisfying*

$$\operatorname{Vol}(S) \leq \frac{2}{3} \cdot \operatorname{Vol}(V) \quad \textit{and} \quad \Phi(S) \leq 2f(\phi).$$

Proof. The proof of $\mathbf{E}[\operatorname{Vol}(C \cap S)] \geq \frac{\operatorname{Vol}(S)}{8\operatorname{Vol}(V)}$ follows from Lemma 11 and the proof of Reference [62, Lemma 3.2]. An upper bound of $\mathbf{Pr}[e \in P^*]$ can be calculated using Lemma 9. More specifically, observe that $v \in Z_{e,\phi,i}$ is a necessary condition for $e \in E(C)$ for the case $b = i$, and so we can upper bound $\mathbf{Pr}[e \in P^*]$ as follows:

$$\begin{aligned}
\mathbf{Pr}[e \in P^*] &\leq \sum_{i=1}^{\ell} \mathbf{Pr}[b=i] \cdot \mathbf{Pr}[v \in Z_{e,\phi,i}] \\
&\leq \sum_{i=1}^{\ell} \mathbf{Pr}[b=i] \cdot \operatorname{Vol}(Z_{e,\phi,i})/\operatorname{Vol}(V) \\
&\leq \sum_{i=1}^{\ell} \frac{2^{-i}}{1-2^{-\ell}} \cdot ((t_0+1)/\epsilon_i)/\operatorname{Vol}(V) && \operatorname{Vol}(Z_{e,\phi,i}) \leq (t_0+1)/\epsilon_i \\
&< \sum_{i=1}^{\ell} (t_0+1)\epsilon_0^{-1}/\operatorname{Vol}(V) && \epsilon_i = \epsilon_0/2^i \\
&= \ell(t_0+1)t_0\epsilon_0^{-1}/\operatorname{Vol}(V),
\end{aligned}$$

where the second inequality follows from the fact that we sample a starting vertex $v \sim \psi_V$ according to the degree distribution. □

4.4 Parallel Nibble

In Reference [62], roughly speaking, it was shown that a nearly most balanced sparse cut can be found with probability $1-p$ by *sequentially* applying `Nibble` with a random starting vertex for $O(|E|\log(1/p))$ times on the *remaining graph* after removing the previous cuts. The final output is then the union of all cuts found.

To facilitate an efficient distributed implementation, we will do these $O(|E|\log(1/p))$ sparse cut computations *in moderate-sized batches.* Note that the naïve approach of doing all $O(|E|\log(1/p))$ `RandomNibble` in parallel on the graph G does not work, since the potentially high overlap between the output subsets of different executions of `RandomNibble` will destroy the required conductance constraint.

Specifically, we consider the algorithm `ParallelNibble`, which involves a simultaneous execution of a moderate number of `ApproximateNibble`. In the description of `ParallelNibble`, we say that e participates in the subroutine `RandomNibble`(G,ϕ) if $e \in P^*$ for the subroutine `ApproximateNibble`(G,v,ϕ,b) during the execution of `RandomNibble`(G,ϕ). We write

$$k \overset{\text{def}}{=} \left\lceil \frac{\operatorname{Vol}(V)}{\ell(t_0+1)\epsilon_0^{-1}} \right\rceil$$

in subsequent discussion.

Algorithm `ParallelNibble`(G, ϕ)
For $i = 1$ to k, do `RandomNibble`(G, ϕ), in parallel. Let C_i be the result of the ith execution of `RandomNibble`(G, ϕ). Let $U_i = \bigcup_{j=1}^{i} C_i$. If there exists an edge e participating in the subroutine `RandomNibble`(G, ϕ) for more than $w \overset{\text{def}}{=} 10 \lceil \ln(\text{Vol}(V)) \rceil$ times, then return $C = \emptyset$. Otherwise select $i^* \in [1, k]$ to be the highest index such that $\text{Vol}(U_{i^*}) \le z \overset{\text{def}}{=} (23/24)\,\text{Vol}(V)$. Return $C = U_{i^*}$.

In the statement of Lemma 13, we define the function g by

$$g(\phi, \text{Vol}(V)) \overset{\text{def}}{=} \left\lceil 10w \cdot \ell(t_0 + 1)\epsilon_0^{-1} \right\rceil = O(\phi^{-5} \log^5(|E|)).$$

In particular, we have $10w\,\text{Vol}(V)/k \le g(\phi, \text{Vol}(V))$. The function g will also be used in the description and the analysis of `Partition` in the subsequent discussion.

Lemma 13 (Analysis of `ParallelNibble`). *For each $0 < \phi \le 1/12$ the following is true for the output C of* `ParallelNibble`(G, ϕ).

(1) *If $C \ne \emptyset$, then $\Phi(C) \le 276w\phi$.*
(2) *For each subset $S \subset V$ satisfying*

$$\text{Vol}(S) \le \frac{2}{3} \cdot \text{Vol}(V) \quad \textit{and} \quad \Phi(S) \le 2f(\phi),$$

define the random variable y as follows:

$$y = \begin{cases} \text{Vol}(S), & \textit{if } \text{Vol}(C) \ge (1/24)\,\text{Vol}(V) \\ \text{Vol}(C \cap S), & \textit{otherwise.} \end{cases}$$

Then, $\mathbf{E}[y] \ge \frac{k\,\text{Vol}(S)}{10w\,\text{Vol}(V)} \ge \frac{\text{Vol}(S)}{g(\phi, \text{Vol}(V))}$.

Proof. We show that if the output subset C is non-empty, then we must have $\Phi(C) \le 276w\phi$. By definition of `ParallelNibble`, if the output C is non-empty, then each edge e incident to C is incident to at most w of these vertex sets $C_1, \ldots, C_{i^*}$. Therefore, $\text{Vol}(C) \ge (1/w) \sum_{i=1}^{i^*} \text{Vol}(C_i)$. Using the fact that the output C_i of `ApproximateNibble`(G, v, ϕ, b) has $\Phi(C_i) \le 12\phi$, we upper bound $|\partial(C)|$ as follows:

$$\begin{aligned} |\partial(C)| &\le \sum_{i=1}^{i^*} |\partial(C_i)| \\ &\le \sum_{i=1}^{i^*} 12\phi\,\text{Vol}(C_i) \\ &\le 12w\phi\,\text{Vol}(C). \end{aligned}$$

The threshold z guarantees that $\text{Vol}(V \setminus C) \ge (1/23)\,\text{Vol}(C)$, and so $|\partial(C)| \le 12 \cdot 23 \cdot w\phi\,\text{Vol}(V \setminus C) = 276w\phi\,\text{Vol}(V \setminus C)$. We conclude that $\Phi(C) \le 276w\phi$.

Next, we analyze the random variable y. We first observe that if $i^* < k$, then $C = U_{i^*}$ has $\text{Vol}(C) \ge (1/24)\,\text{Vol}(V)$. This is because that each C_i must have $\text{Vol}(C_i) \le (11/12)\,\text{Vol}(V)$ by definition of `ApproximateNibble`. If $\text{Vol}(U_{i^*}) < (1/24)\,\text{Vol}(V)$, then $\text{Vol}(U_{i^*+1}) < (1/24)\,\text{Vol}(V) + (11/12)\,\text{Vol}(V) < (23/24)\,\text{Vol}(V)$, contradicting the choice of i^*. Thus, for the case $i^* < k$, we automatically have $y = \text{Vol}(S)$, which is the maximum possible value of y. In view of this, we can lower bound $\mathbf{E}[y]$ as follows:

$$\mathbf{E}[y] \ge \mathbf{E}[\text{Vol}(U_k \cap S)] - \mathbf{Pr}[B] \cdot \text{Vol}(S),$$

where B is the event that there exists an edge participating in the subroutine RandomNibble(G, ϕ) for more than w times. Note that B implies $C = \emptyset$, but not vise versa.

By Lemma 12, we know that $\mathbf{E}[\mathrm{Vol}(C_i \cap S)] \geq \frac{\mathrm{Vol}(S)}{8\,\mathrm{Vol}(V)}$, and this implies $\mathbf{E}[\mathrm{Vol}(U_k \cap S)] \geq (1/w) \sum_{i=1}^{k} \mathbf{E}[\mathrm{Vol}(C_i \cap S)] = \frac{k\,\mathrm{Vol}(S)}{8w\,\mathrm{Vol}(V)}$. Therefore, to obtain the desired bound $\mathbf{E}[y] \geq \frac{k\,\mathrm{Vol}(S)}{10w\,\mathrm{Vol}(V)}$, it remains to show that $\mathbf{Pr}[B] \leq \frac{k}{40w\,\mathrm{Vol}(V)}$.

If $k = 1 \leq w$, then $\mathbf{Pr}[B] = 0$. In what follows, we assume $k \geq 2$, and this, together with the analysis of RandomNibble(G, ϕ) in Lemma 12, implies that for each invocation of RandomNibble(G, ϕ), we have

$$\mathbf{Pr}[e \in P^*] \leq \ell(t_0 + 1)\epsilon_0^{-1} / \mathrm{Vol}(V) \leq 2/k. \tag{1}$$

Let $e \in E$. Define $X_i = 1$ if e participates in the ith RandomNibble(G, ϕ), and define $X_i = 0$ otherwise. Set $X = \sum_{i=1}^{k} X_i$. By Equation (1), we infer that $\mathbf{E}[X] \leq 2$. By a Chernoff bound, $\mathbf{Pr}[X > w] \leq \exp(-2(w-2)/3) \lll (\mathrm{Vol}(V))^{-2}$. By a union bound over all edges $e \in E$, we infer that $\mathbf{Pr}[B] < (\mathrm{Vol}(V))^{-1} \lll \frac{k}{40w\,\mathrm{Vol}(V)}$, as required. □

Intuitively, Lemma 13 shows that we only lose a factor of $O(\log n)$ in conductance if we combine the results of k parallel executions of RandomNibble(G, ϕ). We are now ready to present the algorithm for finding a nearly most balanced sparse cut, which involves executing ParallelNibble sequentially for $s = O(\mathrm{poly}(1/\phi, \log n))$ times on the remaining subgraph.

```
Algorithm $(G, \phi, p)$
Initialize $W_0 = V$. For $i = 1$ to $s \overset{\mathrm{def}}{=} 4g(\phi, \mathrm{Vol}(V)) \lceil \log_{7/4}(1/p) \rceil$ do the following.
    (1) Execute ParallelNibble$(G\{W_{i-1}\}, \phi)$. Let the output be $C_i$.
    (2) Set $W_i = W_{i-1} \setminus C_i$.
    (3) If $\mathrm{Vol}(W_i) \leq (47/48)\,\mathrm{Vol}(V)$ or $i = s$, then return $C = \bigcup_{j=1}^{i} C_j$ and quit.
```

Lemma 14 (Analysis of Partition). *Let C be the output of* Partition(G, ϕ), *with* $0 < \phi \leq 1/12$. *Then the following holds:*

(1) $\mathrm{Vol}(C) \leq (47/48)\,\mathrm{Vol}(V)$.
(2) *If $C \neq \emptyset$, then* $\Phi(C) = O(\phi \log |V|)$.
(3) *Furthermore, for each subset $S \subset V$ satisfying*

$$\mathrm{Vol}(S) \leq \frac{1}{2} \cdot \mathrm{Vol}(V) \quad \textit{and} \quad \Phi(S) \leq f(\phi),$$

with probability at least $1 - p$, *at least one of the following holds:*
(a) $\mathrm{Vol}(C) \geq (1/48)\,\mathrm{Vol}(V)$.
(b) $\mathrm{Vol}(S \cap C) \geq (1/2)\,\mathrm{Vol}(S)$.

Proof. This proof follows the approach of the proof of Reference [62, Theorem 3.3].

Proof of Condition 1. Let i' be the index such that the output subset C is $\bigcup_{j=1}^{i'} C_j$. Then, we have $\mathrm{Vol}(C) \leq \mathrm{Vol}(V \setminus W_{i'-1}) + \mathrm{Vol}(C_{i'})$. Since the algorithm does not terminate at the $(i'-1)$th iteration, we have $\mathrm{Vol}(W_{i'-1}) > (47/48)\,\mathrm{Vol}(V)$, and so $\mathrm{Vol}(V \setminus W_{i'-1}) \leq (1/48)\,\mathrm{Vol}(V)$. By the algorithm description of ParallelNibble, we have $\mathrm{Vol}(C_{i'}) \leq (23/24)\,\mathrm{Vol}(W_{i'-1}) \leq (23/24)\,\mathrm{Vol}(V)$. To summarize, we have $\mathrm{Vol}(C) \leq (1/48)\,\mathrm{Vol}(V) + (23/24)\,\mathrm{Vol}(V) = (47/48)\,\mathrm{Vol}(V)$.

Proof of Condition 2. Note that the sets $C_1, \ldots, C_{i'}$ that constitute $C = \bigcup_{j=1}^{i'} C_j$ are disjoint vertex sets. We have $|\partial(C)| \leq \sum_{j=1}^{i'} |\partial(C_i)| \leq O(\phi \log |V|) \sum_{j=1}^{i'} \mathrm{Vol}(C_i) = O(\phi \log |V|) \cdot \mathrm{Vol}(C)$, where the second inequality is due to Lemma 13. By Condition 1, we infer that $\mathrm{Vol}(V \setminus C) \geq (1/47)\,\mathrm{Vol}(C)$, and so we also have $|\partial(C)| \leq O(\phi \log |V|) \cdot \mathrm{Vol}(V \setminus C)$. Hence, $\Phi(C) = O(\phi \log |V|)$.

Proof of Condition 3. We focus on $h \overset{\mathrm{def}}{=} 4g(\phi, \mathrm{Vol}(V))$ consecutive iterations from $i = x + 1$ to $i = x + h$, for some index x. For each index $j \in [1, h]$, we write H_j to denote the event that (i) $\mathrm{Vol}(S \cap W_{x+j-1}) \leq \mathrm{Vol}(S)/2$ or (ii) the algorithm ends prior to iteration $i = x + j$. We define the random variable Y_j as follows:

$$Y_j = \begin{cases} \frac{\mathrm{Vol}(S)}{2g(\phi, \mathrm{Vol}(V))} & \text{if } H_j \text{ occurs (Case 1)} \\ \mathrm{Vol}(W_{x+j-1} \cap S) & \text{if } \mathrm{Vol}(C_{x+j}) \geq (1/24)\,\mathrm{Vol}(W_{x+j-1}) \text{ (Case 2)} \\ \mathrm{Vol}(C_{x+j} \cap S) & \text{otherwise (Case 3).} \end{cases}$$

We claim that if H_j does not occur, then the preconditions of Lemma 13 are met for the cut $S' = S \cap W_{x+j-1}$ in the graph $G' = G\{W_{x+j-1}\}$ when we run $\texttt{ParallelNibble}(G\{W_{x+j-1}\}, \phi)$ during the $(x + j)$th iteration.

- We show that $\mathrm{Vol}(S \cap W_{x+j-1}) \leq (2/3)\,\mathrm{Vol}(W_{x+j-1})$, as follows:

$$\begin{aligned} \mathrm{Vol}(S \cap W_{x+j-1}) &\leq (1/2)\,\mathrm{Vol}(V) && (\text{since } \mathrm{Vol}(S) \leq \mathrm{Vol}(V)/2) \\ &< (1/2)(48/47)\,\mathrm{Vol}(W_{x+j-1}) && \big(\text{since } \mathrm{Vol}(W_{x+j-1}) > (47/48)\,\mathrm{Vol}(V)\big) \\ &< (2/3)\,\mathrm{Vol}(W_{x+j-1}). \end{aligned}$$

- We show that $\Phi_{G\{W_{x+j-1}\}}(S) \leq 2\Phi(S) \leq 2f(\phi, \mathrm{Vol}(V)) \leq 2f(\phi, \mathrm{Vol}(W_{x+j-1}))$, where we write $f(\phi, r)$ to indicate the value of $f(\phi)$ when the underlying graph has volume r:

$$\begin{aligned} \Phi_{G\{W_{x+j-1}\}}(S) &= \frac{|E(S \cap W_{x+j-1}, W_{x+j-1} \setminus S)|}{\min\{\mathrm{Vol}(S \cap W_{x+j-1}), \mathrm{Vol}(W_{x+j-1} \setminus S)\}} \\ &\leq \frac{|E(S, V \setminus S)|}{\min\{\mathrm{Vol}(S \cap W_{x+j-1}), \mathrm{Vol}(W_{x+j-1} \setminus S)\}} \\ &< \frac{|E(S, V \setminus S)|}{(1/2)\min\{\mathrm{Vol}(S), \mathrm{Vol}(V \setminus S)\}} \\ &= 2\Phi(S). \end{aligned}$$

  The second inequality is explained as follows. We have $\mathrm{Vol}(S \cap W_{x+j-1}) > \mathrm{Vol}(S)/2$, since H_j does not occur, and we also have

$$\begin{aligned} \mathrm{Vol}(W_{x+j-1} \setminus S) &\geq \mathrm{Vol}(V \setminus S) - (1/48)\,\mathrm{Vol}(V) \\ &\geq \mathrm{Vol}(V \setminus S) - (1/24)\,\mathrm{Vol}(V \setminus S) && (\text{since } \mathrm{Vol}(V \setminus S) \geq (1/2)\,\mathrm{Vol}(V)) \\ &> \mathrm{Vol}(V \setminus S)/2. \end{aligned}$$

Thus, we are able to use Lemma 13 to infer that that

$$\mathbf{E}[Y_j \mid \overline{H_j}] \geq \frac{\mathrm{Vol}(S \cap W_{x+j-1})}{g(\phi, \mathrm{Vol}(W_{x+j-1}))} > \frac{\mathrm{Vol}(S)}{2g(\phi, \mathrm{Vol}(V))}.$$

In the calculation, we use the two inequalities $g(\phi, \mathrm{Vol}(V)) \geq g(\phi, \mathrm{Vol}(W_{x+j-1}))$ and $\mathrm{Vol}(S \cap W_{x+j-1}) > \mathrm{Vol}(S)/2$, where the latter is due to $\overline{H_j}$. Combining $\mathbf{E}[Y_j \mid \overline{H_j}] > \frac{\mathrm{Vol}(S)}{2g(\phi, \mathrm{Vol}(V))}$ with the trivial bound $\mathbf{E}[Y_j \mid H_j] = \frac{\mathrm{Vol}(S)}{2g(\phi, \mathrm{Vol}(V))}$, we conclude that

$$\mathbf{E}[Y_j] \geq \frac{\mathrm{Vol}(S)}{2g(\phi, \mathrm{Vol}(V))}.$$

We write $Y = \sum_{j=1}^{h} Y_j$, and we have $\mathbf{E}[Y] \geq 2\operatorname{Vol}(S)$ in view of the above, as we recall $h = 4g(\phi, \operatorname{Vol}(V))$. We claim that we always have $Y \leq 4\operatorname{Vol}(S)$. We may write $Y = Y^1 + Y^2 + Y^3$, where Y^i considers the part of Y due to Case i in the definition of Y_j. It is clear that $Y^1 \leq h \cdot \frac{\operatorname{Vol}(S)}{2g(\phi, \operatorname{Vol}(V))} = 2\operatorname{Vol}(S)$. We claim that $Y^2 \leq \operatorname{Vol}(S)$ by observing that Case 2 can only occur at most once. Suppose Case 2 occurs at iteration $i = x + j$. Then $\operatorname{Vol}(C_{x+j}) \geq (1/24)\operatorname{Vol}(W_{x+j-1}) > (1/48)\operatorname{Vol}(W_{x+j-1})$, which implies $\operatorname{Vol}(W_{x+j}) \leq (47/48)\operatorname{Vol}(W_{x+j-1}) \leq (47/48)\operatorname{Vol}(V)$, and so the algorithm terminates. For Case 3, we have $Y^3 \leq \sum_{j=1}^{h} \operatorname{Vol}(C_{x+j} \cap S) \leq \operatorname{Vol}(S)$. In view of the above, we have

$$(\operatorname{Vol}(S)/2)\mathbf{Pr}[Y < (1/2)\operatorname{Vol}(S)] + 4\operatorname{Vol}(S)(1 - \mathbf{Pr}[Y < (1/2)\operatorname{Vol}(S)]) \geq \mathbf{E}[Y] \geq 2\operatorname{Vol}(S),$$

and this implies $\mathbf{Pr}[Y < (1/2)\operatorname{Vol}(S)] \leq 4/7$. We argue that $Y \geq (1/2)\operatorname{Vol}(S)$ implies that either Condition 3a or Condition 3b holds. If Case 1 ever occurs, then the algorithm terminates before the last iteration $i = s$, and so we must have $\operatorname{Vol}(C) \geq (1/48)\operatorname{Vol}(V)$. Similarly, if Case 2 ever occurs, we automatically have $\operatorname{Vol}(C) \geq (1/48)\operatorname{Vol}(V)$. If Cases 1 and 2 never occur for all $j \in [1, h]$, then we have $\operatorname{Vol}(C \cap S) \geq Y > (1/2)\operatorname{Vol}(S)$.

We divide all s iterations into $\lceil \log_{7/4}(1/p) \rceil$ intervals of length $h = 4g(\phi, \operatorname{Vol}(V))$, and apply the above analysis to each of them. We conclude that with probability at least $1 - (4/7)^{\lceil \log_{7/4}(1/p) \rceil} \geq 1 - p$, there is at least one interval satisfying $Y \geq (1/2)\operatorname{Vol}(S)$. In other words, with probability at least $1 - p$, either Condition 3a or Condition 3b holds. □

4.5 Distributed Implementation

In this section, we show that the algorithm `Partition`(G, ϕ) can be implemented to run in $O(D \cdot \operatorname{poly}(\log n, 1/\phi))$ rounds in CONGEST. We do not make effort in optimizing the round complexity.

Notation. In this section, the input graph $G = (V, E)$ is often a *subgraph* of the underlying communication network (i.e., the input graph), and so $|V|$ may be much smaller than n, the number of vertices in the input graph. Nonetheless, we express the round complexity in terms of n, since the allowable error probability is $1/\operatorname{poly}(n)$, independent of $|V|$. Furthermore, we may be able to broadcast information to all vertices in G by communicating along edges in some larger network G'. The parameter D refers to the diameter of G', *not* G. (The decompositions from Section 3.2 endow each subgraph G with a Steiner tree that uses edges outside of G, with the guarantee that the congestion along any edge is $O(\log n)$.)

Lemma 15 (Implementation of `ApproximateNibble`). *Suppose v initially knows that it is the starting vertex. The algorithm* `ApproximateNibble`(G, v, ϕ, b) *can be implemented to run in $O(\frac{\log^4 n}{\phi^5})$ rounds. Only the edges in P^* participate in the computation. By the end of the algorithm, each vertex u knows whether or not $u \in C$, w.h.p.*

Proof. First, the calculation of $\tilde{p}_t^v(u)$ and $\tilde{\varrho}_t^v(u)$ for each $0 \leq t \leq t_0$ for each vertex $u \in V$ can be done in $t_0 = O(\frac{\log n}{\phi^2})$ rounds.

Next, we have to go over all $O(\frac{\log n}{\phi})$ choices of x and all $O(\frac{\log n}{\phi^2})$ choices of t to see if there is a pair (t, j_x) meeting the required four conditions. More specifically, given t and x, our task is the following.

Search for j_x and $\tilde{\pi}_t^v(1, \ldots, j_x)$. Once $j_{x-1} < j_{\max}$ is computed, j_x is defined to be $\max\{j_{x-1} + 1, j^*\}$, where

$$j^* = \arg\max_{1 \leq j \leq j_{\max}} (\operatorname{Vol}(\tilde{\pi}_t^v(1 \ldots j)) \leq (1 + \phi)\operatorname{Vol}(\tilde{\pi}_t^v(1 \ldots j_{x-1}))).$$

We then need to compute the set $\tilde{\pi}_t^v(1,\ldots,j_x)$. This can be done in $O(t_0 \log n) = O(\frac{\log^2 n}{\phi^2})$ rounds via a "random binary search" on the vertex set U containing all vertices u with $\tilde{p}_t^v(u) > 0$. For the sake of presentation, we rank all vertices $u_1,\ldots,u_{|U|}$ by the ordering $\tilde{\pi}_t^v$. Note that each u_i does not know its rank i, and we cannot afford to compute the rank of all vertices in U.

We maintain two indices L and R that control the search space. Initially, $L \leftarrow 1$ and $R \leftarrow j_{\max}$. In each iteration, we pick one vertex u_i from $\{u_L,\ldots,u_R\}$ uniformly at random, and calculate $\mathrm{Vol}(\tilde{\pi}_t^v(1,\ldots,i))$. This can be done in $O(t_0)$ rounds. More specifically, we build a spanning tree T of the edge set P^* rooted at v, and use only this tree for communication. It is clear that the subgraph induced by P^* is connected and has diameter $O(t_0)$. To sample a vertex from the set $\{u_L,\ldots,u_R\}$ uniformly at random, we first do a bottom-up traversal to let each vertex u in the tree compute the number of vertices in $\{u_L,\ldots,u_R\}$ that are within the subtree rooted at u. Using this information, we can sample one vertex from $\{u_L,\ldots,u_R\}$ uniformly at random by a top-down traversal.

If $\mathrm{Vol}(\tilde{\pi}_t^v(1,\ldots,i)) < (1+\phi)\,\mathrm{Vol}(\tilde{\pi}_t^v(1\ldots j_{x-1}))$, then we update $L \leftarrow i$; if $\mathrm{Vol}(\tilde{\pi}_t^v(1,\ldots,i)) = (1+\phi)\,\mathrm{Vol}(\tilde{\pi}_t^v(1\ldots j_{x-1}))$, we update $L \leftarrow i$ and $R \leftarrow i$; otherwise, we update $R \leftarrow i-1$. We are done when we reach $L = R$.

In each iteration, with probability $1/2$ the rank of the vertex we sample lies in the middle half of $[L,R]$, and so the size of search space $[L,R]$ is reduced by a factor of at least $3/4$. Thus, within $O(\log n)$ iterations, we have $L = R$, and $S_j(q_t) = \{u_1,\ldots,u_j\}$ with $j = L = R$. The round complexity of this procedure is $O(t_0 \log n) = O(\frac{\log^2 n}{\phi^2})$.

Checking (C.1)–(C.3) or (C.1)–(C.3*).* Given the index j_x and the subset $\tilde{\pi}_t^v(1,\ldots,j_x))$, it is straightforward to check whether these conditions are met in $O(t_0)$ rounds.

Round Complexity. To summarize, we go over all $O(\frac{\log n}{\phi})$ choices of x and all $O(\frac{\log n}{\phi^2})$ choices of t, and for each pair (t,x), we have to spend $O(\frac{\log^2 n}{\phi^2})$ rounds. Therefore, the total round complexity is $O(\frac{\log^4 n}{\phi^5})$. □

Lemma 16 (Implementation of `ParallelNibble`). *The algorithm* `ParallelNibble`(G,ϕ) *can be implemented to run in* $O(D\log n + \frac{\log^5 n}{\phi^5})$ *rounds in* CONGEST.

Proof. The implementation of `ParallelNibble`(G,ϕ) has three parts.

Generation of `ApproximateNibble` *Instances.* The first part is to generate all k instances of `ApproximateNibble`(G,v,ϕ,b), where the starting vertex $v \sim \psi_V$ is sampled according to the degree distribution, and $b \in [1,\ell]$ is sampled with $\mathbf{Pr}[b=i] = 2^{-i}/(1-2^{-\ell})$.

This task can be solved in $O(D+\log n)$ rounds, as follows. We build a BFS tree rooted at an arbitrary vertex x. For each vertex v, define $s(v)$ as the sum of $\deg(u)$ 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.

We let the root vertex x sample the parameter b for all k instances of `ApproximateNibble`. Denote K_i as the number of instances with $b = i$. At the beginning, the root x stores K_i amount of i-tokens. Let $L = \Theta(D)$ be the number of layers in the BFS tree. For $j = 1,\ldots,L$, the vertices of layer j do the following. When an i-token arrives at v, the i-token disappears at v with probability $\deg(v)/s(v)$ and v locally generates an instance of `ApproximateNibble` with starting vertex v and parameter $b = i$; otherwise, v passes the i-token to a child u with probability $\frac{s(u)}{s(v)-\deg(v)}$. Though v might need to send a large amount of i-tokens to u, the only information v needs to let u know is the number of i-tokens. Thus, for each i, the generation of all K_i instances of `ApproximateNibble` with a random starting vertex can be done in L rounds. Using pipelining, we can do this for all i in $O(D + \log n)$ rounds, independent of k.

Simultaneous Execution of `ApproximateNibble`. The second part is to run all k instances of `ApproximateNibble` simultaneously. If there is an edge e participating in more than $w = O(\log n)$ of them, then the two endpoints of e broadcast a special message $\star$ to everyone else to notify them to terminate the algorithm with $C = \emptyset$, and the broadcasting takes D rounds. Otherwise, this task can be done in $O(\log n) \cdot O(\frac{\log^4 n}{\phi^5}) = O(\frac{\log^5 n}{\phi^5})$ rounds in view of Lemma 15. Overall, the round complexity is $O(D + \frac{\log^5 n}{\phi^5})$.

Selection of i^* and $C = U_{i^*}$. In the description of the algorithm `ParallelNibble`(G, ϕ), we assume that all `ApproximateNibble` instances are indexed from 1 to k. However, in a distributed implementation, we cannot afford to do this. What we can do is to let the starting vertex v of each `ApproximateNibble` instance locally generate a random $O(\log n)$-bit identifier associated with the `ApproximateNibble` instance. We say that an `ApproximateNibble` instance is the ith instance if its identifier is ranked ith in the increasing order of all k identifiers. With these identifiers, we can now use a random binary search to find i^* and calculate $C = U_{i^*}$ in $O(D \log n)$ rounds w.h.p.

Round Complexity. To summarize, the round complexity for the three parts are $O(D + \log n)$, $O(D + \frac{\log^5 n}{\phi^5})$, and $O(D \log n)$. Thus, the total round complexity is $O(D \log n + \frac{\log^5 n}{\phi^5})$. □

Lemma 17 (Implementation of `Partition`). *The algorithm* `Partition`(G, ϕ, p) *with* $p = 1/\text{poly}(n)$ *can be implemented to run in* $O(\frac{D \log^7 n}{\phi^5} + \frac{\log^{11} n}{\phi^{10}})$ *rounds in* CONGEST.

Proof. This lemma follows immediately from Lemma 16, as `Partition`(G, ϕ) consists of

$$s = O\left(g(\phi, \text{Vol}(V)) \log(1/p)\right) = O\left(\frac{\log^6 n}{\phi^5}\right)$$

iterations of `ParallelNibble` (with $p = 1/\text{poly}(n)$), where each of them costs $O(D \log n + \frac{\log^5 n}{\phi^5})$ rounds (Lemma 16), and so the total round complexity is $O(\frac{D \log^7 n}{\phi^5} + \frac{\log^{11} n}{\phi^{10}})$. □

Now, we are ready to prove Theorem 6.

Proof of Theorem 6. The theorem follows from a re-parameterization of Lemma 14 (properties of the partition) and Lemma 17 (runtime of the algorithm). □

5 CONCLUSION

In this article, we designed an efficient distributed expander decomposition algorithm, and applied it to the basic problem of detecting, counting, or enumerating triangles in $\tilde{O}(n^{1/3})$ rounds. Our Triangle Enumeration algorithm is *optimal*, up to polylogarithmic factors [32]. To our knowledge, this is the only known non-trivial problem whose CONGEST and CONGESTED-CLIQUE complexities (as a function of n) are essentially the same [17, 32].

5.1 Recent Developments

After the initial publication of this work in Reference [11, 12], some additional applications of distributed expander decompositions have been discovered. Daga et al. [16] used our distributed expander decomposition to obtain the first algorithm for computing the *exact* edge connectivity of a graph using sublinear number of rounds.

Eden et al. [19] demonstrated that distributed expander decompositions can be useful for subgraph enumeration beyond triangles. Distributed expander decompositions were used to enumerate all 4-cliques in $\tilde{O}(n^{5/6})$ rounds and all 5-cliques in $\tilde{O}(n^{21/22})$ rounds. For any k-vertex subgraph H, detecting whether a copy of H exists can be done in $n^{2-\Omega(1/k)}$ rounds, nearly matching the $n^{2-O(1/k)}$ lower bound in Reference [24]. Moreover, there is some fixed constant $\delta \in (0, 1/2)$ such that for any k, an $\Omega(n^{1/2+\delta})$ CONGEST lower bound on $2k$-cycle detection implies a new circuit lower bound.

Censor-Hillel, Chang, Le Gall, and Leitersdorf [7, 8] improved the clique enumeration algorithms of Eden et al. [19], showing that k-cliques can be enumerated in $\tilde{O}(n^{(k-2)/k})$ rounds for all $k \geq 4$, which is optimal up to a polylogarithmic factor.

Our $\tilde{O}(n^{1/3})$-round Triangle Enumeration algorithm is clearly *not* optimal when $\Delta \ll n^{1/3}$, as the trivial algorithm can list all triangles in $O(\Delta)$ rounds. Huang, Pettie, Zhang, and Zhang [30] designed a *Local* Triangle Enumeration algorithm[4] in $O(\Delta/\log n + \log\log \Delta)$ rounds with high probability. This matches the $\Omega(\Delta/\log n)$ lower bound of Izumi and Le Gall [32] whenever $\Delta > \log n \log\log\log n$.

Chang and Saranurak [13] recently considered the following *relaxed* variant of (ϵ, ϕ)-expander decomposition: partition the edges $E = E_1 \cup E_2 \cup \cdots \cup E_x \cup E^*$ in such a way that $|E^*| \leq \epsilon|E|$ and the subgraphs $G[E_1], G[E_2], \ldots, G[E_x]$ are vertex-disjoint and have conductance at least ϕ. They designed a randomized algorithm for computing such an expander decomposition with $\phi = 1/\text{poly}(\epsilon^{-1}, \log n)$ in $\text{poly}(\epsilon^{-1}, \log n)$ rounds and a *deterministic* algorithm computing the decomposition with $\phi = \text{poly}(\epsilon)2^{-O(\sqrt{\log n \log\log n})}$ in $\text{poly}(\epsilon^{-1})2^{O(\sqrt{\log n \log\log n})}$ rounds.

5.2 Open Problems

Many interesting problems are left open. In particular, the lower and upper bounds on Triangle Enumeration differ by a $\log^c n$ factor, but c is *huge*, stemming from inefficiencies in (i) the hierarchical routing structure of Reference [27] as modified in Section 2.1, and (ii) our expander decomposition algorithm. Improving the current state of the art of (i) and (ii) will lead to an improved upper bound for Triangle Enumeration, as well as several other problems [16, 19, 27, 28].

The graph underlying the $\Omega(n^{1/3}/\log n)$ lower bound [32, 52] for Triangle Enumeration is the Erdős-Rényi random graph $\mathcal{G}(n, p)$ with $p = 1/2$. Hence, this lower bound does not rule out the possibility of an $n^{1/3-\Omega(1)}$-round CONGEST algorithm for Triangle Enumeration on *sparse* graphs (i.e., $m = o(n^2)$) or Triangle *Detection* on any class of graphs. In the CONGESTED-CLIQUE model, efficient algorithms for these two problems are already known: Triangle Detection can be solved in $n^{1-(2/\omega)+o(1)} = o(n^{0.158})$ rounds [9], and Triangle Enumeration on m-edge graphs can be solved in $O(1 + m/n^{5/3})$ rounds [10, 52]. If *quantum bits* are allowed, then, as Izumi, Le Gall, and Magniez [31] showed, Triangle Detection can be solved in $\tilde{O}(n^{1/4})$ rounds in CONGEST.

REFERENCES

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

[2] Udit Agarwal, Vijaya Ramachandran, Valerie King, and Matteo Pontecorvi. 2018. A deterministic distributed algorithm for exact weighted all-pairs shortest paths in $\tilde{O}(n^{3/2})$ rounds. In *Proceedings of the 38th ACM Symposium on Principles of Distributed Computing (PODC'18)*. 199–205.

[3] Vedat Levi Alev, Nima Anari, Lap Chi Lau, and Shayan Oveis Gharan. 2018. Graph clustering using effective resistance. In *Proceedings of the 9th Innovations in Theoretical Computer Science Conference (ITCS'18)*. 41:1–41:16.

[4] Sanjeev Arora, Boaz Barak, and David Steurer. 2015. Subexponential algorithms for unique games and related problems. *J. ACM* 62, 5, Article 42 (2015), 42:1–42:25 pages.

[4]Recall that in this problem, every triangle is reported by one of its three constituent vertices.

[5] Sanjeev Arora, Satish Rao, and Umesh Vazirani. 2009. Expander flows, geometric embeddings and graph partitioning. *J. ACM* 56, 2, Article 5 (2009), 5:1–5:37 pages.

[6] Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. 1994. Low-diameter graph decomposition is in NC. *Random Struct. Algor.* 5, 3 (1994), 441–452.

[7] Keren Censor-Hillel, Yi-Jun Chang, François Le Gall, and Dean Leitersdorf. 2021. Tight distributed listing of cliques. In *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'21).*

[8] Keren Censor-Hillel, François Le Gall, and Dean Leitersdorf. 2020. On distributed listing of cliques. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC'20).* 474–482.

[9] Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. 2019. Algebraic methods in the congested clique. *Distrib. Comput.* 32, 6 (2019), 461–478.

[10] Keren Censor-Hillel, Dean Leitersdorf, and Elia Turner. 2018. Sparse matrix multiplication and triangle listing in the congested clique model. In *Proceedings of the 22nd International Conference on Principles of Distributed Systems (OPODIS'18)*, Vol. 125. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 4:1–4:17.

[11] Yi-Jun. Chang, Seth Pettie, and Hengjie Zhang. 2019. Distributed triangle detection via expander decomposition. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'19).* 821–840.

[12] Yi-Jun Chang and Thatchaphol Saranurak. 2019. Improved distributed expander decomposition and nearly optimal triangle enumeration. In *Proceedings of the 39th ACM Symposium on Principles of Distributed Computing (PODC'19).* 66–73.

[13] Yi-Jun Chang and Thatchaphol Saranurak. 2020. Deterministic distributed expander decomposition and routing with applications in distributed derandomization. In *Proceedings of the 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS'20).*

[14] Timothy Chu, Yu Gao, Richard Peng, Sushant Sachdeva, Saurabh Sawlani, and Junxing Wang. 2018. Graph sparsification, spectral sketches, and faster resistance computation, via short cycle decompositions. In *Proceedings of the 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS'18).* 361–372.

[15] Michael B. Cohen, Jonathan A. Kelner, John Peebles, Richard Peng, Anup B. Rao, Aaron Sidford, and Adrian Vladu. 2017. Almost-linear-time algorithms for Markov chains and new spectral primitives for directed graphs. In *Proceedings of the 49th Annual ACM Symposium on Theory of Computing (STOC'17).* 410–419.

[16] Mohit Daga, Monika Henzinger, Danupon Nanongkai, and Thatchaphol Saranurak. 2019. Distributed edge connectivity in sublinear time. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC'19).* 343–354.

[17] Danny Dolev, Christoph Lenzen, and Shir Peled. 2012. "Tri, tri again": Finding triangles and small subgraphs in a distributed setting. In *Proceedings of the 26th International Symposium on Distributed Computing (DISC'12).* 195–209.

[18] Andrew Drucker, Fabian Kuhn, and Rotem Oshman. 2014. On the power of the congested clique model. In *Proceedings of the 33rd ACM Symposium on Principles of Distributed Computing (PODC'14).* 367–376.

[19] Talya Eden, Nimrod Fiat, Orr Fischer, Fabian Kuhn, and Rotem Oshman. 2019. Sublinear-time distributed algorithms for detecting small cliques and eEven cycles. In *Proceedings of the International Symposium on Distributed Computing (DISC'19).* 15:1–15:16.

[20] Michael Elkin. 2006. An unconditional lower bound on the time-approximation trade-off for the distributed minimum spanning tree problem. *SIAM J. Comput.* 36, 2 (2006), 433–456.

[21] Michael Elkin. 2017. Distributed exact shortest paths in sublinear time. In *Proceedings of the 49th Annual ACM Symposium on Theory of Computing (STOC'17).* 757–770.

[22] Michael Elkin. 2017. A simple deterministic distributed MST algorithm, with near-optimal time and message complexities. In *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing (PODC'17).* 157–163.

[23] Michael Elkin and Ofer Neiman. 2016. Distributed strong diameter network decomposition. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC'16).* 211–216.

[24] Orr Fischer, Tzlil Gonen, Fabian Kuhn, and Rotem Oshman. 2018. Possibilities and impossibilities for distributed subgraph detection. In *Proceedings of the 30th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'18).* 153–162.

[25] Mohsen Ghaffari and Bernhard Haeupler. 2016. Distributed algorithms for planar networks I: Planar embedding. In *Proceedings of the 36th ACM Symposium on Principles of Distributed Computing (PODC'16).* 29–38.

[26] Mohsen Ghaffari and Bernhard Haeupler. 2016. Distributed algorithms for planar networks II: Low-congestion shortcuts, MST, and min-cut. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'16).* 202–219.

[27] Mohsen Ghaffari, Fabian Kuhn, and Hsin-Hao Su. 2017. Distributed MST and routing in almost mixing time. In *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing (PODC'17).* 131–140.

[28] Mohsen Ghaffari and Jason Li. 2018. New distributed algorithms in almost mixing time via transformations from parallel algorithms. In *Proceedings of the 32nd International Symposium on Distributed Computing (DISC'18)*, Vol. 121. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 31:1–31:16.

[29] Chien-Chung Huang, Danupon Nanongkai, and Thatchaphol Saranurak. 2017. Distributed exact weighted all-pairs shortest paths in $\tilde{O}(n^{5/4})$ Rounds. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS'17)*. 168–179.

[30] Dawei Huang, Seth Pettie, Yixiang Zhang, and Zhijun Zhang. 2020. The communication complexity of set intersection and multiple equality testing. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'20)*. 1715–1732.

[31] Taisuke Izumi, François Le Gall, and Frédéric Magniez. 2020. Quantum distributed algorithm for triangle finding in the CONGEST Model. In *Proceedings of the 37th International Symposium on Theoretical Aspects of Computer Science (STACS'20)*, Vol. 154. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 23:1–23:13.

[32] Taisuke Izumi and François Le Gall. 2017. Triangle finding and listing in CONGEST networks. In *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing (PODC'17)*. 381–389.

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

[34] Tomasz Jurdziński and Krzysztof Nowicki. 2018. MST in $O(1)$ rounds of congested clique. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'18)*. 2620–2632.

[35] Ravi Kannan, Santosh Vempala, and Adrian Vetta. 2004. On clusterings: Good, bad and spectral. *J. ACM* 51, 3 (2004), 497–515.

[36] Ken-Ichi Kawarabayashi and Mikkel Thorup. 2018. Deterministic edge connectivity in near-linear time. *J. ACM* 66, 1, Article 4 (2018), 4:1–4:50 pages.

[37] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. 2014. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'14)*. 217–226.

[38] Sebastian Krinninger and Danupon Nanongkai. 2018. A faster distributed single-source shortest paths algorithm. In *Proceedings of the 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS'18)*. 686–697.

[39] J. Kuczynski and Henryk Wozniakowski. 1992. Estimating the largest eigenvalue by the power and Lanczos algorithms with a random start. *SIAM J. Matrix Anal. Appl.* 13, 4 (1992), 1094–1122.

[40] Fabian Kuhn and Anisur Rahaman Molla. 2015. Distributed sparse cut approximation. In *Proceedings of the 19th International Conference on Principles of Distributed Systems (OPODIS'15)*. 10:1–10:14.

[41] Shay Kutten and David Peleg. 1998. Fast distributed construction of small k-dominating sets and applications. *J. Algor.* 28, 1 (1998), 40–66.

[42] Christoph Lenzen. 2013. Optimal deterministic routing and sorting on the congested clique. In *Proceedings 33rd ACM Symposium on Principles of Distributed Computing (PODC'13)*. 42–50.

[43] Nathan Linial and Michael Saks. 1993. Low diameter graph decompositions. *Combinatorica* 13, 4 (1993), 441–454.

[44] László Lovász and Miklós Simonovits. 1990. The mixing rate of Markov chains, an isoperimetric inequality, and computing the volume. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science (FOCS'90)*. 346–354.

[45] David W. Matula and Farhad Shahrokhi. 1990. Sparsest cuts and bottlenecks in graphs. *Discrete Appl. Math.* 27, 1–2 (1990), 113–123.

[46] Gary L. Miller, Richard Peng, and Shen Chen Xu. 2013. Parallel graph decompositions using random shifts. In *Proceedings of the 25th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'13)*. 196–203.

[47] Danupon Nanongkai and Thatchaphol Saranurak. 2017. Dynamic spanning forest with worst-case update time: Adaptive, Las Vegas, and $O(n^{1/2-\epsilon})$-time. In *Proceedings of the 49th Annual ACM Symposium on Theory of Computing (STOC'17)*. 1122–1129.

[48] Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. 2017. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS'17)*. 950–961.

[49] Krzysztof Nowicki. 2019. A deterministic algorithm for the MST problem in constant rounds of congested clique. Retrieved from https://arXiv:1912.04239.

[50] Lorenzo Orecchia and Nisheeth K. Vishnoi. 2011. Towards an SDP-based approach to spectral methods: A nearly-linear-time algorithm for graph partitioning and decomposition. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'11)*. 532–545.

[51] Lorenzo Orecchia and Zeyuan Allen Zhu. 2014. Flow-based algorithms for local graph clustering. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'14)*. 1267–1286.

[52] Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. 2018. On the distributed complexity of large-scale graph computations. In *Proceedings of the 30th ACM Symposium on Parallelism in Algorithms and Architecture (SPAA'18)*. 405–414.

[53] M. Pătraşcu and M. Thorup. 2007. Planning for fast connectivity updates. In *Proceedings of the 48th IEEE Symposium on Foundations of Computer Science (FOCS'07)*. 263–271.

[54] David Peleg and Vitaly Rubinovich. 2000. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM J. Comput.* 30, 5 (2000), 1427–1442.

[55] Prasad Raghavendra and David Steurer. 2010. Graph expansion and the unique games conjecture. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC'10)*. 755–764.

[56] Václav Rozhoň and Mohsen Ghaffari. 2020. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC'20)*. 350–363.

[57] Thatchaphol Saranurak and Di Wang. 2019. Expander decomposition and pruning: Faster, stronger, and simpler. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'19)*. 2616–2635.

[58] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. 2012. Distributed verification and hardness of distributed approximation. *SIAM J. Comput.* 41, 5 (2012), 1235–1265.

[59] Atish Das Sarma, Anisur Rahaman Molla, and Gopal Pandurangan. 2015. Distributed computation of sparse cuts via random walks. In *Proceedings of the 16th International Conference on Distributed Computing and Networking (ICDCN'15)*. 6:1–6:10.

[60] Alistair Sinclair and Mark Jerrum. 1989. Approximate counting, uniform generation and rapidly mixing Markov chains. *Info. Comput.* 82, 1 (1989), 93–133.

[61] Daniel A. Spielman and Nikhil Srivastava. 2008. Graph sparsification by effective resistances. In *Proceedings of the 40th ACM Symposium on Theory of Computing (STOC'08)*. 563–568.

[62] Daniel A. Spielman and Shang-Hua Teng. 2004. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC'04)*. 81–90.

[63] Daniel A. Spielman and Shang-Hua Teng. 2011. Spectral sparsification of graphs. *SIAM J. Comput.* 40, 4 (2011), 981–1025.

[64] Daniel A. Spielman and Shang-Hua Teng. 2013. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM J. Comput.* 42, 1 (2013), 1–26.

[65] Daniel A. Spielman and Shang-Hua Teng. 2014. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM J. Matrix Anal. Appl.* 35, 3 (2014), 835–885.

[66] Luca Trevisan. 2008. Approximation algorithms for unique games. *Theory Comput.* 4, 5 (2008), 111–128.

[67] Christian Wulff-Nilsen. 2017. Fully-dynamic minimum spanning forest with improved worst-case update time. In *Proceedings of the 49th Annual ACM Symposium on Theory of Computing (STOC'17)*. 1130–1143.

Received November 2019; revised September 2020; accepted December 2020