Deterministic Distributed Expander Decomposition and Routing with Applications in Distributed Derandomization

Yi-Jun Chang
ETH Institute for Theoretical Studies
Switzerland
yi-jun.chang@eth-its.ethz.ch

Thatchaphol Saranurak
Toyota Technological Institute at Chicago
USA
saranurak@ttic.edu

***Abstract*—There is a recent exciting line of work in distributed graph algorithms in the CONGEST model that exploit expanders. *All* these algorithms so far are based on two tools: *expander decomposition* and *expander routing*. An (ϵ, ϕ)-expander decomposition removes ϵ-fraction of the edges so that the remaining connected components have conductance at least ϕ, i.e., they are ϕ-expanders, and expander routing allows each vertex v in a ϕ-expander to very quickly exchange $\deg(v)$ messages with *any* other vertices, not just its local neighbors.**

In this paper, we give the first efficient *deterministic* distributed algorithms for both tools. We show that an (ϵ, ϕ)-expander decomposition can be deterministically computed in $\text{poly}(\epsilon^{-1})n^{o(1)}$ rounds for $\phi = \text{poly}(\epsilon)n^{-o(1)}$, and that expander routing can be performed deterministically in $\text{poly}(\phi^{-1})n^{o(1)}$ rounds. Both results match previous bounds of randomized algorithms by [Chang and Saranurak, PODC 2019] and [Ghaffari, Kuhn, and Su, PODC 2017] up to subpolynomial factors.

Consequently, we *derandomize* existing distributed algorithms that exploit expanders. We show that a minimum spanning tree on $n^{-o(1)}$-expanders can be constructed deterministically in $n^{o(1)}$ rounds, and triangle detection and enumeration on *general* graphs can be solved deterministically in $O(n^{0.58})$ and $n^{2/3+o(1)}$ rounds, respectively.

Using similar techniques, we also give the first *polylogarithmic-round* randomized algorithm for constructing an (ϵ, ϕ)-expander decomposition in $\text{poly}(\epsilon^{-1}, \log n)$ rounds for $\phi = 1/\text{poly}(\epsilon^{-1}, \log n)$. This algorithm is faster than the previous algorithm by [Chang and Saranurak, PODC 2019] in all regimes of parameters. The previous algorithm needs $n^{\Omega(1)}$ rounds for any $\phi \geq 1/\text{poly}\log n$.

***Keywords*-distributed algorithms; expander decomposition; derandomization; triangle enumeration;**

I. Introduction

In this paper[1], we consider the CONGEST model of distributed computing, where the underlying distributed network is represented as an graph $G = (V, E)$, where each vertex corresponds to a computer, and each edge corresponds to a communication link. Each vertex $v \in V$ has a distinct $\Theta(\log n)$-bit identifier $\text{ID}(v)$, where $n = |V|$ is the number of vertices in the graph. The computation proceeds in synchronized *rounds*. In each round, each vertex $v \in V$ can perform unlimited local computation, and may send a distinct $O(\log n)$-bit message to each of its neighbors. In the randomized variant of CONGEST, each vertex can generate unlimited local random bits, but there is no global randomness. A related model called the CONGESTED-CLIQUE model is a variant of the CONGEST model where each vertex $v \in V$ is able to send a separate $O(\log n)$-bit message to each vertex in $V \setminus \{v\}$.

Expander routing: Ghaffari, Kuhn, and Su [1] considered a routing problem on high-conductance graphs. They proved that if each vertex $v \in V$ 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 with high probability, where $\tau_{\text{mix}}(G)$ is the mixing time of the lazy random walk on G, and we have the following relation [2] between the mixing time $\tau_{\text{mix}}(G)$ and conductance $\Phi(G)$:

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

The $2^{O(\sqrt{\log n \log\log n})}$ factor was later improved by Ghaffari and Li [3] to $2^{O(\sqrt{\log n})}$.

Expander routing is a very useful tool in designing distributed algorithms on high-conductance graphs. In particular, it was shown in [1] that a minimum spanning tree can be constructed in $\text{poly}\left(\phi^{-1}\right) \cdot 2^{O(\sqrt{\log n})}$ rounds on graphs G with $\Phi(G) = \phi$, bypassing the $\Omega(\sqrt{n}/\log n)$ lower bound for general graphs [4], [5].

More generally, as the expander routing algorithms of [1], [3] allows the vertices to communicate arbitrarily, only subjecting to some bandwidth constraints, these routing algorithms allow us to simulate known algorithms from other models of parallel and distributed graph algorithms with small overhead. Indeed, it was shown in [3] that many work-efficient PRAM algorithms can be transformed into round-efficient distributed algorithms in the CONGEST model when the underlying graph G has high conductance.

Expander decompositions: A major limitation of the approach of [1], [3] is that it is only applicable to graphs with high conductance. A natural idea to extend this line of research to general graphs is to consider *expander decompositions*. Formally, an (ϵ, ϕ)-expander decomposition

[1]See https://arxiv.org/pdf/2007.14898.pdf for the full version of the paper.

2575-8454/20/$31.00 ©2020 IEEE
DOI 10.1109/FOCS46700.2020.00043

of a graph $G = (V, E)$ is a partition of the edges $E = E_1 \cup E_2 \cup \cdots \cup E_x \cup E^r$ meeting the following conditions.

- The subgraphs $G[E_1], G[E_2], \ldots, G[E_x]$ induced by the clusters are vertex-disjoint.
- $\Phi(G[E_x]) \geq \phi$, for each $1 \leq i \leq x$.
- The number of remaining edges is at most ϵ fraction of the total number of edges, i.e., $|E^r| \leq \epsilon|E|$.

In other words, an (ϵ, ϕ)-expander decomposition of a graph $G = (V, E)$ is a removal of at most ϵ fraction of its edges in such a way that each remaining connected component has conductance at least ϕ. It is well known that for *any* graph, and an (ϵ, ϕ)-expander decomposition exists for any $0 < \epsilon < 1$ and $\phi = \Omega(\epsilon/\log n)$ [6], [7], [8], and this bound is *tight*: after removing any constant fraction of the edges in a hypercube, some remaining component must have conductance at most $O(1/\log n)$ [9]. Expander decompositions have a wide range of applications, and it has been applied to solving linear systems [8], unique games [10], [11], [12], minimum cut [13], and dynamic algorithms [14].

Distributed expander decompositions: Recently, Chang, Pettie, and Zhang [15] applied expander decompositions to the field of distributed computing, and they showed that a variant of expander decomposition can be computed efficiently in CONGEST. Using this decomposition, they showed that triangle detection and enumeration can be solved in $\tilde{O}(n^{1/2})$ rounds.[2] The previous upper bounds for triangle detection and enumeration were $\tilde{O}(n^{2/3})$ and $\tilde{O}(n^{3/4})$, respectively, due to Izumi and Le Gall [16]. Later, Chang and Saranurak [17] improved the expander decomposition algorithm of [15]. For any $0 < \epsilon < 1$, and for any positive integer k, an (ϵ, ϕ)-expander decomposition of a graph $G = (V, E)$ with $\phi = (\epsilon/\log n)^{2^{O(k)}}$ can be constructed in $O\left(n^{2/k} \cdot \text{poly}\left(\phi^{-1}, \log n\right)\right)$ rounds with high probability. As a consequence, triangle detection and enumeration can be solved in $\tilde{O}(n^{1/3})$ rounds, matching the $\tilde{\Omega}(n^{1/3})$ lower bound of Izumi and Le Gall [16] and Pandurangan, Robinson, and Scquizzato [18].

The triangle finding algorithms of [15], [17] are based on the following generic framework. Construct an expander decomposition, and then the routing algorithms of [1], [3] enables us to simulate some known CONGESTED-CLIQUE algorithms with small overhead on the high-conductance subgraphs $G[E_1], G[E_2], \ldots, G[E_x]$, and finally the remaining edges E^r can be handled using recursive calls.

After [15], [17], other applications of distributed expander decompositions have been found. Daga et al. [19] applied distributed expander decompositions to obtain the first sublinear-round algorithm for *exact* edge connectivity.

Eden et al. [20] showed that distributed expander decompositions are also useful for various distributed subgraph finding problems beyond triangles. For *any* k-vertex subgraph H, whether a copy of H exists can be detected in $n^{2-\Omega(1/k)}$ rounds, matching the $n^{2-O(1/k)}$ lower bound of Fischer et al. [21]. There is a constant $\delta \in (0, 1/2)$ such that for any k, any $\Omega(n^{(1/2)+\delta})$ lower bound on $2k$-cycle detection would imply a new circuit lower bound.

Eden et al. [20] also showed that all 4-cliques can be enumerated in $\tilde{O}(n^{5/6})$ rounds, and all 5-cliques in $\tilde{O}(n^{21/22})$ rounds. Censor-Hillel, Le Gall, and Leitersdorf [22] later improved this result, showing that all k-cliques can be enumerated in $\tilde{O}(n^{2/3})$ rounds for $k = 4$ and $\tilde{O}(n^{1-2/(k+2)})$ rounds for $k \geq 5$, getting closer to the $\tilde{\Omega}(n^{1-2/k})$ lower bound of Fischer et al. [21].

A. Our Contribution

The main contribution of this paper is to offer the first efficient *deterministic* distributed algorithms for both expander decomposition and routing in the CONGEST model.

Theorem 1.1 (Deterministic expander decomposition): Let $0 < \epsilon < 1$ be a parameter. An (ϵ, ϕ)-expander decomposition of a graph $G = (V, E)$ with $\phi = \text{poly}(\epsilon) 2^{-O(\sqrt{\log n \log\log n})}$ can be computed in $\text{poly}(\epsilon^{-1}) 2^{O(\sqrt{\log n \log\log n})}$ rounds deterministically.

More generally, there is a tradeoff of parameters in Theorem 1.1. For any $1 > \gamma \geq \sqrt{\log\log n / \log n}$, there is a deterministic expander decomposition algorithm with round complexity $\epsilon^{-O(1)} \cdot n^{O(\gamma)}$ with parameter $\phi = \epsilon^{O(1)} \log^{-O(1/\gamma)} n$. For example, an (ϵ, ϕ)-expander decomposition of a graph $G = (V, E)$ with $\phi = \text{poly}(\epsilon/\log n)$ can be computed deterministically in $\text{poly}\left(\epsilon^{-1}\right) n^{0.001}$ rounds.

Theorem 1.2 (Deterministic routing on expanders): Let $G = (V, E)$ be a graph with $\Phi(G) = \phi$, where vertex $v \in V$ is a source and a destination of $O(L) \cdot \deg(v)$ messages. Then there is a deterministic algorithm that routes all messages to their destination in $O(L) \cdot \text{poly}\left(\phi^{-1}\right) \cdot 2^{O\left(\log^{2/3} n \log^{1/3} \log n\right)}$ rounds.

These results open up the possibility of *derandomization* of randomized distributed algorithms that are based on these techniques [1], [3], [15], [17], [19], [20], [23], [24], [25], [22].

We show that triangle detection and counting can be solved deterministically in $n^{1-\frac{1}{\omega}+o(1)} < O(n^{0.58})$ rounds, and triangle enumeration can be solved deterministically in $n^{\frac{2}{3}+o(1)}$ rounds, by derandomizing the algorithm of [15] using Theorems 1.1 and 1.2. See Section II for a formal definition of these problems. To the best of our knowledge, before this work, the only known deterministic algorithm for triangle detection is the trivial $O(n)$-round algorithm that asks each vertex v to send $\{\text{ID}(u) \mid u \in N(v)\}$ to all its neighbors.

Theorem 1.3 (Triangle finding): Triangle detection and counting can be solved deterministically in $n^{1-\frac{1}{\omega}+o(1)} < O(n^{0.58})$ rounds, and triangle enumeration can be solved deterministically in $n^{\frac{2}{3}+o(1)}$ rounds.

[2] The $\tilde{O}(\cdot)$ notation hides any factor polylogarithmic in n.

We show that a minimum spanning tree can be constructed in $\text{poly}\left(\phi^{-1}\right) \cdot n^{o(1)}$ rounds deterministically on graphs G with conductance $\Phi(G) = \phi$, by derandomizing the algorithm of [1] using Theorem 1.2. To the best of our knowledge, before this work, there is no known deterministic algorithm that can take advantage of the fact that the underlying graph has high conductance. Even for the case of $\Phi(G) = \Omega(1)$, the state-of-the-art deterministic algorithm is the well-known one that costs $O(D + \sqrt{n}\log^* n)$ rounds, where D is the diameter of the graph [26].

Theorem 1.4 (Minimum spanning trees): A minimum spanning tree of a graph G with $\Phi(G) = \phi$ can be constructed deterministically in $\text{poly}\left(\phi^{-1}\right) \cdot n^{o(1)}$ rounds.

The techniques used in our deterministic expander decomposition algorithms also enables us to obtain an improved expander decomposition algorithm in the randomized setting. More specifically, we can afford to have $\phi = \Omega\left(\epsilon^3 \log^{-10} n\right)$ in Theorem 1.5.

Theorem 1.5 (Randomized expander decomposition): Let $0 < \epsilon < 1$ be a parameter. An (ϵ, ϕ)-expander decomposition of a graph $G = (V, E)$ with $\phi = 1/\text{poly}(\epsilon^{-1}, \log n)$ can be computed in $\text{poly}(\epsilon^{-1}, \log n)$ rounds with high probability.

This is the *first* polylogarithmic-round distributed algorithm for expander decomposition with $\phi = 1/\text{poly}\log n$. The previous algorithm [17] needs $n^{\Omega(1)}$ rounds if $\phi = 1/\text{poly}\log n$. In fact, Theorem 1.5 attains better round complexity than the algorithm of [17] in all regimes of parameters. Specifically, for any given positive integer k, the algorithm of [17] computes an expander decomposition with $\phi = 1/\text{poly}(\epsilon^{-1}, \log n)^{2^{O(k)}}$ in $\text{poly}(\epsilon^{-1}, \log n)^{2^{O(k)}} \cdot n^{2/k}$ rounds, where the exponent $2^{O(k)}$ is enormous even for small constant k. We note, however, that the expander decomposition of [17] has an additional guarantee that each cluster $G[E_i]$ in the decomposition is a *vertex-induced* subgraph.

B. Preliminaries

We present the graph terminologies used in this paper.

Basic graph notation: The parameter n always denotes the number of vertices, and the parameter Δ always denotes the maximum degree. We write "with high probability" to denote a success probability of $1 - 1/\text{poly}(n)$. For any graph $H = (V', E')$, we write $H[S]$ to denote the subgraph of H induced by S, where S can be an edge set $S \subseteq E'$ or a vertex set $S \subseteq V'$.

For each vertex v, denote $N(v)$ as the set of neighbors of v. We write $\text{dist}(u, v)$ to denote the distance between u and v. For a subset $S \subseteq V$, denote by $E(S)$ the set of all edges with both endpoints in S. Similarly, $E(S_1, S_2)$ is the set of all edges between S_1 and S_2. These terms depend on the underlying graph G, which appears subscripted if not clear from context.

Embeddings: An *embedding* of a graph $H = (V', E')$ into a vertex set $S \subseteq V$ of $G = (V, E)$ with *congestion* c and *dilation* d consists of the following.

- A bijective mapping f between the vertices of V' and S.
- A set of paths $\mathcal{P} = \{P_e \mid e \in E'\}$, where path P_e is a path in G whose two ends are $f(u)$ and $f(v)$.
- Each path $\mathcal{P}$ has length at most d, and each edge $e \in E$ appears in at most c paths in $\mathcal{P}$.

Steiner trees: A *Steiner tree* T for a vertex set S is a tree whose leaf vertices are exactly S. In this paper we often need to deal with graphs $H = (V', E')$ that have high diameter or are disconnected, but they will be supplied with a Steiner tree T' for V' with small diameter so that the vertices in V' are able to communicate efficiently. Throughout this paper, unless otherwise stated, D denotes the diameter of the Steiner tree of the current graph, not the diameter of the current graph.

Conductance: Consider a graph $G = (V, E)$. For a vertex subset S, we write $\text{Vol}(S)$ to denote $\sum_{v \in S} \deg(v)$. Let $\partial(S) = E(S, V \setminus S)$ be the set of edges $e = \{u, v\}$ with $u \in S$ and $v \in V \setminus S$. The *conductance* of a cut S is defined as

$$\Phi(S) = \frac{|\partial(S)|}{\min\{\text{Vol}(S), \text{Vol}(V \setminus S)\}}.$$

For the special case of $S = \emptyset$ and $S = V$, we set $\Phi(S) = 0$. The *conductance* of a graph G is

$$\Phi(G) = \min_{S \subseteq V \text{ s.t. } S \neq \emptyset \text{ and } S \neq V} \Phi(S).$$

In other words, $\Phi(G)$ is the minimum value of $\Phi(S)$ over all non-trivial cuts $S \subseteq V$.

Sparsity: The *sparsity* of a cut S is defined as

$$\Psi(S) = \frac{|\partial(S)|}{\min\{|S|, |V \setminus S|\}}.$$

For the special case of $S = \emptyset$ and $S = V$, we set $\Psi(S) = 0$. The *sparsity* of a graph G is

$$\Psi(G) = \min_{S \subseteq V \text{ s.t. } S \neq \emptyset \text{ and } S \neq V} \Psi(S).$$

In other words, sparsity is a variant of conductance where the volume of a vertex set is measured by its cardinally. Note that sparsity is also commonly known as *edge expansion*. Sparsity and conductance differs by a factor of at most Δ.

Inner and outer sparsity: Consider a partition $\mathcal{V} = \{V_1, V_2, \ldots, V_x\}$ of V for a graph $G = (V, E)$. We say that a cut S *respects* $\mathcal{V}$ if for each $V_i \in \mathcal{V}$, either $V_i \subseteq S$ or $V_i \subseteq V \setminus S$. The *outer sparsity* $\Psi^{\mathsf{out}}_{G,\mathcal{V}}$ of G is the minimum of $\Psi_G(S)$ over all cuts S that respects $\mathcal{V}$. The *inner sparsity* $\Psi^{\mathsf{in}}_{G,\mathcal{V}}$ of G is the minimum of $\Psi(G[V_i])$ over all $V_i \in \mathcal{V}$.

Expander split graphs: The *expander split* $G^\diamond = (V^\diamond, E^\diamond)$ of $G = (V, E)$ is constructed as follows.

- For each $v \in V$, construct a $\deg(v)$-vertex expander graph X_v with $\Delta(X_v) = \Theta(1)$ and $\Phi(X_v) = \Theta(1)$.
- For each vertex $v \in V$, consider an arbitrary ranking of its incident edges. Denote $r_v(e)$ the rank of e at v. For each edge $e = \{u, v\} \in E$, add an edge linking the $r_u(e)$th vertex of X_u and the $r_v(e)$th vertex of X_v.

The concept of expander split graphs and the inner and outer sparsity are from [27]. Note that in the distributed setting, $G^\diamond$ can be simulated in G with no added cost and $\Psi(G^\diamond)$ and $\Phi(G)$ are within a constant factor of each other.

C. Technical Overview

Throughout the paper, we say that a cut $C \subseteq V$ is *balanced* to informally indicate that $\min\{|V \setminus C|, |C|\}$ or $\min\{\mathrm{Vol}(V \setminus C), \mathrm{Vol}(C)\}$ is high, and we say that a graph G is *well-connected* or is an *expander* to informally indicate that $\Psi(G)$ or $\Phi(G)$ is high.

The main ingredients underlying our distributed expander decomposition algorithms are efficient randomized and deterministic distributed algorithms solving the following task $\mathcal{P}$. See the full version of the paper for the precise specifications of the task $\mathcal{P}$ that we use.

Input: A bounded-degree graph $G = (V, E)$ and two parameters $0 < \psi_{\mathsf{emb}} < \psi_{\mathsf{cut}} < 1$.

Output: Two subsets $W \subseteq V$ and $C \subseteq V$ satisfying the following conditions.

- Expander: The induced subgraph $G[W]$ has $\Psi(G[W]) \geq \psi_{\mathsf{emb}}$.
- Cut: The cut C satisfies $0 \leq |C| \leq |V|/2$ and $\Psi(C) \leq \psi_{\mathsf{cut}}$.
- Balance: Either one of the following is met.
  - $|C| = \Omega(1) \cdot |V|$, i.e., C is a *balanced* cut.
  - $|V \setminus (C \cup W)|$ is *small*.

Note that the requirement $\Psi(G[W]) \geq \psi_{\mathsf{emb}} > 0$ implies that $G[W]$ must be a connected subgraph of G if $W \neq \emptyset$, and it is possible that $G[C]$ is disconnected.

Randomized sequential algorithms: We first review existing sequential algorithms solving $\mathcal{P}$. The task $\mathcal{P}$ can be solved using a technique called the *cut-matching game*, which was first introduced by Khandekar, Rao, and Vazirani [28]. The goal of a cut-matching game on $G = (V, E)$ is to either find a *small-congestion embedding* of a well-connected graph H into V, certifying that G is also well-connected, or to find a sparse cut $C \subseteq V$. The game proceeds in iterations. Roughly speaking, in each iteration, the cut player uses some strategy to produce two disjoint subsets $S \subseteq V$ and $T \subseteq V$ with $|S| \leq |T|$, and then the matching player tries to embed a *matching* between S and T that *saturates all vertices* in S with *small congestion* to the underlying graph G. If the matching player fails to do so because such a small-congestion matching embedding does not exist, then there must be a sparse cut C in G separating the unmatched vertices in S and the unmatched vertices in T. If the matching player is always successful, then the strategy for the cut player guarantees that after $\mathrm{poly}\log n$ iterations, the union of all matchings found by the matching player forms a graph H with sparsity $\Psi(H) = \Omega(1)$, certifying that G itself is well-connected.

The above algorithm does not solve $\mathcal{P}$ yet. To facilitate discussion, let G be a graph that contains a well-connected subgraph $G[W]$ with $|W| = (1 - \beta)|V|$ and a sparse cut C with $|C| = \Omega(\beta) \cdot |V|$, for some parameter $0 < \beta < 1$. Intuitively, the presence of the sparse cut C implies the non-existence of a well-connected subgraph $G[W']$ of G with $|W| = (1 - o(\beta))|V|$, and so G itself is not well-connected. If we apply the above cut-matching game algorithm on G, then it will return us some sparse cut C', as the matching player will at some point fail to embed a large enough matching. However, there is no guarantee on the balance of the cut C', and $|C'|$ can be arbitrarily small.

Räcke, Shah, and Täubig [29] considered a variant of the cut-matching game that deals with this issue. The main difference is that in the RST cut-matching game, we continue the cut-matching game on the remaining part of the graph even if a sparse cut C_i is found in an iteration i, unless the union of all sparse cuts $C_1 \cup C_2 \cup \cdots \cup C_i$ found so far already has size $\Omega(|V|)$. Hence there are two possible outcomes of the RST cut-matching game. If the RST cut-matching game stops early, then the output is a sparse cut C of size $\Omega(|V|)$, which is good, as we can set $W = \emptyset$ to satisfy the requirements for $\mathcal{P}$. Otherwise, the output is a sparse cut C together with a small-congestion embedding of the union of all matchings $H = M_1 \cup M_2 \cup \cdots \cup M_\tau$ that the matching player found. If $C = \emptyset$, then as discussed earlier, the small-congestion embedding of H certifies that G itself is well-connected, then we can set $W = V$ to satisfy the requirements for $\mathcal{P}$.

If $C \neq \emptyset$, then the embedding of H does not guarantee anything about $\Psi(G)$. To deal with this issue, Saranurak and Wang [30] showed that $G[V \setminus C]$ is actually nearly an expander in the following sense. A subgraph $G[U]$ of G induced by $U \subset V$ is said to be *nearly* a ϕ-expander if for all $S \subseteq U$ with $0 < \mathrm{Vol}(S) \leq \mathrm{Vol}(U)/2$, we have $|\partial_G(S)| \geq \phi \,\mathrm{Vol}(S)$. Note that if we change the requirement from $|\partial_G(S)| \geq \phi \,\mathrm{Vol}(S)$ to $|\partial_{G[U]}(S)| \geq \phi \,\mathrm{Vol}(S)$, then $G[U]$ would have been a ϕ-expander. Moreover, given such a subgraph $G[U]$, they presented an efficient sequential *expander trimming* algorithm that removes a small fraction $U' \subseteq U$ of the vertices in U in such a way that $\Phi(G[U \setminus U']) = \Omega(\phi)$. Applying this expander trimming algorithm to $G[V \setminus C]$, we can obtain a well-connected subgraph $G[W]$, and C and W together satisfy the requirement for the task $\mathcal{P}$.

Deterministic sequential algorithms: The above sequential algorithm for $\mathcal{P}$ is *randomized* because the strategy of the cut player in the cut-matching game of [28], [29] is inherently randomized. Recently, Chuzhoy et al. [27] designed an efficient *deterministic* sequential algorithm for $\mathcal{P}$ by considering a different cut-matching by Khandekar et al. [31], where the cut player can be implemented recursively and deterministically.

In the KKOV cut-matching game [31], the strategy of the cut player in iteration i is to simply find a sparse cut $S \subseteq V$ that is *as balanced as possible* in the graph $H_{i-1} = M_1 \cup M_2 \cup \cdots \cup M_{i-1}$ formed by the union of all matchings found so far, and set $T = V \setminus S$. Observe that the output cut C of the task $\mathcal{P}$ is also *approximately* as balanced as possible, as the output well-connected subgraph $G[W]$ implies that we cannot find a cut C' that is significantly more balanced than C and significantly sparser than C at the same time. This gives rise to a possibility of solving $\mathcal{P}$ *recursively*. To realize this idea, the approach taken in [27] is to decompose the current vertex set V into k subsets $\mathcal{V} = \{V_1, V_2, \ldots, V_k\}$ of equal size, and then run the KKOV cut-matching game simultaneously on each part of $\mathcal{V}$, and so the cut player can be implemented by solving $\mathcal{P}$ recursively on instances of size $O(n/k)$.

Similar to the case of [28], [29], during the process, it is possible that in some iteration, the matching for some part $V_i \in \mathcal{V}$ returned by the matching player does not saturate all of S. The approach taken in [27] is to consider some threshold $0 < \beta < 1$. If adding at most $\beta|V|$ *fake edges* to the graph is enough to let the cut-matching games in all parts to continue, then we add them to G. Otherwise, we can obtain a sparse cut C of size $\Omega(\beta) \cdot |V| \leq |C| \leq |V|/2$, which is already sufficiently balanced. Suppose we are always able to add a small number of fake edges to let the cut-matching games in all parts continue in each iteration, then in the end we obtain a small-congestion *simultaneous embedding* of an expander H_i to each part $V_i \in \mathcal{V}$ in the *augmented graph* G' which is formed by adding a small number of fake edges to the original graph G. Now, select C to be a sparse cut of G that *respects* $\mathcal{V}$ and is as balanced as possible. If C is already sufficiently balanced, then we can return C. Otherwise, $|C|$ is small, and so we can add a small amount of fake edges to G' to make it a well-connected graph G''.

Now we face a situation that is similar to the case of [28], [29] discussed earlier. That is, we have obtained something that is almost an expander, and we just need to turn it into an expander.

Specifically, we are given a graph G and a small set of fake edges E^* such that the graph resulting from adding all these fake edges to G is well-connected. Saranurak and Wang [30] showed that in such a situation, there is an efficient deterministic *expander pruning* algorithm that is able to find a well-connected subgraph $G[W]$ of G such that $\mathrm{Vol}(V \setminus W)$ is small.

Distributed algorithms: In order to apply these approaches to the distributed setting, we need to deal with the following challenges.

- We need an efficient distributed algorithm to implement the matching player in cut-matching games.
- There is no known efficient distributed algorithms for expander trimming and expander pruning. Moreover, we cannot afford to add fake edges to the distributed network G.

To implement the matching player efficiently, it suffices to be able to solve the following problem efficiently. Given two disjoint sets $S \subseteq V$ and $T \subseteq V$, find a *maximal* set of vertex-disjoint S-T paths of length at most d. In the randomized setting, this problem can be solved in $\mathrm{poly}(d, \log n)$ rounds with high probability using the augmenting path finding algorithm in the distributed approximate matching algorithm of Lotker, Patt-Shamir, and Pettie [32].

In the deterministic setting, we are not aware of an algorithm that can solve this problem in $\mathrm{poly}(d, \log n)$ rounds.[3] However, if we allow a small number of *leftover* vertices, then we can solve the problem efficiently using the approach of Goldberg, Plotkin, and Vaidya [34]. Specifically, given a parameter $0 < \beta < 1$, in $\mathrm{poly}\left(d, \beta^{-1}, \log n\right)$ rounds we can find a set B of size $|B| \leq \beta|V \setminus T|$ and a set of vertex-disjoint S-T paths $\{P_1, P_2, \ldots, P_k\}$ of length at most d in such a way that any S-T path must vertex-intersect either B or some path in $\{P_1, P_2, \ldots, P_k\}$. We will revise the analysis of the KKOV cut-matching game to show that it still works well if the matching returns by the matching player only saturates a constant fraction of S, and so we are allowed to use flow algorithms that have leftover vertices.

The more crucial challenge is the lack of efficient distributed algorithms for expander trimming and expander pruning. To get around this issue, we will consider a different approach of extracting a well-connected subgraph from the embedding of matchings in the cut-matching game. In the randomized setting that is based on the cut-matching game of [28], [29], [30], we will show that we can identify a large subset $U \subseteq V \setminus C$ of *good vertices* so that if we start from $u \in U$ a certain random walk defined the by the matchings $M_1, M_2, \ldots, M_\tau$, then the probability that the walk ends up in $v \in U$ is $\Theta(1/n)$, for all $u, v \in U$. Hence we can use random walks to embed a small-degree random graph to U. If we take W to be the set of all vertices involved in the embedding, then $G[W]$ is well-connected because the embedding has small congestion and small dilation. Here we

[3] Cohen [33] showed that the $(1-\epsilon)$-approximate maximum flow problem on a directed acyclic graph with depth r can be solved in PRAM deterministically in $\mathrm{poly}(r, \epsilon^{-1}, \log n)$ time with $O(|E|/r)$ processors, and this algorithm can be adapted to the CONGEST model with the same round complexity $\mathrm{poly}(r, \epsilon^{-1}, \log n)$ if *fractional flow* is allowed. The part of Cohen's algorithm that rounds a fractional flow into an integral flow does not seem to have an efficient implementation in CONGEST, and hence we are unable to use this algorithm.

need the embedding to have *small dilation* in order to bound the sparsity of $G[W]$. This is different from the previous works [28], [29], [30] where we only need the embedding to have small congestion. Note that W not only includes vertices in $V \setminus C$, but it might include vertices in C as well. In contrast, the subset W obtained using expander trimming [30] is guaranteed to be within $V \setminus C$.

In the deterministic setting that is based on the approach of [27], [31], we cannot afford to add fake edges in a distributed network. To get around this issue, we will carry out the simultaneous execution of KKOV cut-matching games using the "non-stop" style of [29] in the sense that we still continue the cut-matching games on the remaining parts of $\mathcal{V}$ even if the cut-matching games in some parts of $\mathcal{V}$ have already failed. Specifically, during the simultaneous execution of cut-matching games, we maintain a cut C in such a way that if the cut-matching game in some part V_i fails, then C includes a constant fraction of vertices in V_i. Therefore, in the end, either constant fraction of the cut-matching games are successful, or $\Omega(1) \cdot |V| \leq |C| \leq |V|/2$. i.e., C is a sufficiently balanced sparse cut. For each part V_i that is successful, it is guaranteed to have a subset $U_i \subseteq V_i$ that is embedded a well-connected graph H_i and $|U_i| \geq (2/3)|V_i|$. Moreover, the overall simultaneous embedding has small congestion and dilation. Now the union of U_i over all successful parts V_i constitutes a constant fraction of vertices in V. We apply the deterministic flow algorithm based on [34] described earlier to enlarge and combine these expander embeddings. During this process, it is possible that a sparse cut is found, and also because of the nature of the approach of [34], there might be a small number of leftover vertices that we cannot handle, but we can show that our algorithm always end up with a sparse cut C with $|C| \leq |V|/2$ and a well-connected subgraph $G[W]$ in one of the following situations.

- $|V|/100 \leq |C|$ and $W = \emptyset$, i.e., C is already a balanced cut.
- $V = C \cup W$.
- $C = \emptyset$ but $|V \setminus W|$ is small, i.e., there is only a small number of leftover vertices that we cannot handle.

This output is already good enough for our purpose.

Expander routing: At a very high level, our deterministic expander routing algorithm follows a similar approach of the randomized routing algorithm of Ghaffari, Kuhn, and Su [1]. Consider a distributed network $G = (V, E)$ with high conductance, where each vertex v is a source and a sink for at most $\deg(v)$ messages. The GKS routing algorithm works as follows. First, simulate a $2|E|$-vertex $O(\log n)$-degree random graph G_0 to G, where each vertex v in G is responsible for $\deg(v)$ vertices in G_0, and the edges in G_0 are constructed by performing lazy random walks. Next, partition the vertices of G_0 into k parts $\mathcal{V} = \{V_1, V_2, \ldots, V_k\}$ of equal size, and simultaneously embed an $O(\log n)$-degree random graph H_i to each part V_i, with small congestion and dilation, also using lazy random walks. The routing is performed recursively by first routing all the messages between parts in $\mathcal{V} = \{V_1, V_2, \ldots, V_k\}$, and then recursively route the messages inside each part to their destinations.

Recall that an intermediate product of our deterministic distributed expander decomposition algorithm is a simultaneous embedding of a high-conductance graph H_i into $U_i \subseteq V_i$ with $|U_i| \geq (2/3)|V_i|$ for the parts $V_i \in \mathcal{V} = \{V_1, V_2, \ldots, V_k\}$ where the cut-matching game does not return a balanced sparse cut. If the underlying graph is guaranteed to be well-connected, then every cut-matching game cannot return a balanced sparse cut. Therefore, we have a simultaneous expander embedding for all parts in $\mathcal{V}$.

In order to apply the recursive approach of [1] using this simultaneous expander embedding, we need to handle the leftover vertices in $V_i \setminus U_i$, as we can only do recursive calls on expanders, and the nature of our approach is that for each part V_i, we can only embed an expander on a subset $U_i \subseteq V_i$, and there are always some remaining vertices. By increasing the round complexity, it is possible to reduce the size of $V_i \setminus U_i$, but we cannot afford to make it an empty set. To deal with these leftover vertices, for each $v \in V_i \setminus U_i$, we will find another vertex $v^\star \in U_i$ that serves as the *representative* of v in all subsequent recursive calls. For each leftover vertex v and its representative $v^\star$, we will establish a communication link between them. In other words, we embed a matching between $V_i \setminus U_i$ and U_i that saturates all vertices in $V_i \setminus U_i$, for each $1 \leq i \leq k$.

The communication links between v and $v^\star$ and also the communication links between different parts U_i and U_j are based on deterministic flow algorithms using the approach of [34]. The issue that there are always some unmatched source vertices in S in the approach of [34] can be resolved for the case of $T = V \setminus S$ by increasing the congestion and round complexity from $\mathrm{poly}(\log n)$ to $2^{O(\sqrt{\log n})}$. Note that we are allowed to have super-polylogarithmic congestion here, as this congestion has no effect on the expander embedding, and so it has no effect on the cost of simulation for recursive calls.

D. Organization

In the full version of the paper, we present the formal proof of our main algorithm for distributed expander decomposition in both the randomized and the deterministic models. Our algorithm for deterministic expander routing is also in the full version of the paper. Below, in Section II, we show two simple applications of our main results in derandomizing distributed graph algorithms.

II. Derandomization

In this section, we present two simple applications of our results in derandomizing distributed graph algorithms.

A. Triangle Finding

In this paper, we consider the following variants of the distributed triangle finding problems. We say that a vertex v found a triangle $\{x, y, z\}$ if v learned the three edges $\{x, y\}$, $\{y, z\}$, and $\{x, z\}$.

- Triangle detection: If the graph contains at least one triangle, then at least one vertex finds a triangle.
- Triangle counting: Each vertex v outputs a number t_v such that $\sum_{v \in V} t_v$ equals the number of triangles in the graph.
- Triangle enumeration: Each triangle in the graph is found by at least one vertex.

We prove the following theorem.

Theorem 1.3 (Triangle finding): Triangle detection and counting can be solved deterministically in $n^{1-\frac{1}{\omega}+o(1)} < O(n^{0.58})$ rounds, and triangle enumeration can be solved deterministically in $n^{\frac{2}{3}+o(1)}$ rounds.

Our triangle finding algorithm follows the high-level idea of [15]. We start with an expander decomposition, and consider a threshold d. For vertices with degree at most d, we can apply the trivial $O(d)$-round triangle listing algorithm, For the remaining high-degree vertices, we simulate a known CONGESTED-CLIQUE algorithm for triangle finding on each high-conductance component of the expander decomposition.

An (ϵ, ϕ)-expander decomposition of a graph $G = (V, E)$ is a partition of its edges $E = E_1 \cup E_2 \cup \cdots \cup E_k \cup E^{\mathsf{r}}$. We write $G_i = (V_i, E_i) = G[E_i]$ to denote the subgraph induced by E_i. Consider the following sets.

$$W_i = \{\, v \in V_i \mid \deg_{E_i}(v) \geq \deg_{E \setminus E_i}(v) \,\}$$
$$E_i^+ = \{\, e = \{u, v\} \in E \mid (u \in W_i) \vee (v \in W_i) \vee (e \in E_i) \,\}$$
$$E_i^- = \{\, e = \{u, v\} \in E_i \mid \{u, v\} \subseteq W_i \,\}$$

In other words, W_i is a subset of V_i containing vertices whose majority of incident edges are in E_i, E_i^+ is the set of all edges that are incident to a vertex in W_i or belong to E_i, E_i^- is the set of edges in E_i whose both endpoints are contained in W_i.

The following lemma allows us to focus triangles with at least one edge in $E_1^-, E_2^-, \ldots, E_k^-$, and then recurse on the remaining edges, and the depth of the recursion is at most $O(\log n)$.

Lemma 2.1 (Number of remaining edges is small): If $\epsilon \leq 1/6$, then $|E_1^- \cup E_2^- \cup \cdots \cup E_k^-| \geq |E|/2$.

Proof: Observe that $|E_i \setminus E_i^-|$ must be at most the number of edges in E^{r} incident to V_i. Since each edge $e \in E^{\mathsf{r}}$ can be incident to at most two parts V_i, V_j, we have $|E| - |E^{\mathsf{r}}| - |E_1^- \cup E_2^- \cup \cdots \cup E_k^-| = \sum_{1 \leq i \leq k} |E_i \setminus E_i^-| \leq 2|E^{\mathsf{r}}|$, and so $|E_1^- \cup E_2^- \cup \cdots \cup E_k^-| \geq |E| - 3|E^{\mathsf{r}}| \geq (1 - 3\epsilon)|E| \geq |E|/2$. ∎

Note that for any triangle with at least one edge in E_i^-, all its three edges must be completely within E_i^+. The following lemma shows that $\Phi(G[E_i^+])$ is high.

Lemma 2.2 ($G[E_i^+]$ has high conductance): $\Phi(G[E_i^+]) \geq \phi/4$.

Proof: Recall that $G[E_i] = (V_i, E_i)$ has $\Phi(G[E_i]) \geq \phi$. We write $G[E_i^+] = (V_i^+, E_i^+)$. To prove the lemma, it suffices to show that for any cut $C \subseteq V_i^+$ of $G[E_i^+]$ with $\Phi_{G[E_i]}(C) \leq 1/2$, we always have $C \cap V_i \neq \emptyset$ and

$$\frac{|\partial_{G[E_i^+]}(C)|}{\mathrm{Vol}_{G[E_i^+]}(C)} \geq \frac{1}{4} \cdot \frac{|\partial_{G[E_i]}(C \cap V_i)|}{\mathrm{Vol}_{G[E_i]}(C \cap V_i)}.$$

Since $E_i \subseteq E_i^+$, we have $|\partial_{G[E_i^+]}(C)| \geq |\partial_{G[E_i]}(C)| \geq |\partial_{G[E_i]}(C \cap V_i)|$. Therefore, we only need to show that

$$\mathrm{Vol}_{G[E_i^+]}(C) \leq 4 \cdot \mathrm{Vol}_{G[E_i]}(C \cap V_i).$$

We define the following four numbers.

$$a = |\{\, e = \{u, v\} \in E_i \mid u \in C, v \notin C \,\}|$$
$$b = |\{\, e = \{u, v\} \in E_i \mid \{u, v\} \subseteq C \,\}|$$
$$c = |\{\, e = \{u, v\} \in E_i^+ \setminus E_i \mid u \in C, v \notin C \,\}|$$
$$d = |\{\, e = \{u, v\} \in E_i^+ \setminus E_i \mid \{u, v\} \subseteq C \,\}|$$

It is clear that $a + 2b = \mathrm{Vol}_{G[E_i]}(C \cap V_i)$ and $a + 2b + c + 2d = \mathrm{Vol}_{G[E_i^+]}(C)$.

Since $\Phi_{G[E_i]}(C) \leq 1/2$, we have $2a + 2c = 2 \cdot |\partial_{G[E_i^+]}(C)| \leq \mathrm{Vol}_{G[E_i^+]}(C) = a + 2b + c + 2d$. Observe that all edges in $\{\, e = \{u, v\} \in E_i^+ \setminus E_i \mid \{u, v\} \subseteq C \,\}$ are outside of E_i and are incident to $C \cap W_i \subseteq C \cap V_i$. In view of the definition of W_i, the size d of this set $\{\, e = \{u, v\} \in E_i^+ \setminus E_i \mid \{u, v\} \subseteq C \,\}$ is at most $a + b$, since $a + b$ is the total number of edges in E_i incident to $C \cap V_i$. Using these two inequalities $2a + 2c \leq a + 2b + c + 2d$ and $d \leq a + b$, we have

$$\begin{aligned} c &\leq -a + 2b + 2d \\ c + 2d &\leq -a + 2b + 4d \\ c + 2d &\leq 3a + 6b \\ a + 2b + c + 2d &\leq 4a + 8b \end{aligned}$$

where the first inequality is because $2a + 2c \leq a + 2b + c + 2d$ and the third inequality is because $d \leq a + b$. Therefore, $\mathrm{Vol}_{G[E_i^+]}(C) \leq 4 \cdot \mathrm{Vol}_{G[E_i]}(C \cap V_i)$, as required. ∎

In view of the above lemma, we are able to apply the routing algorithm of Theorem 1.2 on $G[E_i^+] = (V_i^+, E_i^+)$. Note that each edge $e \in E$ belongs to at most two sets E_i^+ and E_j^+, and so congestion of processing $G[E_i^+]$ for $1 \leq i \leq k$ in parallel is not an issue. We deal with low-degree vertices and high-degree vertices separately.

Low-degree vertices: Consider the $S_i = \{v \in V_i \mid \deg_{G[E_i^+]}(v) \leq d\}$, where d is a threshold to be determined. It is clear that by having all vertices in $v \in S_i$ sending $\{\mathrm{ID}(u) \mid \{u, v\} \in E_i^+\}$ to all its neighbors, we are able to list all triangles in $G[E_i^+]$ involving at least one edge incident to S_i, in the sense that any such triangle is found by some vertex in $G[E_i^+]$. This algorithm takes $O(d)$

rounds deterministically. Furthermore, we can additionally require that each triangle is listed by exactly one vertex by introducing a tie breaking mechanism. For a triangle $\{x,y,z\}$ in $G[E_i^+]$ with $\{x,y,z\} \cap S_i \neq \emptyset$, we let the vertex in $\{x,y,z\} \cap S_i$ that has the largest ID to list the triangle $\{x,y,z\}$.

High-degree vertices: Now, we let $G_i^* = (V_i^*, E_i^*)$ to denote the subgraph of $G[E_i^+]$ induced by $V_i^* = V_i^+ \setminus S_i$. All triangles in $G[E_i^+]$ that are not listed yet belong to this graph G_i^*. We will handle these triangles by simulating a known CONGESTED-CLIQUE algorithm. Specifically, triangle enumeration can be solved in $O(n^{1/3}/\log n)$ rounds [35] deterministically in CONGESTED-CLIQUE, and triangle detection and counting can be solved in $n^{1-2\omega^{-1}+o(1)} < O(n^{0.158})$ rounds [36] deterministically in CONGESTED-CLIQUE, where $\omega < 0.2373$ is the exponent for the complexity of matrix multiplication.

Let $n' = |V_i^*| \leq n$. To simulate CONGESTED-CLIQUE algorithms on G_i^*, we first need to re-assign the IDs of vertices in V_i^* to $\{1,2,\ldots,n'\}$. ID re-assignment can be done straightforwardly in $O(D) = O(\phi^{-2}\log n)$ rounds deterministically, see [15, Lemma 4.1].

To simulate one round of CONGESTED-CLIQUE on G_i^*, we need to be able to let each vertex in $v \in V_i^*$ to route a separate $O(\log n')$-bit message to all other vertices in V_i^*. Since each vertex $v \in V_i^*$ has degree $\deg_{G_i^+}(v) > d$, one invocation of the routing algorithm of Theorem 1.2 on $G[E_i^+]$ with $L = O(n/d)$ allows each vertex $v \in V_i^*$ to send and receive $O(n)$ messages, which is enough to simulate one round of CONGESTED-CLIQUE. Therefore, the overhead of simulation is $O(n/d) \cdot \text{poly}\left(\phi^{-1}\right) \cdot n^{o(1)}$.

Round complexity analysis: We set $\epsilon = 1/6$ and $\phi = 1/2^{O(\sqrt{\log n \log\log n})}$. An (ϵ,ϕ)-expander decomposition of a graph $G = (V,E)$ can be found in $2^{O(\sqrt{\log n \log\log n})}$ rounds deterministically using Theorem 1.1. Note that the round complexity for finding an (ϵ,ϕ)-expander decomposition is negligible comparing with other costs.

For triangle enumeration, the overall round complexity is $O(d) + O(n^{1/3}/\log n) \cdot O(n/d) \cdot n^{o(1)}$ by simulating the $O(n^{1/3}/\log n)$-round CONGESTED-CLIQUE algorithm of [35]. Setting $d = n^{2/3}$, we obtain the overall round complexity $n^{(2/3)+o(1)}$.

For triangle counting and detection, the overall round complexity is $O(d) + n^{1-2\omega^{-1}+o(1)} \cdot O(n/d) \cdot n^{o(1)}$ by simulating the $n^{1-2\omega^{-1}+o(1)}$-round CONGESTED-CLIQUE algorithm of [36]. Setting $d = n^{1-\omega^{-1}}$, we obtain the overall round complexity $n^{1-\omega^{-1}+o(1)} < O(n^{0.158})$. Hence we conclude the proof of Theorem 1.3.

Isolating a triangle: We note that the triangle detection algorithm of [36] only lets us know whether a triangle exists, but it does not find one explicitly when there is at least one triangle. To be able to find a triangle explicitly, we can apply the following strategy. Suppose that the algorithm of [36] tells us that there is at least one triangle in $G_i^* = (V_i^*, E_i^*)$. Then we partition the edge set E_i^* into four parts E^1, E^2, E^3, E^4 of equal size. We apply the same triangle detection algorithm for the subgraph induced by $E_i^* \setminus E^j$, for each $1 \leq j \leq 4$. Note that one of these four edge sets $E_i^* \setminus E^j$ must contain a triangle, and the triangle detection algorithm is able to tell us which of them has at least one triangle, and then we recurse on that edge set. It is clear that after $O(\log n)$ iterations, we are able to isolate exactly one triangle of G_i^*, and this only adds an $O(\log n)$ factor in the round complexity.

Avoiding repeated counting: To solve the triangle counting problem, we need to avoid counting a triangle more than once. Therefore, when counting triangles in $G_i^* = (V_i^*, E_i^*)$, we need to make sure that we only count the triangles with at least one edge in E_i^-. This can be done by first calculating the number n_1 of triangles in G_i^*, and then calculating the number n_2 of triangles in the subgraph of G_i^* induced by the edges $E_i^* \setminus E_i^-$. Then $n_1 - n_2$ equals the number of triangles in G_i^* with at least one edge in E_i^-.

B. Minimum Spanning Trees

We prove the following theorem.

Theorem 1.4 (Minimum spanning trees): A minimum spanning tree of a graph G with $\Phi(G) = \phi$ can be constructed deterministically in $\text{poly}\left(\phi^{-1}\right) \cdot n^{o(1)}$ rounds.

Our MST algorithm follows the approach of [1], which implements *Boruvka's greedy algorithm*. During the algorithm, we maintain a forest F. Initially F is the trivial forest consisting of 1-vertex trees for each vertex $v \in V$. When F contains more than one tree, pick any tree T in the forest, and let e be any smallest weight edge connecting T and $G \setminus T$, and then add this edge e to the forest F. Note that in each step, two trees in the forest are merged into one. It is well-known that at the end of this process, we obtain a minimum spanning tree [1].

Distributed implementation: The distributed implementation in [1] is randomized. Here we consider the following distributed version of Boruvka's greedy algorithm that can be implemented deterministically.

1) Suppose currently the forest F consists of the trees $T_1, T_2, \ldots, T_k$. Let e_i be any minimum weight edge connecting T_i and $G \setminus T_i$.
2) Let $G^* = (V^*, E^*)$ be a graph defined by $V^* = \{T_1, T_2, \ldots, T_k\}$ and $\{T_i, T_j\} \in E^*$ if either e_i or e_j connects T_i and T_j. It is straightforward to see that G^* is a forest, and $|E^*| \geq k/2$.
3) For each edge $\{T_i, T_j\} \in E^*$, if it is added to E^* because of e_i, then we orient this edge as $T_i \to T_j$; if it is added to E^* because of e_j, then we orient this edge as $T_j \to T_i$. If an edge $\{T_i, T_j\}$ can be oriented in both directions, then it is oriented arbitrarily. Define $\text{indeg}(T_i)$ to be the number of edges in E^* oriented towards T_i.

4) Find an independent set I^* of G^* with $\sum_{T_i \in I^*} \text{indeg}(T_i) \geq |E^*|/3$.
5) For each $T_i \in I^*$ and for each T_j with $T_j \to T_i$, add e_j to the current forest.

Since $\sum_{T_i \in I^*} \text{indeg}(T_i) \geq |E^*|/3 \geq k/6$, the number of trees in the forest F is reduced by a factor of $5/6$ in each iteration of the above algorithm. Therefore, $O(\log n)$ iterations suffice to obtain an MST. For the rest of the proof, we focus on the implementation detail of this algorithm, and we will show that $\text{poly}(\log n)$ invocations of the routing algorithm of Theorem 1.2 with $L = O(1)$ are enough. In subsequent discussion, we write $\tau_0 = \text{poly}\left(\phi^{-1}\right) \cdot n^{o(1)}$ to denote the round complexity of the routing algorithm of Theorem 1.2 with $L = O(1)$.

Maintenance of low-diameter Steiner trees: During the algorithm, for each tree T_i in the current forest F, we maintain a low-diameter Steiner tree T_i^{st} such that the leaf vertices of T_i^{st} are exactly the set of vertices V_i of T_i. Note that the diameter of T_i can be very large, and so we need a low-diameter Steiner tree T_i^{st} to enable efficient communication between the vertices in T_i.

- The Steiner tree T_i^{st} is rooted, and denote r_i the root of T_i^{st}.
- Each vertex v in T_i^{st} has at most two children, and v knows the IDs of its children. Each vertex $v \neq r_i$ in T_i^{st} also knows the ID of its parent.
- We say that a vertex v in T_i^{st} is in layer j if $\text{dist}_{T_i^{\text{st}}}(v, r_i) = j$. Define $\ell_i = \max_{v \in T_i^{\text{st}}} \text{dist}(v, r_i)$ as highest layer number. We assume that each vertex in T_i^{st} knows its layer number and ℓ_i.
- Each edge in T_i^{st} does not need to be an edge in the underlying graph G. We use the routing algorithm of Theorem 1.2 for vertices in T_i^{st} to communicate.
- We require that T_i^{st} contains only the vertices in V_i. We allow each vertex $v \in V_i$ to correspond to multiple vertices in T_i^{st}, but they have to belong to different layers.

Given this implementation, each tree T_i can finds its minimum weight outgoing edge e_i in $O(\ell_i \tau_0)$ rounds by a bottom-up information gathering. In the end, all vertices in V_i knows e_i.

Intuitively, if the algorithm under consideration is sufficiently simple, then one round of G^* can be simulated by $O(\ell^* \tau_0)$ rounds in G, where $\ell^* = \max_{1 \leq i \leq k} \ell_i$, and k is the number of trees $T_1, T_2, \ldots, T_k$ in the current forest F.

Lemma 2.3 (Finding an independent set): An independent set I^* of G^* with $\sum_{T_i \in I^*} \text{indeg}(T_i) \geq |E^*|/3$ can be found in $O(D + \ell^* \tau_0 \log n)$ rounds deterministically.

Proof: A proper 3-vertex coloring of a tree can be found in $O(\log n)$ rounds deterministically [37]. We simulate this algorithm on G^*, and the simulation takes $O(\ell^* \tau_0 \log n)$ rounds on G. After that, we go over each color class I_1, I_2, and I_3 to calculate $\sum_{T_i \in I_j} \text{indeg}(T_i)$ for each $1 \leq j \leq 3$ in $O(D)$ rounds. One I_j of these three sets must have $\sum_{T_i \in I_j} \text{indeg}(T_i) \geq |E^*|/3$. We set $I^* = I_j$. ∎

Given I^*, we know the set of edges that we need to add to F, but we still need to update the Steiner trees. We focus on one tree $T_i \in I^*$, and denote $\mathcal{T}_i$ as the set of trees T_j such that $T_j \to T_i$. We need to merge the Steiner tree T_i^{st} of T_i and the Steiner tree T_j^{st} of all $T_j \in \mathcal{T}$ together into a new Steiner tree.

Lemma 2.4 (Merging the Steiner trees): For each $T_i \in I^*$ in parallel, we can merge the Steiner tree T_i^{st} of T_i and the Steiner trees T_j^{st} of all $T_j \in \mathcal{T}_i$ into a new Steiner tree in $O(\ell^* \tau_0 \log n)$ rounds. Moreover, the height of the new Steiner tree is at most $\ell^* + O(\log n)$.

Proof: Recall that V_i denotes the vertex set of T_i, and V_i is identical to the leaf vertices of T_i^{st}. We assume that at the beginning of the algorithm, each leaf v of T_i^{st} holds a list L_v indicating the set of all $\text{ID}(r_j)$, for each edge e_j added to the forest that are incident to v. Note that for each $T_j \in \mathcal{T}_i$, r_j belongs to the list L_v for the unique vertex $v \in V_i$ incident to the edge e_j. If $v \in T_i^{\text{st}}$ it not a leaf, then we assume $L(v) = \emptyset$ initially.

Consider the following algorithm based on a bottom-up traversal of T_i. From $j = \ell_i$ to $j = 0$, all vertices v at layer j of T_i^{st} do the following. It organizes the elements in its set L_v into pairs. If $|L_v|$ is odd, then there will be one leftover element, then v sends it to its parent u to have it added to the list L_u, unless v itself is the root r_i of T_i^{st}. For each pair $(\text{ID}(r_x), \text{ID}(r_y))$, v inform the two vertices r_x and r_y to ask them to merge T_x^{st} and T_y^{st} into a new tree T' by selecting any one of $r' \in \{r_x, r_y\}$ to be the new root, and adding the two new edges $\{r, r_x\}$ and $\{r, r_y\}$. Note that we allow a vertex to appear multiple times in a Steiner tree, so long as all of its appearances are in different layers. After processing all pairs, the vertex v resets its set L_v as the set of IDs of the roots of the merged trees.

It is clear if $|\mathcal{T}_i|$ is an even number, then all trees are merged, and if $|\mathcal{T}_i|$ is an odd number, then all trees except one of them are merged. Therefore, if we continue this process with the new L-sets, then we are done merging all trees in $\mathcal{T}_i$ in $O(\log n)$ iterations. In the end, we merge T_i with the tree resulting from combining all trees in $\mathcal{T}_i$. Overall, the algorithm costs $O(\ell_i \log n) = O(\ell^* \log n)$ rounds, and the final tree resulting from merging has height at most $\ell^* + O(\log n)$, since the number of iterations is $O(\log n)$. ∎

Since the overall algorithm has $O(\log n)$ iterations, Lemma 2.4 implies that the height of all the Steiner trees in the algorithm is at most $O(\log^2 n)$. Therefore, Lemma 2.3 costs $O(D + \tau_0 \log^3 n)$ rounds and Lemma 2.4 costs $O(\tau_0 \log^3 n)$ rounds in each iteration. Recall that the diameter of a graph with conductance ϕ is at most $D = O\left(\phi^{-2} \log n\right)$. Hence the overall round complexity is $O(D \log n + \tau_0 \log^4 n) = \text{poly}\left(\phi^{-1}\right) \cdot n^{o(1)}$. We conclude the proof of Theorem 1.4.

III. Conclusions and Open Questions

In this paper, we give the first subpolynomial-round deterministic distributed algorithms for expander decomposition and routing, and we also give the first polylogarithmic-round randomized distributed algorithms for expander decomposition.

The main obstacle that we overcome is the lack of efficient distributed algorithms for expander trimming and expander pruning. For the randomized setting, we develop a new technique of extracting an expander from a near-expander resulting from the cut-matching game, without using expander trimming. To use this technique, it is crucial that the embedding of the matchings has small dilation. For the deterministic setting, we carry out the simultaneous execution of KKOV cut-matching games using the "non-stop" style of [29]. This allows us to avoid adding fake edges to the graph as in [27].

We believe that our end-results and the techniques therein are of interest beyond the CONGEST model of distributed computing. Since we do not abuse the unlimited local computation power in the definition of CONGEST, our expander decomposition algorithms can be adapted to PRAM and the *massively parallel computation* (MPC) model [38] by a straightforward simulation. Specifically, our deterministic distributed expander decomposition algorithm implies that an (ϵ, ϕ)-expander decomposition of an m-edge n-vertex graph with $\phi = \mathrm{poly}(\epsilon)2^{-O(\sqrt{\log n \log\log n})}$ can be computed in $O(m) \cdot \mathrm{poly}(\epsilon^{-1})2^{O(\sqrt{\log n \log\log n})}$ work and $\mathrm{poly}(\epsilon^{-1})2^{O(\sqrt{\log n \log\log n})}$ depth deterministically in PRAM. Note that PRAM algorithms can be simulated in the MPC model efficiently with the same round complexity [39], [38].

Open questions: The ultimate goal of this research is to give deterministic polylogarithmic-round distributed algorithms for both expander decomposition and routing.

Currently we only have *randomized* polylogarithmic-round algorithm for expander decomposition, and it is still open whether expander routing can be solved in $\mathrm{poly}(\phi^{-1}, \log n)$ rounds even randomness is allowed.

A shortcoming of our expander decomposition algorithms is that each expander in the decomposition is not a vertex-induced subgraph. To transform any expander decomposition to an expander decomposition where each expander is a vertex-induced subgraph, it suffices to use *expander trimming* [30]. It is an open question whether expander trimming can be solved efficiently in CONGEST.

A key ingredient in our deterministic algorithms is the nearly maximal flow algorithm of [34], which results in a small number of leftover vertices. It is an open question whether we can do it without leftover vertices. Specifically, given any two subsets $S \subseteq V$ and $T \subseteq V$, the goal is to find a maximal set of vertex-disjoint paths of length at most d. Can we solve this problem deterministically in $\mathrm{poly}(d, \log n)$ rounds? An affirmative answer to this question will simplify our deterministic expander decomposition and routing quite a bit.

A common version of the low-diameter decomposition is a partition $V = V_1 \cup V_2 \cup \cdots \cup V_x$ such that the number of inter-cluster edges is at most $\beta|E|$, and the (strong) diameter of $G[V_i]$ is $\mathrm{poly}(\beta^{-1}, \log n)$ for each $1 \leq i \leq x$. Such a decomposition can be computed in $\mathrm{poly}(\beta^{-1}, \log n)$ rounds deterministically in the LOCAL model, and it is open whether it can also be computed in $\mathrm{poly}(\beta^{-1}, \log n)$ rounds in CONGEST deterministically [40]. An affirmative answer to this question will also simplify the proofs in this paper as this allows us to get rid of Steiner trees (due to the algorithm of [40]).

Our expander routing algorithm uses the load balancing algorithm of Ghosh et al. [41] to balance the message load across the vertices. To the best of our knowledge, this is the first time this technique is applied to the CONGEST model of distributed computing. It will be interesting to see more applications of this technique. In particular, does it give us some kind of distributed local graph clustering like PageRank [42], [8]?

References

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

[2] M. Jerrum and A. Sinclair, "Approximating the permanent," *SIAM Journal on Computing*, vol. 18, no. 6, pp. 1149–1178, 1989.

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

[4] D. Peleg and V. Rubinovich, "A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction," *SIAM J. Comput.*, vol. 30, no. 5, pp. 1427–1442, 2000.

[5] A. D. Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer, "Distributed verification and hardness of distributed approximation," *SIAM J. Comput.*, vol. 41, no. 5, pp. 1235–1265, 2012.

[6] O. Goldreich and D. Ron, "A sublinear bipartiteness tester for bounded degree graphs," *Combinatorica*, vol. 19, no. 3, pp. 335–373, Mar 1999.

[7] R. Kannan, S. Vempala, and A. Vetta, "On clusterings: good, bad and spectral," *J. ACM*, vol. 51, no. 3, pp. 497–515, May 2004. [Online]. Available: http://doi.acm.org/10.1145/990308.990313

[8] D. A. Spielman and S.-H. Teng, "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems," in *Proceedings 36th Annual ACM Symposium on Theory of Computing (STOC)*, 2004, pp. 81–90.

[9] V. L. Alev, N. Anari, L. C. Lau, and S. O. Gharan, "Graph clustering using effective resistance," in *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, 2018, pp. 41:1–41:16.

[10] S. Arora, B. Barak, and D. Steurer, "Subexponential algorithms for unique games and related problems," *J. ACM*, vol. 62, no. 5, pp. 42:1–42:25, Nov. 2015.

[11] L. Trevisan, "Approximation algorithms for unique games," *Theory of Computing*, vol. 4, no. 5, pp. 111–128, 2008.

[12] P. Raghavendra and D. Steurer, "Graph expansion and the unique games conjecture," in *Proceedings 42nd ACM Symposium on Theory of Computing (STOC)*, 2010, pp. 755–764.

[13] K.-I. Kawarabayashi and M. Thorup, "Deterministic edge connectivity in near-linear time," *J. ACM*, vol. 66, no. 1, pp. 4:1–4:50, Dec. 2018.

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

[15] Y.-J. Chang, S. Pettie, and H. Zhang, "Distributed triangle detection via expander decomposition," in *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2019, pp. 821–840.

[16] T. Izumi and F. Le Gall, "Triangle finding and listing in CONGEST networks," in *Proceedings 37th ACM Symposium on Principles of Distributed Computing (PODC)*, 2017, pp. 381–389. [Online]. Available: http://doi.acm.org/10.1145/3087801.3087811

[17] Y.-J. Chang and T. Saranurak, "Improved distributed expander decomposition and nearly optimal triangle enumeration," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, ser. PODC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 66–73. [Online]. Available: https://doi.org/10.1145/3293611.3331618

[18] G. Pandurangan, P. Robinson, and M. Scquizzato, "On the distributed complexity of large-scale graph computations," in *Proceedings 30th ACM Symposium on Parallelism in Algorithms and Architecture (SPAA)*, 2018.

[19] M. Daga, M. Henzinger, D. Nanongkai, and T. Saranurak, "Distributed edge connectivity in sublinear time," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, ser. STOC 2019. New York, NY, USA: Association for Computing Machinery, 2019, pp. 343–354.

[20] T. Eden, N. Fiat, O. Fischer, F. Kuhn, and R. Oshman, "Sublinear-time distributed algorithms for detecting small cliques and even cycles," in *Proceedings of the International Symposium on Distributed Computing (DISC)*, 2019, pp. 15:1–15:16.

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

[22] K. Censor-Hillel, F. L. Gall, and D. Leitersdorf, "On distributed listing of cliques," in *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, 2020.

[23] T. Izumi, F. L. Gall, and F. Magniez, "Quantum distributed algorithm for triangle finding in the CONGEST model," in *37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), C. Paul and M. Bläser, Eds., vol. 154. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020, pp. 23:1–23:13. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2020/11884

[24] S. Chatterjee, G. Pandurangan, and N. D. Pham, "Distributed MST: a smoothed analysis," in *Proceedings of the 21st International Conference on Distributed Computing and Networking*, ser. ICDCN 2020. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3369740.3369778

[25] H.-H. Su and H. T. Vu, "Distributed data summarization in well-connected networks," in *33rd International Symposium on Distributed Computing (DISC 2019)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), J. Suomela, Ed., vol. 146. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, pp. 33:1–33:16. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2019/11340

[26] S. Kutten and D. Peleg, "Fast distributed construction of small k-dominating sets and applications," *Journal of Algorithms*, vol. 28, no. 1, pp. 40 – 66, 1998.

[27] J. Chuzhoy, Y. Gao, J. Li, D. Nanongkai, R. Peng, and T. Saranurak, "A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond," 2019.

[28] R. Khandekar, S. Rao, and U. Vazirani, "Graph partitioning using single commodity flows," *J. ACM*, vol. 56, no. 4, Jul. 2009. [Online]. Available: https://doi.org/10.1145/1538902.1538903

[29] H. Räcke, C. Shah, and H. Täubig, "Computing cut-based hierarchical decompositions in almost linear time," in *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Society for Industrial and Applied Mathematics, 2014, pp. 227–238.

[30] T. Saranurak and D. Wang, "Expander decomposition and pruning: Faster, stronger, and simpler," in *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2019, pp. 2616–2635.

[31] R. Khandekar, S. Khot, L. Orecchia, and N. K. Vishnoi, "On a cut-matching game for the sparsest cut problem," *Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2007-177*, vol. 6, no. 7, p. 12, 2007.

[32] Z. Lotker, B. Patt-Shamir, and S. Pettie, "Improved distributed approximate matching," *J. ACM*, vol. 62, no. 5, Nov. 2015.

[33] E. Cohen, "Approximate max-flow on small depth networks," *SIAM J. Comput.*, vol. 24, no. 3, pp. 579–597, 1995.

[34] A. V. Goldberg, S. A. Plotkin, and P. M. Vaidya, "Sublinear-time parallel algorithms for matching and related problems," *J. Algor.*, vol. 14, no. 2, pp. 180–213, 1993.

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

[36] K. Censor-Hillel, P. Kaski, J. H. Korhonen, C. Lenzen, A. Paz, and J. Suomela, "Algebraic methods in the congested clique," *Distributed Computing*, 2016.

[37] L. Barenboim and M. Elkin, "Sublogarithmic distributed MIS algorithm for sparse graphs using Nash-Williams decomposition," *Distributed Computing*, vol. 22, no. 5-6, pp. 363–379, 2010.

[38] H. Karloff, S. Suri, and S. Vassilvitskii, "A model of computation for mapreduce," in *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '10. USA: Society for Industrial and Applied Mathematics, 2010, p. 938–948.

[39] M. T. Goodrich, N. Sitchinava, and Q. Zhang, "Sorting, searching, and simulation in the mapreduce framework," in *Algorithms and Computation*, T. Asano, S.-i. Nakano, Y. Okamoto, and O. Watanabe, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 374–383.

[40] V. Rozhoň and M. Ghaffari, "Polylogarithmic-time deterministic network decomposition and distributed derandomization," in *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2020.

[41] B. Ghosh, F. T. Leighton, B. M. Maggs, S. Muthukrishnan, C. G. Plaxton, R. Rajaraman, A. W. Richa, R. E. Tarjan, and D. Zuckerman, "Tight analyses of two local load balancing algorithms," *SIAM J. Comput.*, vol. 29, no. 1, pp. 29–64, 1999. [Online]. Available: https://doi.org/10.1137/S0097539795292208

[42] R. Andersen, F. R. K. Chung, and K. J. Lang, "Local partitioning for directed graphs using PageRank," *Internet Mathematics*, vol. 5, no. 1, pp. 3–22, 2008.