Deterministic Expander Routing: Faster and More Versatile

Yi-Jun Chang
National University of Singapore
Singapore
cyijun@nus.edu.sg

Shang-En Huang
Boston College
Chestnut Hill, Massachusetts, USA
huangaul@bc.edu

Hsin-Hao Su
Boston College
Chestnut Hill, Massachusetts, USA
suhx@bc.edu

ABSTRACT

We consider the expander routing problem formulated by Ghaffari, Kuhn, and Su (PODC 2017), where the goal is to route all the tokens to their destinations given that each vertex is the source and the destination of at most $\deg(v)$ tokens. They developed *randomized algorithms* that solve this problem in $\text{poly}(\phi^{-1}) \cdot 2^{O(\sqrt{\log n \log\log n})}$ rounds in the CONGEST model, where ϕ is the conductance of the graph. In addition, as noted by Chang, Pettie, Saranurak, and Zhang (JACM 2021), it is possible to obtain a preprocessing/query tradeoff so that the routing queries can be answered faster at the cost of more preprocessing time. The efficiency and flexibility of the processing/query tradeoff of expander routing have led to many other distributed algorithms in the CONGEST model, such as subpolynomial-round minimum spanning tree algorithms in expander graphs and near-optimal algorithms for k-clique enumeration in general graphs.

As the routing algorithm of Ghaffari, Kuhn, and Su and the subsequent improved algorithm by Ghaffari and Li (DISC 2018) are both randomized, all the resulting applications are also randomized. Recently, Chang and Saranurak (FOCS 2020) gave a deterministic algorithm that solves an expander routing instance in $2^{O(\log^{2/3} n \cdot \log^{1/3}\log n)}$ rounds. The deterministic algorithm is less efficient and does not allow preprocessing/query tradeoffs, which precludes the de-randomization of algorithms that require this feature, such as the aforementioned k-clique enumeration algorithm in general graphs.

The main contribution of our work is a new deterministic expander routing algorithm that not only matches the randomized bound of Ghaffari, Kuhn, and Su but also allows preprocessing/query tradeoffs. Our algorithm solves a single instance of routing query in $2^{O(\sqrt{\log n \cdot \log\log n})}$ rounds. For instance, this allows us to compute an MST in an expander graph in the same round complexity deterministically, improving the previous state-of-the-art $2^{O(\log^{2/3} n \cdot \log^{1/3}\log n)}$. Our algorithm achieves the following preprocessing and query tradeoffs: For $0 < \epsilon < 1$, we can answer every routing query in $\log^{O(1/\epsilon)} n$ rounds at the cost of a $(n^{O(\epsilon)} + \log^{O(1/\epsilon)} n)$-round preprocessing procedure. Combining this with the approach of Censor-Hillel, Leitersdorf, and Vulakh (PODC 2022), we obtain a near-optimal $\tilde{O}(n^{1-2/k})$-round deterministic algorithm for k-clique enumeration in general graphs, improving the previous state-of-the-art $n^{1-2/k+o(1)}$.

As a side result of independent interest, we demonstrate the *equivalence* between expander routing and *sorting* in the sense that they are reducible to each other up to a polylogarithmic factor in round complexities in the CONGEST model.

CCS CONCEPTS

• **Theory of computation → Distributed algorithms**.

KEYWORDS

CONGEST, routing, MST, sorting, triangle detection

ACM Reference Format:
Yi-Jun Chang, Shang-En Huang, and Hsin-Hao Su. 2024. Deterministic Expander Routing: Faster and More Versatile. In *ACM Symposium on Principles of Distributed Computing (PODC '24), June 17–21, 2024, Nantes, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3662158.3662797

1 INTRODUCTION

The CONGEST model is a prominent model that captures both the locality and the bandwidth in the study of distributed graph algorithms. In this model, the underlying network is a graph $G = (V, E)$, where we let $n = |V|$, $m = |E|$, and Δ = the maximum degree of G. Every vertex v hosts a processor with an unique ID $\in [1, \text{poly}(n)]$. The computation proceeds in synchronized rounds. In each round, each vertex sends a distinct message of $O(\log n)$ bits to each of its neighbors, receives messages from its neighbors, and performs local computations. The complexity of an algorithm is measured as the number of rounds.

In this work, we focus on networks with high *conductance*. Throughout the paper, we say that a graph is a ϕ-*expander* if its conductance is at least ϕ, and we informally say that a graph is an *expander* if it has high conductance. Depending on the context, the conductance of an expander can be $\Omega(1)$, $\log^{-O(1)} n$, or $n^{-o(1)}$.

We consider the following routing problem in a ϕ-expander G in the CONGEST model. Suppose that each vertex v is the source and the destination of at most $\deg(v)$ tokens. The goal is to route all the tokens to their destinations. Ghaffari, Kuhn, and Su [16] developed an algorithm that routes the tokens in $\text{poly}(\phi^{-1}) \cdot 2^{O(\sqrt{\log n \log\log n})}$ rounds. By using such a primitive, a minimum spanning tree (MST) can be computed in $\text{poly}(\phi^{-1}) \cdot 2^{O(\sqrt{\log n \log\log n})}$ rounds in the CONGEST model, beating the $\Omega(\sqrt{n}/\log n)$ lower bound of [28, 30] in general graphs. Later, the $2^{O(\sqrt{\log n \cdot \log\log n})}$ term in the running time has been improved to $2^{O(\sqrt{\log n})}$ later by Ghaffari and Li [17].

Chang, Pettie, Saranurak, and Zhang [7] leveraged the expander routing algorithms to general graphs by developing distributed algorithms for expander decomposition. They showed the method

PODC '24, June 17–21, 2024, Nantes, France

ACM ISBN 979-8-4007-0668-4/24/06.
https://doi.org/10.1145/3662158.3662797

can be used to obtain efficient algorithms for a series of problems. In particular, they obtained CONGEST algorithms for triangle counting, detection, and enumeration whose running times match the triangle enumeration lower bound of [22] up to polylog(n) factors. The approach is to decompose the input graph into disjoint expanders, where only a small number of edges are crossing between different expanders. Within each expander, the ease of routing provided by these algorithms allows one to solve the problem efficiently. They also noted that the algorithms of [16] can be tweaked to have preprocessing/query tradeoffs and used this perk in obtaining the above optimal-round algorithms. In particular, if one spends $O(n^\epsilon)$ time doing the preprocessing then each subsequent routing instance can be answered in $O(\log^{O(1/\epsilon)} n)$ time. This is particularly useful for algorithms that need a polynomial number of queries, as each query can be answered in polylogarithmic rounds if we spend a small-polynomial time for preprocessing.

One major issue left by [16, 17] was that their routing algorithms are randomized. As a result, all the resulting applications are randomized. In [8], they made progress by giving an deterministic algorithm that solves a routing instance in $\text{poly}(\phi^{-1}) \cdot 2^{O(\log^{2/3} n \cdot \log^{1/3} \log n)}$, which is suboptimal compared to the randomized bound of $\text{poly}(\phi^{-1}) \cdot 2^{O(\sqrt{\log n \cdot \log \log n})}$. More importantly, it did not achieve processing/query tradeoffs as in [16]. Therefore, for many applications of the deterministic expander routing, such as the aforementioned results for triangle detection and triangle enumeration, it induces an additional factor of $2^{O(\log^{2/3} n \cdot \log^{1/3} \log n)}$, leaving a substantial gap between randomized and deterministic algorithms.

1.1 Our Contribution

The main contribution of our paper is a deterministic expander routing algorithm that matches the randomized bound of [16] with preprocessing/routing tradeoffs.

Theorem 1.1. *Given a graph $G = (V, E)$ be a ϕ-expander. Let $\epsilon > 0$ be a constant. There exists an algorithm that preprocesses the graph in $n^{O(\epsilon)} + \text{poly}(\phi^{-1}) \cdot (\log n)^{O(1/\epsilon)}$ time such that each subsequent routing instance can be solved in $\text{poly}(\phi^{-1}) \cdot (\log n)^{O(1/\epsilon)}$ rounds.*

Here we see that a single routing instance can be solved in time similar to the bounds obtained by [16] by setting $\epsilon = \sqrt{\log \log n / \log n}$ in Theorem 1.1.

Corollary 1.2. *A single expander routing instance can be solved in $\text{poly}(\phi^{-1}) \cdot 2^{O(\sqrt{\log n \cdot \log \log n})}$ rounds deterministically without preprocessing.*

Corollary 1.2 is an improvement over the previous deterministic expander routing algorithm of [8], which costs $\text{poly}(\phi^{-1}) \cdot 2^{O(\log^{2/3} n \log^{1/3} \log n)}$ rounds.

Expander routing is extremely useful as a fundamental *communication primitive* in designing distributed algorithms in expander graphs. Expander routing has been used to design efficient MST and minimum cut algorithms [16], efficient subgraph finding algorithms [7], and efficient algorithms for sorting, top-k frequent elements, and various data summarization tasks [31] in expander graphs. Expander routing allows us to transform a large class of work-efficient PRAM algorithms into CONGEST algorithms with small overhead [17]. Expander routing has also been utilized in a smooth analysis for distributed MST [9] and to design sparsity-aware algorithms for various shortest path computation tasks in the CONGEST model [4].

Our improved expander routing algorithm immediately leads to improved deterministic upper bounds for all of the aforementioned applications. In particular, our result implies that an MST of an ϕ-expander can be computed in $\text{poly}(\phi^{-1}) \cdot 2^{O(\sqrt{\log n \log \log n})}$ rounds deterministically, improving upon the previous deterministic bound $\text{poly}(\phi^{-1}) \cdot 2^{O(\log^{2/3} n \log^{1/3} \log n)}$ [8] and nearly matching the current randomized bound $\text{poly}(\phi^{-1}) \cdot 2^{O(\sqrt{\log n})}$ [16, 17].

Corollary 1.3. *An MST of an ϕ-expander can be computed in $\text{poly}(\phi^{-1}) \cdot 2^{O(\sqrt{\log n \log \log n})}$ rounds deterministically.*

Proof. Similar to the randomized MST algorithm in [16], it was shown in [8] that an MST can be constructed using polylogarithmic deterministic rounds and invocations of expander routing. Therefore, an MST of an ϕ-expander can be computed in $\text{poly}(\phi^{-1}) \cdot 2^{O(\sqrt{\log n \log \log n})}$ rounds deterministically by implementing the MST algorithm using the expander routing algorithm of Corollary 1.2. □

Expander routing is also useful in designing distributed algorithms in *general graphs* indirectly via the use of *expander decompositions*. An (ϵ, ϕ) expander decomposition of a graph removes at most ϵ fraction of the edges in such a way that each remaining connected component induces a ϕ-expander. In the CONGEST model, this decomposition is commonly applied in a divide-and-conquer approach, where efficient expander routing algorithms are employed to solve subproblems within ϕ-expanders. This approach has been particularly successful in the area of distributed subgraph finding [1–3, 5, 7, 8, 12, 23, 26]. A different use of expander decompositions and routing is to establish barrier for proving lower bounds in CONGEST [12].

Again, our improved deterministic expander routing algorithm leads to improved bounds for the aforementioned applications. In particular, we obtain a near-optimal $\tilde{O}(n^{1-2/k})$-round deterministic algorithm for k-clique enumeration in general graphs, improving the previous deterministic upper bound $n^{1-2/k+o(1)}$ [5].

Corollary 1.4. *There is a deterministic algorithm that list all k-cliques in $\tilde{O}(n^{1-2/k})$ rounds deterministically.*

Proof. By slightly modifying the algorithm of [5], we know that all k-cliques can be listed using $\tilde{O}(n^{1-2/k})$ deterministic rounds and invocations of expander routing on ϕ-expanders with $\phi = 1/\text{polylog}(n)$. The modification needed is to alter the parameters for the deterministic (ϵ, ϕ) expander decomposition in [5, Theorem 5]. Here we want to make $\phi = 1/\text{polylog}(n)$.

As discussed in [8], the deterministic (ϵ, ϕ) expander decomposition algorithm admits the following tradeoff: for any $1 \geq \gamma \geq \sqrt{\log \log n / \log n}$, there is a deterministic expander decomposition algorithm with round complexity $\epsilon^{-O(1)} \cdot n^{O(\gamma)}$ with parameter $\phi = \epsilon^{O(1)} \log^{-O(1/\gamma)} n$. In the k-clique enumeration

algorithm of [5], the parameter ϵ is set to be some constant. By selecting γ to be a sufficiently large constant, we can ensure that $\phi = \epsilon^{O(1)} \log^{-O(1/\gamma)} n = 1/\text{polylog}(n)$ and the round complexity $\epsilon^{-O(1)} \cdot n^{O(\gamma)}$ for constructing the decomposition is upper bounded by $\tilde{O}(n^{1-2/k})$.

If we implement the k-clique enumeration algorithm with the $\text{poly}(\phi^{-1}) \cdot 2^{O(\sqrt{\log n \log\log n})}$-round deterministic expander routing algorithm of [8], then the overall round complexity for k-clique enumeration is $\tilde{O}(n^{1-2/k}) \cdot 2^{O(\sqrt{\log n \log\log n})} = n^{1-2/k+o(1)}$. To improve the upper bound to $\tilde{O}(n^{1-2/k})$, we use our new deterministic expander routing algorithm. Specifically, by selecting ϵ to be a sufficiently small constant in Theorem 1.1, we can ensure that each routing instance can be solved in $\text{poly}(\phi^{-1}) \cdot (\log n)^{O(1/\epsilon)} = \text{polylog}(n)$ rounds and the cost $n^{O(\epsilon)} + \text{poly}(\phi^{-1}) \cdot (\log n)^{O(1/\epsilon)}$ of the preprocessing step is upper bounded by $\tilde{O}(n^{1-2/k})$. □

Our algorithm is optimal up to a polylogarithmic factor, as the upper bound $\tilde{O}(n^{1-2/k})$ for k-clique enumeration in Corollary 1.4 matches the $\tilde{\Omega}(n^{1-2/k})$ lower bound [13, 22]. Previously, such an upper bound was only known to be achievable in the randomized setting [1]. Moreover, for $k = 4$, our algorithm is tight even for the easier k-clique *detection* problem, due to the $\tilde{\Omega}(\sqrt{n})$ 4-clique detection lower bound of [11].

Theorem 1.1 and Corollary 1.4 resolve an open question of Censor-Hillel,[1] which asks whether the cost of each instance of expander routing in the triangle enumeration algorithm can be made both *deterministic* and has a *polylogarithmic* round complexity. Corollary 1.4 yields a deterministic triangle enumeration algorithm that is optimal up to a polylogarithmic factor.

1.2 Previous Results and Key Challenges

For ease of discussion, in this section, we assume that our input graph has an $O(1)$ maximum degree and is an expander with constant conductance.

Randomized Approach. We first summarize at a high level the general idea of [16] and explain the difficulty of de-randomization. Roughly speaking, the general idea is to partition the current base graph X into $k = n^{\epsilon}$ parts $X_1, \ldots, X_k$ with roughly equal sizes. For each part X_i, by using random walk techniques, they embed a virtual Erdős–Renyi graph $G(|X_i|, p)$ onto it for $p = O(\log n/|X_i|)$, where all the virtual edges correspond to a set of paths $\mathscr{P}$ with $\text{polylog}(n)$ *congestion* and *dilation* in X, where the congestion c is defined to be $c = \max_e |\{P \ni e \mid P \in \mathscr{P}\}|$ and the dilation d is defined to be $d = \max_{P \in \mathscr{P}} |P|$. The quantity $c + d$ is known as the quality of $\mathscr{P}$ or the quality of the embedding, as one round of communication in the virtual graph can be simulated within $O(cd)$ rounds in the base graph deterministically, and $\tilde{O}(c + d)$ rounds with randomization [14, 27]. As Erdős–Renyi graphs are good expanders, they may recurse on each X_i by viewing the base graph as the virtual graph $G(|X_i|, p))$ to further partition X_i into k parts and embed a $G(n, p)$ on each of them. The hierarchy goes on for $O(1/\epsilon)$ levels. Since each level only incurs a $\text{polylog}(n)$ blow up on the congestion and dilation. A set of paths of subgraphs in any level with quality $c + d$ corresponds to a set of paths in the original graph of quality $(c + d) \cdot \log^{O(1/\epsilon)} n$. With such a hierarchy embedding structure, they showed a routing instance can be routed using paths that consist of edges in the virtual graphs across different levels with quality $\log^{O(1/\epsilon)} n$, which translates to paths of quality $(\log^{O(1/\epsilon)} n)^2 = \log^{O(1/\epsilon)} n$ in the original graph.

Now we examine the deterministic routing algorithm of [8] and address the reasons why it did not obtain the randomized bound and the preprocessing/query tradeoffs.

Challenge I – Speed. At a high level, the deterministic routing algorithm of [8] still follows the same recursive framework used in the randomized algorithm of [16]. While a low-congestion and low-dilation simultaneous embedding of virtual expanders into $X_1, \ldots, X_k$ can be obtained easily by random walks, obtaining such a simultaneous embedding of expanders is much more difficult in the deterministic setting. In [8], low-congestion and low-dilation simultaneous embedding of virtual expanders is computed recursively using an approach similar to that of [10] based on the *cut-matching game* of [24].

We give a brief and informal introduction to how the cut-matching game works. The cut-matching game is a procedure that returns a balanced sparse cut or a low-congestion and low-dilation embedding of a virtual expander. The algorithm works by iteratively finding a sparse cut of the virtual graph and then finding a low-congestion and low-dilation embedding of a large matching between the two parts of the cut. If we cannot obtain a large matching at some stage of the algorithm, then a balanced sparse cut can be obtained. Otherwise, the virtual graph is guaranteed to be an expander. In [8, 10], the implementation of the sparse cut algorithm in the cut-matching game is done recursively with a recursive structure similar to that of [16] where recursion is applied to multiple smaller instances.

Due to the recursive nature of the approach discussed above, the deterministic simultaneous embedding of virtual expanders in [8] has a much worse guarantee compared to the randomized approach of [16]: Specifically, within $(n^{O(\epsilon)} + \log^{O(\log(1/\epsilon))} n)$ rounds, the expanders obtained have conductance of $1/(\log^{O(1/\epsilon)} n)$. As discussed earlier, to build the hierarchical structure needed to solve the routing problem, one has to repeat the process of simultaneous embedding of virtual expanders recursively, and the depth of recursion is $O(1/\epsilon)$. Since each level incurs a blow-up of $\log^{O(1/\epsilon)} n$ factor on the routing quality, from the bottom to the top, it introduces a $\log^{O(1/\epsilon^2)} n$ blow-up in total, as opposed to $\log^{O(1/\epsilon)} n$ in the randomized construction of [16]. By balancing the terms $(n^{O(\epsilon)} + \log^{O(\log(1/\epsilon))} n)$ and $\log^{O(1/\epsilon^2)} n$, it turns out setting $\epsilon = (\log\log n/\log n)^{1/3}$ yields the best possible bound of $2^{O(\log^{2/3} \cdot \log^{1/3} \log n)}$, which is sub-optimal compared to the randomized algorithms of [16].

Challenge II – Preprocessing/Query Tradeoffs. In the randomized routing algorithm of [16], it is possible to obtain a preprocessing/query tradeoff, where the preprocessing phase builds the hierarchy of expander embeddings in $(n^{O(\epsilon)} + \log^{O(\log(1/\epsilon))} n)$ rounds. Each routing query in the query phase can be done in $\log^{O(\log(1/\epsilon))} n$ rounds. Very different from the randomized approach, the deterministic routing algorithm of [8] still requires

[1] Open Problem 2.2 of https://arxiv.org/abs/2203.06597v3.

$(n^{O(\epsilon)} + \log^{O(\log(1/\epsilon^2))} n)$ rounds for every routing query, so a tradeoff between preprocessing and query cannot be achieved.

We briefly explain why the disparity occurs. In the randomized setting [16], the *same* collection of routing paths constructed in the preprocessing step can be reused for *all* subsequent routing requests that are *oblivious* to the randomness used in the preprocessing step. Such an oblivious assumption can be made without loss of generality by first using random walks to redistribute the messages to be routed. In the deterministic setting [8], the paths for routing the messages are recomputed from scratch for each routing request, as we explain below.

Suppose the current base graph is X. Let $X_1 \dots X_k$ be the children of X in the hierarchy. We classify the tokens needed to be routed $T_1 \dots T_k$ based on their destinations, where T_i is the set of tokens whose destinations are in X_i. The routing task of the current level of recursion is to route all the tokens T_i to X_i. Once such a task has been achieved, we can just recurse in each X_i. The deterministic algorithm of [8] resolves this task by iterating over all the $O(k^2)$ X_i-X_j pairs sequentially. For each X_i-X_j pair, they find a set of paths to send the tokens T_j from X_i to X_j with quality $\text{poly}(k) \cdot 2^{O(\sqrt{\log n})}$ by adapting the *maximal paths* algorithm in [18], which were originally used to compute matching and DFS in PRAM. As a result, there is a $\text{poly}(k) = n^{O(\epsilon)}$ dependency on the query complexity, which is not needed in the randomized algorithm of [16].

1.3 Our Approach

We describe how we overcome the above two challenges as follows. First, to get the bound that matches the randomized algorithm of [16], we do a one-shot hierarchical decomposition.

One-Shot Hierarchical Decomposition. Instead of applying the deterministic simultaneous expander embedding framework [8] as a black box and recursing on each embedded expander to build the embedding hierarchy, we observe that for the algorithm of [8] to return such an embedding of expanders in one level, the algorithm already builds some kind of a hierarchy of expander embedding during the recursive construction. Therefore, a natural idea for improving the deterministic routing algorithm of [8] is to run the simultaneous expander embedding algorithm **only once** in the base level and use the hierarchical decomposition constructed in the algorithm to solve the routing problem in a way similar to that of [8, 16]. To realize this idea, we need to overcome some technical difficulties. In particular, here each level in the hierarchy not only introduces a loss in the conductance guarantee but also a loss in the number of vertices covered by the expander embedding, as the hierarchical decomposition only embeds expanders on a constant fraction of vertices in each level. One observation of why such an approach is still plausible is that the depth of the hierarchy is $O(1/\epsilon)$, so the expanders at the bottom level consist of $1/2^{O(1/\epsilon)}$ fraction of the vertices. Therefore, it might be possible to find delegates in those bottom-level expanders, which we will refer to as the *best nodes*, for every vertex in the original graph in such a way that each best node represents at most $2^{O(1/\epsilon)}$ vertices. This would incur at most $2^{O(1/\epsilon)}$ blow up on the congestion. Moreover, the edges in the virtual expanders in each level of the hierarchy correspond to paths of quality at most $\text{polylog}(n)$ in the parent level. The total blow up on the quality is at most $(\log^{O(1/\epsilon)} n)^2 = \log^{O(1/\epsilon)} n$. This is in contrast with the algorithm of [8], which has a blow-up of $\log^{O(1/\epsilon^2)} n$.

We define additional tasks and reduce the original problem to these tasks to implement the delegation idea. However, for the ease of illustration in the introduction, let us assume for now a base graph X is a partition into $X_1 \dots X_k$, where an expander can be embedded into each X_i. Also, the hierarchy has been constructed recursively on the expander of each X_i.

A Randomized, Meeting in the Middle Approach. Second, to achieve a preprocessing/query tradeoff, given base graph X, we need a routing algorithm that has no polynomial dependencies on k that routes the tokens to the corresponding parts. We first describe a randomized version of our approach and explain how to de-randomize it: Perform lazy random walks simultaneously for all the tokens together until they mix. For tokens destined to X_i (call these tokens T_i), they are now roughly equally distributed across different parts. Suppose that we call such a configuration the *dispersed configuration* and the desired configuration the *final configuration*. To route from the dispersed configuration to the final configuration, we start with the final configuration, transform it into the dispersed configuration by the same method, and reverse the paths. The only problem left now is that the two dispersed configurations can be different, and we still need to match up T_i tokens with T_i' tokens for each i inside each part X_j. Here, we can then embed a sorting network into each X_j to sort the tokens so they are aligned to match up (see the Expander Sorting paragraph at the end of the section).

De-randomization by Pre-embeddings of Shufflers. Now the only issue left is to remove the randomness needed in the process of routing tokens from any configuration to a dispersed configuration. The cut-matching game, introduced by [25], is a potential deterministic way to achieve a similar effect of random walks. Roughly speaking, the goal of the game is to produce *a shuffler*, which consists of matchings of virtual edges $M^1, M^2, \dots, M^\lambda$ such that the natural random walk on the sequence of matchings converges to a nearly uniform distribution from any initial distribution, where each M^r corresponds to a set of paths of low congestion and dilation (i.e. if $(u, v) \in M^r$ then there is a u-v path in the set). The natural random walk defined by $M^1, M^2, \dots, M^\lambda$ is a random walk such that for $r = 1, \dots, \lambda$, if the current vertex v is matched to u then we move to u (through its corresponding path) with probability $1/2$, and stay at v with probability $1/2$. If v is not matched, then it stays at v.

Once we have such a shuffler, we can distribute the tokens deterministically according to the behavior of a lazy random walk. In particular, at each node u, consider if the number of T_i-tokens that are on u is x_i. For $r = 1 \dots, \lambda$, if u is matched to v in M^r, we need to send $x_i/2$ T_i-token from u to its mate v. Assuming the tokens are splittable (to be fractional). In the end, every node would hold a roughly equal amount of T_i tokens due to the mixing property of the shuffler. This would lead to the dispersed configuration.

Coarse-grained Shufflers. However, the tokens are not splittable. To this end, instead of building a shuffler on X, we build a shuffler on Y, where Y is a multi-graph obtained from X by contracting each X_i. By doing such a coarse-grained shuffling, the rounding

error due to the integrality of the tokens becomes negligible when $|X_i| \gg |Y|$.

Yet, directly running the cut-matching game on Y will lead to insufficient bandwidth for token distribution. If X_j is matched $X_{j'}$ by the matching player, then we need to send $x_i/2$ T_i-tokens from X_j to $X_{j'}$ in the simulation of lazy random walk, where x_i is the number of T_i-tokens on X_j. Since each matched edge only corresponds to one path and it can be the case that $x_i = \omega(1)$, the bandwidth may not be enough.

To resolve this, we implement the cut player on Y and the matching player on X to ensure the matching player finds enough paths. This will lead the algorithm to produce a shuffler consisting of matchings of X along their path embeddings of low congestion and dilation. The matchings of X can be naturally translated to fractional matchings of Y by normalization. We then simulate the token distribution on Y according to these fractional matchings, using the path embeddings in X.

Routing to Shuffler Portals. Once the shufflers are constructed, it will be ready to process queries of routing instances. Recall that paths that correspond to a matching of the shuffler will be used to transport the tokens. The endpoint of such paths is known as *portals*. To route the tokens according to the fractional matchings, the main task is to send them to the corresponding portals so that they can follow the paths to the corresponding parts. For example, suppose there are x_i T_i-tokens on X_j for each i. If according to the fractional matching, we need to send $x_{i,j'}$ tokens to $X_{j'}$ then we need to route these $x_{i,j'}$ tokens to the portals in X_i. The routing tasks stemming from processing a fractional matching now become parallel instances of the routing task on each X_i. The cut-matching games end in $O(\log n)$ iterations. So the problem recurses into $O(\log n)$ of parallel routing instances of the next level. To load balance the tokens over the portals, we again use the expander sorting technique to resolve it without dependency on $\text{poly}(k)$. As a result, a query can be answered without dependency on $\text{poly}(k)$.

Expander Sorting. One particular subroutine—deterministic expander sorting—serves as a core tool in our routing algorithm. It has been used in, e.g., the aforementioned procedure for routing tokens to shuffler portals as well as other procedures such as re-writing token destinations and solving the problem within leaf components.

The goal of expander sorting is to re-distribute all tokens among the vertices such that, if we collect all the tokens from the vertex with the smallest ID to the vertex with the largest ID, these tokens' pre-defined keys are sorted in non-decreasing order. Su and Vu [31] considered a slightly simpler version of the problem where each vertex holds a unique ID from $[1, n]$ and gave a randomized algorithm for it. Here, the IDs can range from $[1, \text{poly}(n)]$. We gave deterministic algorithms for expander sorting along the way and developed several handy tools based on it. For example, gathering and propagating information with custom grouping keys.

The Equivalence Between Routing and Sorting. As a side result, we showed that expander routing and expander sorting tasks are actually *equivalent* up to a polylogarithmic factor, in the sense that if there is a CONGEST algorithm $\mathcal{A}_{route}$ that solves the expander routing problem in $T_{\mathsf{route}}(n, \phi, L)$ rounds, then an expander sorting instance can be solved within $O(\phi^{-1} \log n) + O(\log n) \cdot T_{\mathsf{route}}(n, \phi, L)$ rounds. Conversely, if there is a CONGEST algorithm $\mathcal{A}_{\mathsf{sort}}$ that solves the expander sorting problem in $T_{\mathsf{sort}}(n, \phi, L)$ rounds, then an expander routing instance can be solved within $O(1) \cdot T_{\mathsf{sort}}(n, \phi, 2L)$ rounds. We prove the equivalence in the full version [6].

We believe that the equivalence result is of independent interest and can contribute to the study of the complexity of distributed graph problems in expander graphs. Much like the significance of *network decomposition* in the LOCAL model, expander routing stands out as the only nontrivial technique in the design of distributed graph algorithms on expanders in the CONGEST model. Akin to the theory of P-SLOCAL-completeness developed in [15], an interesting research direction is to explore the possibility of identifying a wide range of fundamental distributed problems on expanders that are equivalent to expander routing.

Due to the page limit, the appendices and some proofs are deferred to the full version of this paper [6].

2 PRELIMINARIES

Let n denote the number of vertices and Δ be the maximum degree. Throughout the paper, we assume our graph has a constant maximum degree, i.e., $\Delta = O(1)$. A reduction from general graphs to constant degree graphs can be found in the full version [6, Appendix E]. We state some definitions and some basic properties here.

Conductance. Consider a graph $G = (V, E)$. Given a vertex set subset S, define $\text{vol}(S) = \sum_{v \in S} \deg(v)$. Let $\delta(S) = \{(u, v) \mid u \in S, v \in V \setminus S\}$. The *conductance* of a cut S and that of a graph G are defined as follows.

$$\Phi(S) = \frac{|\delta(S)|}{\min(\text{vol}(S), \text{vol}(V \setminus S))} \qquad \Phi(G) = \min_{\substack{S \subseteq V \\ S \neq \emptyset \text{ and } S \neq V}} \Phi(S)$$

Sparsity. The *sparsity* of a cut S and that of a graph G are defined as follows.

$$\Psi(S) = \frac{|\delta(S)|}{\min(|S|, |V \setminus S|)} \qquad \Psi(G) = \min_{\substack{S \subseteq V \\ S \neq \emptyset \text{ and } S \neq V}} \Psi(S)$$

We remark that the sparsity $\Psi(G)$ of a graph G is also commonly known as *edge expansion*.

Diameter. Given a graph $G = (V, E)$. For $u, v \in V$, let $\text{dist}_G(u, v)$ denote the distance between u and v in G. The diameter D is defined to be $D(G) = \max_{u,v \in V(G)} \text{dist}_G(u, v)$. The following upper bound on the diameter can be obtained by a standard ball-growing argument:

Fact 2.1. *Let G be a graph with conductance ϕ. The diameter $D(G)$ is upper bounded by $O(\phi^{-1} \log n)$.*

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

- For each $v \in V$, create an expander graph X_v with $\deg(v)$ vertices with $\Delta(X_v) = \Theta(1)$ and $\Phi(X_v) = \Theta(1)$.
- For each $v \in V$, fix an arbitrary ranking of the edges incident to v. Let $r_v(e)$ denotes the rank of e in v. For each edge $e = uv \in E$, add an edge between the $r_u(e)$'th vertex of X_u and the $r_v(e)$'th vertex of X_v.

The expander split will be used to obtain the reduction from general graphs to constant degree graphs in [6, Appendix E]. A key property is that $\Psi(G^\diamond) = \Theta(\Phi(G))$. The proof, as well as more properties on expander split, can be found in [8, Appendix C].

Quality of Paths. Given a set of paths $\mathcal{P}$. The quality of $\mathcal{P}$, $Q(\mathcal{P})$, is defined to be the congestion + dilation of the set of paths. Notice that the smaller this quantity is, the better *quality* we have. Such a notion has been introduced in [20, 21], as there exist randomized algorithms that route along each path simultaneously in $\tilde{O}(Q(\mathcal{P}))$ rounds [14, 27]. In the deterministic setting, it is straightforward to execute the routing in congestion × dilation $\leq Q(\mathcal{P})^2$ rounds by spending congestion rounds per edge on the paths.

Fact 2.2. *Let $\mathcal{P}$ be a set of precomputed routing paths. Sending one token along every path $P \in \mathcal{P}$ simultaneously can be done in deterministic $Q(\mathcal{P})^2$ rounds.*

Embeddings. Given graphs H_1, H_2 with $V(H_1) \subseteq V(H_2)$, an embedding of H_1 into H_2 is a function $f : E(H_1) \to \mathscr{P}(H_2)$ that maps the edges of H_1 to $\mathscr{P}(H_2)$, the set of all paths in H_2. The quality of the embedding $Q(f)$ is defined to be the quality of the set of paths $\bigcup_{e\in E(H_1)} f(e)$. As the vertex set of H_1 is always a subset of $V(H_2)$, we sometimes specify H_1 *only* by its edge set.

For the ease of composition, given an embedding f, we tweak it so that it can map paths in H_1 to paths H_2 by defining $f(e_1, \ldots, e_l) = (f(e_1), \ldots, f(e_l))$ for $(e_1, \ldots, e_l) \in \mathscr{P}(H_1)$. Given an embedding f that embeds H_1 onto H_2 and an embedding g that embeds H_2 onto H_3, $(g \circ f)$ is an embedding of H_1 into H_3.

Given embedding f that embeds H_1 to G_1 and embedding g that embeds H_2 to G_2 with $V(H_1) \cap V(H_2) = \emptyset$, the embedding $(f \cup g) : E(H_1 \cup H_2) \to \mathscr{P}(G_1 \cup G_2)$ is defined to be

$$(f \cup g)(e) = \begin{cases} f(e) & e \in E(H_1) \\ g(e) & e \in E(H_2) \end{cases}$$

Matching Embedding. The following result, developed in [8, 19], allows us to embed a matching between S and T, where S and T are two disjoint subsets:

Lemma 2.3. *Consider a graph $G = (V, E)$ with maximum degree $\Delta = \text{polylog}(n)$ and a parameter $0 < \psi < 1$. Given a set of source vertices S and a set of sink vertices T with $|S| \leq |T|$, there is a deterministic algorithm that finds a cut C and an embedding f_M of a matching M between S and T saturating S with the following requirement in $2^{O(\sqrt{\log n})} \cdot \text{poly}(1/\psi)$ rounds.*

- ***Matching:*** *The embedding f_M has quality $O(\psi^{-2}) \cdot \text{polylog}(n)$.*
- ***Cut:*** *Let $S' \subseteq S$ and $T' \subseteq T$ be the subsets that are not matched by M. If $S' \neq \emptyset$, then C satisfies $S' \subseteq C, T' \subseteq V \setminus C$, and $\Psi(C) \leq \psi$; otherwise $C = \emptyset$.*

3 THE HIERARCHICAL DECOMPOSITION

Consider a constant degree graph $G = (V, E)$. Chang and Saranurak [8] gave an algorithm that either finds a balanced sparse cut C with $\Psi(C) \leq \psi$ and $|C| \geq |V|/4$ or finds a subset of vertices $W \subseteq V$ such that $\Psi(G[W]) \geq \log^{-O(1/\epsilon)} n \cdot \text{poly}(\psi)$ with $|W| \geq (2/3) \cdot |V|$, where $0 < \epsilon < 1$ is a parameter that the running time depends on. In the latter case, it also produces a hierarchical decomposition $\mathcal{T}$, whose property we summarize in Property 3.1. We set $\psi = \Psi(G)/2$ to force it to go into the latter case, as no cut C with $\Psi(C) \leq \psi/2$ can be found.

Property 3.1. *Each node of $\mathcal{T}$ is a vertex set $X \subseteq V$. The root of the tree is a vertex set W with $|W| \geq (2/3) \cdot |V|$. A node of T can be either good or bad. The number of levels $\ell(\mathcal{T})$ in the hierarchy is upper bounded by $O(1/\epsilon)$. Moreover:*

(1) Let $k = |V(G)|^\epsilon$. If a node is good, then it is either terminal or internal. A bad node or a terminal good node has no children. A good internal node X consists of a number of good children $X_1 \ldots X_t$, where $(2/3) \cdot k \leq t \leq k$ and they can be ordered so that $\max_{x\in X_i} \text{ID}(x) \leq \min_{y\in X_{i+1}} \text{ID}(y)$ for $1 \leq i < t$. Moreover, it has the same number of bad children $X'_1 \ldots X'_t$. Let $X^*_i = X_i \cup X'_i$. We have $X = X^*_1 \cup \ldots \cup X^*_t$. There exists $\tau = \Theta(|X|/k)$ such that for each i,

$$\frac{1}{3} \cdot \frac{|X|}{k} \leq |X^*_i| \leq 6 \cdot \frac{|X|}{k} \quad \text{and} \quad \frac{2}{3}(\tau - 1) \leq |X^*_i| \leq 2 \cdot (\tau + 1)$$

(2) Let $p(X)$ denote the parent node of X. If a non-root node X is good then it is also associated with a virtual graph H_X with maximum degree $O(\log n)$ whose vertex set is X, and an embedding f_X that embeds H_X to $H_{p(X)}$. The root X is associated with the virtual graph $H_X = G[X]$ with $f_X(e) = e$.
Suppose $X \in \mathcal{T}$ is a good internal node. Let $X_1 \ldots X_t$ be the good children of X. The embedding $\bigcup_{i=1}^t f_{X_i}$ that embeds $H_{X_1} \cup \ldots \cup H_{X_t}$ onto H_X has quality $\text{polylog}(n) \cdot O(\psi^{-1})$ in X if X is the root, and $\text{polylog}(n)$ otherwise.
In addition, for any good node X, $\Psi(H_X) = \text{poly}(\psi) \cdot \log^{-O(1/\epsilon)} n$ if X is the root and $\Psi(H_X) = \Omega(1/\log^{\Theta(\ell(\mathcal{T})-\ell(X))} n) = \log^{-O(1/\epsilon)} n$ otherwise, where $\ell(X)$ is the level that X is at in the hierarchy (the root has level 0).

(3) Suppose that X is a good internal node. For each H_{X_i}, it can be extended to a virtual graph H^*_i of X^*_i by adding a matching M^*_i between X_i and X'_i to H_{X_i} such that each vertex in X'_i is matched. This also implies $|X'_i| \leq |X_i|$ and so

$$|X_1 \cup \ldots \cup X_t| \geq |X|/2$$

Moreover, there exists an embedding f_{M_X} that embeds $\bigcup_{i=1}^t M^*_i$ onto H_X with quality $\text{polylog}(n)$ if $\ell(X) \geq 1$, and quality of $O(\psi^{-1}) \cdot \text{polylog}(n)$ if $\ell(X) = 0$.

Property 3.1(1) says that every part X^*_i has roughly the same size, with up to a constant factor difference. Property 3.1(2) describes the embedding inside each X_i. Property 3.1(3) describes the embedding between X_i and X'_i. See Figure 1 for an illustration of the decomposition.

Some properties listed above may not be explicitly stated in [8]. Thus, for the sake of completeness we will go over the construction of [8] to verify these properties in the full version [6, Appendix A].

Theorem 3.2 ([8]). *Let G be a constant degree ϕ-expander and $k = n^\epsilon$ be a parameter. Then, there exists a deterministic* CONGEST *algorithm that computes a hierarchical decomposition $\mathcal{T}$ that satisfies Property 3.1 in $\text{poly}(\phi^{-1}) \cdot (n^{O(\epsilon)} + \log^{O(1/\epsilon)} n)$ time.*

Definition 3.3. Let $X \in \mathcal{T}$ be a good node whose level is $\ell(X)$. The *flatten embedding* f^0_X is an embedding that embeds H_X to G,

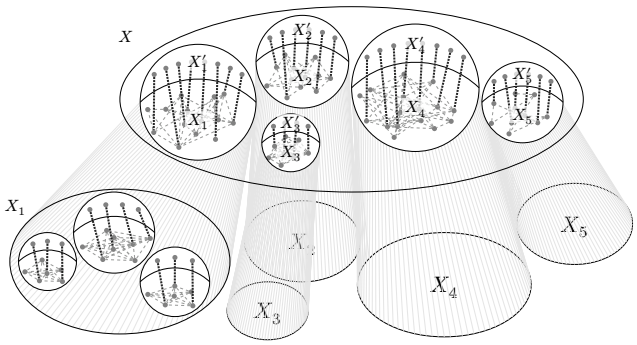

Figure 1: An illustration of the hierarchical decomposition. The gray dotted edges denote the expander embedding as described in Property 3.1(2). For example, the gray dotted edges inside X_1 is the virtual graph H_{X_1}. The base graph of the child node with vertex set X_1 is now H_{X_1}. The black dotted edges between X_i and X'_i form a matching embedding described in Property 3.1(3).

defined as

$$f^0_X = f_{p^{(\ell(X))}(X)} \circ \ldots \circ f_{p^{(2)}(X)} \circ f_{p(X)} \circ f_X$$

Corollary 3.4. *For each $X \in \mathcal{T}$, let $\mathcal{P}_X$ be any collection of paths in X. Suppose that the quality of each $\mathcal{P}_X$ is upper bounded by Q. Let $\mathcal{P}' = \bigcup_{X\in\mathcal{T}} f^0_X(\mathcal{P}_X)$ be the flatten mapping of these paths to G. We have that $Q(\mathcal{P}') = Q \cdot \mathrm{poly}(\psi^{-1}) \cdot \log^{O(1/\epsilon)} n$.*

Proof. Let $\mathcal{T}_i = \{X \in \mathcal{T} \mid \ell(X) = i\}$. Define $f^i = \bigcup_{X\in\mathcal{T}_i} f_X$ to be the union of embedding from level-i nodes to level-$(i-1)$ nodes. By Property 3.1(2), $Q(f^i) = \mathrm{polylog}\, n$ if $i > 1$ and $Q(f^i) = \mathrm{poly}(\psi^{-1}) \cdot \mathrm{polylog}\, n$ otherwise. Since $\bigcup_{X\in\mathcal{T}_i} f^0_X(P_X) = (f^1 \circ \ldots \circ f^i)(\bigcup_{X\in\mathcal{T}_i} \mathcal{P}_X)$, we have $Q(\bigcup_{X\in\mathcal{T}_i} f^0_X(\mathcal{P}_X)) = Q \cdot O(\psi^{-1}) \cdot \log^{O(i)} n$. Summing this over each $i = 1, \ldots, O(1/\epsilon)$, we conclude that the quality of $\mathcal{P}'$ is at most $Q \cdot O(\psi^{-1}) \cdot \log^{O(1/\epsilon)} n$. □

Embedding a Matching To Cover the Whole Graph. We note that the root $W \in \mathcal{T}$ does not cover all the vertices in V. Using Lemma 2.3, we can pre-embed a matching between vertices of $V \setminus W$ and W with good quality so that tokens can be routed to the hierarchy easily.

Lemma 3.5. *Let $W \in \mathcal{T}$ be the root of the hierarchical decomposition. There exists a* CONGEST *algorithm that finds an embedding $f_{M_{root}}$ of a matching M_{root} between $V \setminus W$ and W that saturates $V \setminus W$ with $Q(f_{M_{root}}) = \psi^{-2} \cdot \log^{O(1)} n$ and the runtime is $2^{O(\sqrt{\log n})} \cdot \mathrm{poly}(\psi^{-1})$.*

Proof. Note that $|W| \geq (2/3)|V|$ by Property 3.1. We set $S = W$ and $T = V \setminus W$ and so $|S| < |T|$. We then apply Lemma 2.3 with $\psi = \Psi(G)/2$ so that it returns a matching embedding with the desired quality. □

Leaf Trimming. We will trim the leaves of $\mathcal{T}$ so that every leaf node contains at least $k^4 = O(n^{4\epsilon})$ vertices. This can be done level by level from the last level. Each level takes $O(D' + k^4) \cdot \log^{O(1/\epsilon)} n$ rounds, where $D' = \log^{O(1/\epsilon)} n$ is a diameter upper bound of H_X of each $X \in \mathcal{T}$ in that level.

Definition 3.6. Given $X \in \mathcal{T}$, define $X_{best} \subseteq X$ to be the union of the good leaf nodes in the subtree rooted at X.

Definition 3.7. Define $\rho_{best} = \max_{X\in\mathcal{T}} |X|/|X_{best}|$.

Note that with Property 3.1, we have $\rho_{best} = 2^{O(1/\epsilon)}$.

4 REDUCING TO INTERNAL ROUTING TASKS

We first consider the core setting of the expander routing problem, where the input graph G is a constant degree expander of sparsity ψ. The main task described below summarizes the routing task on G:

*Definition 4.1 (**Task 1**).* Let G be a constant degree ψ-expander, where each vertex of G has a unique destination ID in $\{1, 2, \ldots, n^{O(1)}\}$. Let L be a parameter that depicts the maximum load. Suppose that each node in G holds at most L tokens, and each node is the destination of at most L tokens. The goal is to route the tokens to their destinations.

However, as the leaves (i.e., the best nodes) of our hierarchical decomposition do not cover the whole graph, it would be difficult to solve **Task 1** directly. Instead, we consider a routing problem where all the destinations of the tokens are on the best nodes, specified by their ranks. We introduce Task 2 below:

*Definition 4.2 (**Task 2**).* Let X be a good node in the hierarchical decomposition $\mathcal{T}$ of the input constant degree ψ-expander G. Let L be a parameter. Suppose that each node holds for at most L tokens. Each token z has a *destination marker* i_z and there are at most $L\rho_{best}$ tokens for each destination marker i_z. The goal of the task is to route all tokens with destination marker i_z to the i_z-th smallest vertex among X_{best}.

The reduction from Task 1 to Task 2 is in the full version [6]. Note that as **Task 2** will be solved recursively, we define the task on every component X of the hierarchy $\mathcal{T}$. We will now focus on solving **Task 2** by using the ideas discussed in the introduction. In the following, We identify the key task for solving **Task 2** recursively.

Let $X \in \mathcal{T}$ be an internal component and let $X^*_1, X^*_2, \ldots, X^*_t$ be the parts of X derived from Theorem 3.2. We note that with broadcasts, it is possible for every vertex $v \in X$ obtaining the number of best vertices within all its parts during preprocessing in $O((k + D(H_X)) \cdot Q(\bigcup_{X'\in\mathcal{T}} f^0(H_{X'}))) = \mathrm{poly}(\psi^{-1}, k, \log^{1/\epsilon} n)$ rounds for all $X \in \mathcal{T}$ in parallel. Furthermore, by Property 3.1(1), the IDs of the vertices in X_{best} are partitioned in the sorted order. This allows the algorithm to rewrite the destination marks at the beginning of handling a query on X: For any token z with destination mark i_z, the algorithm computes two values (j_z, i'_z), where $j_z \in \{1, 2, \ldots, t\}$ is the index of the part containing the i_z-th smallest best vertex, and $i'_z = i_z - \sum_{j<j_z} |X_{best} \cap X^*_j|$ is the next-level destination mark.

Therefore, to solve **Task 2** on X, it suffices to first route all tokens z to any vertex in the part $X^*_{j_z}$. Finally, for any part $X^*_j = X_j \cup X'_j$, through Property 3.1(3) we are able to route all tokens X^*_j to X_j such that a next-level **Task 2** can be called. We summarize this task as follows.

*Definition 4.3 (**Task 3**).* Let $X \in \mathcal{T}$ be a non-leaf good node and $X_1^*, X_2^*, \ldots, X_t^*$ be its parts. Each vertex holds at most L tokens for some parameter L. For each token z there is a *part mark* j_z. Suppose that for each $j \in \{1, 2, \ldots, t\}$ there are at most $L \cdot |X_j^*|$ tokens having the same part mark j. The task is accomplished whenever every token z with a part mark j_z is located at a vertex in $X_{j_z}^*$ and each vertex holds at most $2L$ tokens.

In the following sections, we introduce tools and aim to give algorithms for **Task 3**.

5 CORE TOOLS: SHUFFLER AND EXPANDER SORTING

As mentioned in Section 1.3, **Task 3** is solved by routing all tokens into a dispersed configuration. The tokens are routed through a shuffler. In Section 5.1, we describe an algorithm that constructs such a shuffler. Our algorithm implements Räcke, Shah, and Täubig's cut-matching game [29] but with a twist in order to be constructed efficiently. The efficiency comes from the fact that we already have the precomputed hierarchical decomposition $\mathcal{T}$. To process queries using the constructed shuffler, it is necessary to route the tokens to specified *shuffler portals*. This can be implemented by expander sorting procedures. In Section 5.2 we introduce the expander sorting and convenient procedures that can be applied to solving **Task 3**.

5.1 Shuffler

In this subsection, we define shufflers and the algorithm for constructing them during the preprocessing time. Consider a good node $X \in \mathcal{T}$ with $|X| \geq n^{4\epsilon}$. Let $X_1^* \ldots X_t^*$ be a partition of X, where $X_i^* = X_i \cup X_i'$ as defined in Property 3.1.

Cut-Matching Game and Shuffler. To achieve this, we run a variant of cut-matching game [25]. The cut-matching game consists of a cut player and a matching player, and they alternatively pick a cut and add a matching to an initially empty graph. In our variant, the cut-matching game is "played" on the *cluster graph* Y, which is the multigraph by contracting each of the t parts of X. In each iteration q of the cut-matching game, the cut player first obtains a cut on Y, which implies a cut on X.

Then, the matching player finds an embedding $f_{M_X^q}$ of a virtual matching M_X^q on X, which we will show how to transform to a natural *fractional matching* on Y. The sequence of all computed matchings and embeddings $\mathscr{M}_X := ((M_X^1, f_{M_X^1}), (M_X^2, f_{M_X^2}), \ldots, (M_X^\lambda, f_{M_X^\lambda}))$ is then called the *shuffler*. Now, we formally define the aforementioned terms.

Definition 5.1. Let Y denote the cluster graph obtained by contracting each X_i^* in H_X. The set of vertices $V(Y) = \{v_1, v_2, \ldots, v_t\}$ has exactly $|Y| = t$ vertices where v_i corresponds to the vertex contracted from X_i^*. Given a subset $S \subseteq V(Y)$, let S_X denote the corresponding vertex set $\cup_{i:v_i \in S} X_i^*$ in X.

A *fractional matching* $M = \{x_{uv}\}$ of a graph Y is a mapping that maps each unordered pair $\{u, v\} \in \binom{V(Y)}{2}$ to a real number $x_{uv} \in [0, 1]$ such that for all $u \in V(Y)$, the fractional degree is at most one: $\sum_v x_{uv} \leq 1$. Given a matching M_X in X, the corresponding *natural fractional matching* $M = \{x_{uv}\}_{(u,v) \in \binom{V(Y)}{2}}$ is Y of defined to be:

$$x_{uv} = \frac{|\{(a, b) \in M_X \mid a \in X_u^*, b \in X_v^*\}|}{n'}, \text{ where } n' = 6|X|/k.$$

Definition 5.2. Given any fractional matching $M = \{x_{uv}\}$ on Y, we define a $t \times t$ matrix R_M with

$$R_M[i, j] := \begin{cases} \frac{1}{2} + \frac{1}{2} \cdot (1 - \sum_{k \neq i} x_{v_i v_k}) & \text{if } i = j, \\ \frac{1}{2} \cdot x_{v_i v_j} & \text{if } i \neq j. \end{cases}$$

Let $(M^1, \ldots, M^i)$ be a sequence of fractional matchings. If the context is clear, we omit the sequence and denote by R_i the product of matrices $R_{M^i} \cdots R_{M^2} R_{M^1}$. For any vertex $y \in V(Y)$, let $R_i[y]$ be the row vector in R_i that corresponds to the vertex y. For $a, b \in V(Y)$, the b-th entry of $R_i[a]$ can be interpreted as the probability of a random walk that starts from b and ends up at a. It is straightforward to verify that all entries of $R_i[y]$ adds up to 1 for all $y \in V(Y)$.

Let $\mathbf{1}$ be the all-one vector, and for brevity, we denote $\frac{\mathbf{1}}{|Y|} := \frac{1}{|Y|}\mathbf{1}$. The following definition sets up a potential function for the cut-matching game. Let $\| \cdot \|$ to be the standard 2-norm function for a vector.

Definition 5.3 ([25]). Let Y be a cluster graph defined in Definition 5.1 and let $(M^1, \ldots, M^i, \ldots)$ be a sequence of fractional matchings on Y. We define the potential function $\Pi(i) := \sum_{y \in Y(V)} \|R_i[y] - \frac{\mathbf{1}}{|Y|}\|^2$.

Definition 5.4 (Shuffler). Given $X \in \mathcal{T}$, a *shuffler* of X consists of a sequence of λ matching embeddings $\mathscr{M}_X := ((M_X^1, f_{M_X^1}), (M_X^2, f_{M_X^2}), \ldots, (M_X^\lambda, f_{M_X^\lambda}))$ on X such that if $(M_1, \ldots, M^\lambda)$ is the corresponding fractional matching of $(M_X^1, \ldots, M_X^\lambda)$ in Y, the random walk induced by it nearly mixes, as characterized by the following bound on the potential function:

$$\sum_{y \in V(Y)} \left\| R_i[y] - \frac{\mathbf{1}}{|Y|} \right\|^2 \leq \frac{1}{9n^3}$$

In addition, for each i, $1 \leq i \leq \lambda$, the embedding $f_{M_X^i}$ has quality $\log^{O(1/\epsilon)} n$ in X for non-root X, and $\text{poly}(\psi(G)^{-1}) \log^{O(1/\epsilon)} n$ if X is the root. The quality of the shuffler, which essentially has the same order of magnitude as the quality of each embedding $f_{M_X^i}$, is defined to be

$$Q(\mathscr{M}_X) := Q\left(\bigcup_{i=1}^{\lambda} f_{M_X^i}(M_X^i)\right).$$

We prove the following lemma in [6, Appendix B].

Lemma 5.5. *There exists an* CONGEST *algorithm such that, given a good node $X \in \mathcal{T}$, computes a shuffler of X in $\text{poly}(\psi^{-1}, k, \log^{1/\epsilon} n)$ rounds. Moreover, the shuffler has $\lambda = O(\log n)$ matching embeddings with quality $Q(\mathscr{M}_X) = \text{poly}(\psi^{-1}, \log^{1/\epsilon} n)$.*

5.2 Distributed Expander Sorting

In this subsection, we introduce several primitives that are recursively dependent on the internal routing tasks (i.e., **Task 2** and **Task 3**). Perhaps, the most interesting side-product result we obtain is a deterministic sorting algorithm on an expander graph, described as follows.

Theorem 5.6 (Deterministic Expander Sort). *Let $X^* = X \cup X'$ be a virtual graph such that X is a good node in the hierarchical decomposition $\mathcal{T}$ of a ψ-sparsity expander and there is an embedded X'-matching f_M from X' to X with a flattened quality $Q(f_M^0) = \mathrm{poly}(\psi^{-1}, \log^{1/\epsilon} n)$. Suppose that each node holds at most L tokens, and each token z has a (not necessarily unique) key k_z. Then, there exists a* CONGEST *algorithm such that, when the algorithm stops, for any two tokens x and y on two different vertices u and v with $ID(u) < ID(v)$, we have $k_x \le k_y$. Moreover, each vertex holds at most L tokens. The preprocessing time satisfies the following recurrence relations:*

$$T^{pre}_{\mathrm{sort}}(|X^*|) = 2Q(f^0_{M_{root}})^2 + T^{pre}_{\mathrm{sort}}(|X|)$$

$$T^{pre}_{\mathrm{sort}}(|X|) = \begin{cases} T^{pre}_2(|X|) + O(\log n) \cdot T_2(|X|, 1) \\ \qquad + \mathrm{poly}(\psi^{-1}, k, \log^{1/\epsilon} n) & \text{if } X \text{ is non-leaf,} \\ \mathrm{poly}(\psi^{-1}, k, \log^{1/\epsilon} n) & \text{if } X \text{ is a leaf.} \end{cases}$$

The query time satisfies the following recurrence relations:

$$T_{\mathrm{sort}}(|X^*|, L) = 2Q(f^0_{M_{root}})^2 + T^{pre}_{\mathrm{sort}}(|X|)$$

$$T_{\mathrm{sort}}(|X|, L) = \begin{cases} T_3(|X|, L) + L\rho_{best} \cdot Q(\mathcal{I}_{AKS})^2 + L \cdot Q(f^0_{M_X})^2 \\ \qquad + T_{\mathrm{sort}}(6|X|/k, L) & \text{if } X \text{ is non-leaf,} \\ L \cdot \mathrm{poly}(\psi^{-1}, \log^{1/\epsilon} n) & \text{if } X \text{ is a leaf component.} \end{cases}$$

Solving the recurrence relation requires solving the recurrence relations for **Task 2** and **Task 3**, which we defer to the full version of our paper [6].

Applications. The distributed expander sort can be used for the following useful primitives, including token ranking, local propagation, local serialization, and local aggregation. The term *local* here refers to the flexibility of setting an arbitrary *grouping key*, such that the described tasks are performed on each group of tokens independently but simultaneously. We state the results here and provide detailed proofs in the full version [6, Appendix C].

Theorem 5.7 (Token Ranking). *In $O(T_{\mathrm{sort}}(|X^*|, L))$ rounds, each token receives a rank r_z which equals the number of distinct keys that are strictly less than k_z.*

Lemma 5.8 (Local Propagation). *Suppose that each token has a key k_z, a unique tag u_z, and a variable v_z. In $O(T_{\mathrm{sort}}(|X^*|, L))$ rounds, each token's variable is rewritten as v_{z^*} where $z^* = \mathrm{argmin}_x\{u_x \mid k_x = k_z\}$.*

Corollary 5.9 (Local Serialization). *In $O(T_{\mathrm{sort}}(|X^*|, L))$ rounds, each token receives a distinct value $SID_z \in \{0, 1, \ldots, Count(k_z) - 1\}$ among all tokens with the same key. Here $Count(k_z)$ refers to the number of tokens with key k_z.*

Corollary 5.10 (Local Aggregation). *In $O(T_{\mathrm{sort}}(|X^*|, L))$ rounds, each token z learns the value $Count(k_z)$.*

6 SOLVING TASK 2 AND TASK 3 USING SHUFFLERS

In this section, we aim to describe our algorithms for solving **Task 2** and **Task 3**.

Algorithm for Task 2. The algorithm routes the tokens to the corresponding parts, sends the tokens to vertices along the matching toward the good node, and then recurse on the good nodes. For details on solving Task 2 concerning leaf components, we refer readers to the full version [6].

Algorithm for Task 3. To solve Task 3, we use another meet-in-the-middle idea. Suppose that we are able to disperse all tokens with the same part mark as even as possible. Then, the algorithm may apply the same procedure on the instance where $2L$ dummy tokens are created with part mark j on each vertex at X^*_j. Once these dummy tokens are dispersed and meet the real token of the same part mark, a desired routing is found and each dummy token brings *at most one* real tokens to the goal.

Definition 6.1. Let $X \in \mathcal{T}$ be a non-leaf good node with t parts $X^*_1, X^*_2, \ldots, X^*_t$. We say that a configuration is a *dispersed configuration*, if for all $i, j \in \{1, 2, \ldots, t\}$, the number of tokens on vertices of X^*_i having part mark $j_z = j$, denoted as $|T_{i,j}|$, satisfies

$$0.9\frac{N_j}{t} - 0.1\frac{|X|}{t^2} \le |T_{i,j}| \le 1.1\frac{N_j}{t} + 0.1\frac{|X|}{t^2},$$

where N_j is the total number of tokens whose part mark equals j. We say that a configuration is a *final configuration*, if every token with part mark $j_z = j$ is located on a vertex in X^*_j.

Notice that the condition of **Task 3** requires that the number of real tokens with any part mark j is at most $L \cdot |X^*_j|$. The total number of dummy tokens generated from part j is $2L \cdot |X^*_j|$. With the above definition Definition 6.1, if both real tokens and dummy tokens are routed to a dispersed configuration, we will show that on each part X^*_i and for each part mark j the total number of dummy tokens is guaranteed to be outnumbered than the number of real tokens of the same part mark.

Suppose that we have already applied the above idea where the real tokens and the dummy tokens are routed into dispersed configurations. There is a caveat: these tokens may be located at different vertices within the same part X^*_i and they do not meet each other. Therefore, to complete the route, we will have to match the real tokens and the dummy tokens of the same part mark within each X^*_i.

In Section 6.1 and Section 6.2 we show how to transform an arbitrary configuration to a dispersed configuration recursively. Due to the page limit, the rest part of the analysis — including merging two dispersed configurations, solving the leaf case of **Task 2** and round complexity analysis — are deferred to the full version [6].

6.1 Arbitrary Configuration to the Dispersed Configuration

Let $\mathscr{M}_X := ((M^1_X, f_{M^1_X}), (M^2_X, f_{M^2_X}), \ldots, (M^\lambda_X, f_{M^\lambda_X}))$ be the shuffler of X, where $\lambda = O(\log n)$. Let $(M^1, \ldots, M^\lambda)$ be the sequence of corresponding natural fraction matching in Y to the sequence of matching $(M^1_X, M^2_X, \ldots, M^\lambda_X)$. Recall from Definition 5.4 that the random walk $R_{M^\lambda} \ldots R_{M^1}$ converges to nearly uniform distribution from any initial distribution. We will distribute the tokens according to the fraction matching iteration by iteration. That is, in iteration q, we will distribute the tokens according to F_{M^q}.

Consider a fractional matching M in an iteration s. Let $T_{i,l}$ be the T_l tokens at X_i^*. In each iteration, the goal is the following: For each i, j, l, we send $\lfloor (m_{ij}/2)|T_{i,l}| \rfloor$ tokens in $T_{i,l}$ from X_i^* to X_j^*. To achieve this, we will need to first route the $\lfloor (m_{ij}/2)|T_{i,l}| \rfloor$ tokens to the *portals* $P_{i,j}$, where $P_{i,j} \subseteq X_i^*$ is defined as $P_{i,j} = \{x \in X_i^* \mid (x, y) \in M_X^q \text{ for some } y \in X_j^*\}$.

In Section 6.2, we describe how such a task of routing the tokens to the portals can be done recursively. Once the tokens are routed to the portals, they can follow the path embedding corresponding to the virtual matching edge to arrive in X_j^*. The following lemma shows that a dispersed configuration is achieved after doing the token distribution according to $(M^1, \ldots, M^\lambda)$.

Lemma 6.2. *Let $(M^1, \ldots, M^\lambda)$ be the sequence of natural fractional matching in Y that corresponds to the sequence of matching in $\mathscr{M}_X$. For $q = 1 \ldots \lambda$, suppose that during iteration q, we send $\lfloor (m_{ij}/2)|T_{i,l}^{q-1}| \rfloor$ tokens in $T_{i,l}^{q-1}$ from X_i^* to X_j^* for each $1 \le i, j, l \le t$, where $T_{i,l}^{q-1}$ is set of tokens in X_i^* destined to part l at the end of iteration $q-1$. Then, a dispersed configuration is achieved at the end.*

We defer the proof to the full version [6].

The following corollary is useful to control the number of tokens in each part, which can be useful for deriving the final recurrence. Due to its similar flavor, we state it here:

Corollary 6.3. *Let $N_{\max}$ be the maximum number of tokens within any part at the beginning of the routing. For any q such that $1 \le q \le \lambda$, the total number of tokens within the part X_i^* after sending the tokens along fraction matchings $M^1, M^2, \ldots, M^q$ is at most $N_{\max} + t^2 q$.*

Proof. This can be done by an induction on q. Let N_i^q be the number of tokens within the part X_i^* after iteration q and let $N_{\max}^q = \max_i \{N_i^q\}$. Then, for all i we have $N_i^0 \le N_{\max}$.

We notice that each M^q is a fraction matching. If the tokens are fractional as well then the total amount of tokens does not change. However, due to the fact that tokens are integral, we have to carefully upper bound the total number of tokens:

$$\begin{aligned} N_i^q &\le \Bigg(N_i^{q-1} - \underbrace{\sum_{j=1}^{t} \sum_{l=1}^{t} \left\lfloor \frac{1}{2} m_{i,j} |T_{i,l}^{q-1}| \right\rfloor}_{\text{tokens sent away}} \Bigg) + \underbrace{\sum_{j=1}^{t} \sum_{l=1}^{t} \left\lfloor \frac{1}{2} m_{j,i} |T_{j,l}^{q-1}| \right\rfloor}_{\text{tokens received}} \\ &\le \left(\frac{1}{2} N_i^{q-1} + t^2 \right) + \frac{1}{2} N_{\max}^{q-1} \\ &\le N_{\max}^{q-1} + t^2 \,. \end{aligned}$$

□

6.2 Routing the Tokens to the Portals

Consider a particular part X_i^* and a particular fraction matching M^q. In this subsection, we describe a subroutine that routes all tokens on X_i^* to designated portals $P_{i,j}$. The goal of this subroutine is that for all l and for all tokens with the same specific part mark l, there will be exactly $\lfloor (m_{i,j}/2)|T_{i,l}| \rfloor$ tokens being routed to vertices in $P_{i,j}$. Moreover, the tokens routed to $P_{i,j}$ should be load-balanced.

Tie-Breaking The Tokens via Serialization. We would need two tie-breaking operations here: first, for any l, each token with the part mark l learns the portal group index j (or no-op). This can be done by invoking (1) a local aggregation procedure (Corollary 5.10) that obtains the value $|T_{i,l}|$ and (2) a local serialization procedure (Corollary 5.9) that gives each token a serial number in $\{0, 1, \ldots, |T_{i,l}| - 1\}$. With the serial number and the total count, we can now assign locally the portal index j for each token.

To enforce the load-balancedness requirement, the second tie-breaking must be made such that each node in $P_{i,j}$ receives roughly the same number of tokens. This can be done by applying two additional serialization steps. The first local serialization procedure, using the portal index j as the key, assigns each token z that goes to the same portal group $P_{i,j}$ a serial number SID_z. Using the size $|P_{i,j}|$ that is preprocessed and stored at each vertex in X_i^*, each token z obtains an index $\chi(z) := SID_z \bmod |P_{i,j}|$. The second local serialization procedure can actually be preprocessed — it assigns each portal vertex in $P_{i,j}$ a serial number.

We remark that all local aggregation and local serialization procedures described above are actually running over the virtual graph of the corresponding part X_i^*. This avoids cyclic dependency of invoking token ranking, expander sorting, and Task 3.

Now, the problem of routing the tokens to the portals reduces to the following "Task 2 style" task. In this task, each token has a portal group index j and an index $\chi(z)$. The goal is to route all tokens on X_i^* to the $\chi(z)$-th vertex within $P_{i,j}$.

Meet-In-The-Middle Again. The above task can be solved using the meet-in-the-middle trick and running expander sorting (Theorem 5.6) twice within X_i^*. In the first expander sorting, we assign for each token a key $k_z := (j, \chi(z))$ and sort the tokens within X_i^*. In the second expander sorting, for each portal vertex of $P_{i,j}$ with a serial index s, we create a certain number, say $\sigma_{j,s}$, of dummy tokens all with the same key $k_z := (j, s)$. The number of dummy tokens for (j, s) will be exactly the same as the number of actual tokens that will be sent to this vertex. We also add dummy tokens such that every vertex reaches the same maximum load of L tokens. These two expander sortings should now give a perfect match between the actual tokens and the dummy tokens. Thus, by reverting the routes of dummy tokens, each dummy token brings one actual token to the desired destination.

Obtaining the value $\sigma_{j,s}$ is again can be done by running a local aggregation (Corollary 5.10) over the instance where besides actual tokens, each portal vertex also creates one dummy token of the same key. After running the local aggregation, this token learns the total count $\sigma_{j,s} + 1$ and goes back to the actual portal vertex.

ACKNOWLEDGMENTS

Yi-Jun Chang is supported by the NUS Presidential Young Professorship startup grant. Shang-En Huang and Hsin-Hao Su are supported by NSF CCF-2008422. The authors thank the anonymous reviewers for their helpful comments.

REFERENCES

[1] Keren Censor-Hillel, Yi-Jun Chang, François Le Gall, and Dean Leitersdorf. 2021. Tight distributed listing of cliques. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2878–2891.

[2] Keren Censor-Hillel, Orr Fischer, François Le Gall, Dean Leitersdorf, and Rotem Oshman. 2022. Quantum Distributed Algorithms for Detection of Cliques. In *Proceedings of the 13th Innovations in Theoretical Computer Science Conference (ITCS) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 215)*, Mark

Braverman (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 35:1–35:25. https://doi.org/10.4230/LIPIcs.ITCS.2022.35

[3] Keren Censor-Hillel, François Le Gall, and Dean Leitersdorf. 2020. On distributed listing of cliques. In *Proceedings of the 39th Symposium on Principles of Distributed Computing (PODC)*. 474–482.

[4] Keren Censor-Hillel, Dean Leitersdorf, and Volodymyr Polosukhin. 2021. On sparsity awareness in distributed computations. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*. 151–161.

[5] Keren Censor-Hillel, Dean Leitersdorf, and David Vulakh. 2022. Deterministic near-optimal distributed listing of cliques. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing (PODC)*. 271–280.

[6] Yi-Jun Chang, Shang-En Huang, and Hsin-Hao Su. 2024. Deterministic Expander Routing: Faster and More Versatile. https://doi.org/10.48550/arXiv.2405.03908 arXiv:2405.03908 [cs.DC] arXiv:2405.03908. Full version of this paper..

[7] Yi-Jun Chang, Seth Pettie, Thatchaphol Saranurak, and Hengjie Zhang. 2021. Near-optimal Distributed Triangle Enumeration via Expander Decompositions. *J. ACM* 68, 3 (2021).

[8] 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)*.

[9] Soumyottam Chatterjee, Gopal Pandurangan, and Nguyen Dinh Pham. 2020. Distributed MST: A smoothed analysis. In *Proceedings of the 21st International Conference on Distributed Computing and Networking*. 1–10.

[10] Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. 2020. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *Proceedings of the 61st Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 1158–1167.

[11] Artur Czumaj and Christian Konrad. 2020. Detecting cliques in CONGEST networks. *Distributed Computing* 33, 6 (2020), 533–543.

[12] Talya Eden, Nimrod Fiat, Orr Fischer, Fabian Kuhn, and Rotem Oshman. 2022. Sublinear-time distributed algorithms for detecting small cliques and even cycles. *Distributed Computing* (2022), 1–28.

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

[14] Mohsen Ghaffari. 2015. Near-Optimal Scheduling of Distributed Algorithms. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 3–12.

[15] Mohsen Ghaffari, Fabian Kuhn, and Yannic Maus. 2017. On the complexity of local distributed graph problems. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*. 784–797.

[16] Mohsen Ghaffari, Fabian Kuhn, and Hsin-Hao Su. 2017. Distributed MST and Routing in Almost Mixing Time. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 131–140.

[17] Mohsen Ghaffari and Jason Li. 2018. New Distributed Algorithms in Almost Mixing Time via Transformations from Parallel Algorithms. In *32nd International Symposium on Distributed Computing (DISC) (LIPIcs, Vol. 121)*. 31:1–31:16.

[18] Andrew V. Goldberg, Serge A. Plotkin, and Pravin M. Vaidya. 1993. Sublinear-Time Parallel Algorithms for Matching and Related Problems. *J. Algor.* 14, 2 (1993), 180–213.

[19] Bernhard Haeupler, D. Ellis Hershkowitz, and Thatchaphol Saranurak. 2023. Maximum Length-Constrained Flows and Disjoint Paths: Distributed, Deterministic, and Fast. In *Proc. 55th Annual ACM Symposium on Theory of Computing (STOC)*. 1371–1383.

[20] Bernhard Haeupler, Harald Räcke, and Mohsen Ghaffari. 2022. Hop-constrained expander decompositions, oblivious routing, and distributed universal optimality. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2022)*. 1325–1338. https://doi.org/10.1145/3519935.3520026

[21] Bernhard Haeupler, David Wajc, and Goran Zuzic. 2021. Universally-optimal distributed algorithms for known topologies. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing*. 1166–1179. https://doi.org/10.1145/3406325.3451081

[22] 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)*. 381–389. https://doi.org/10.1145/3087801.3087811

[23] 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)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[24] Rohit Khandekar, Subhash Khot, Lorenzo Orecchia, and Nisheeth K Vishnoi. 2007. *On a cut-matching game for the sparsest cut problem.* Technical Report UCB/EECS-2007-177. Univ. California, Berkeley.

[25] Rohit Khandekar, Satish Rao, and Umesh Vazirani. 2009. Graph partitioning using single commodity flows. *J. ACM* 56, 4, Article 19 (jul 2009), 15 pages.

[26] François Le Gall and Masayuki Miyamoto. 2021. Lower Bounds for Induced Cycle Detection in Distributed Computing. In *Proceedings of the 32nd International Symposium on Algorithms and Computation (ISAAC) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 212)*, Hee-Kap Ahn and Kunihiko Sadakane (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 58:1–58:19. https://doi.org/10.4230/LIPIcs.ISAAC.2021.58

[27] Frank Thomson Leighton, Bruce M. Maggs, and Satish Rao. 1994. Packet Routing and Job-Shop Scheduling in *O*(Congestion + Dilation) Steps. *Comb.* 14, 2 (1994), 167–186.

[28] 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.

[29] Harald Räcke, Chintan Shah, and Hanjo Täubig. 2014. Computing Cut-Based Hierarchical Decompositions in Almost Linear Time. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 227–238.

[30] 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.

[31] Hsin-Hao Su and Hoa T. Vu. 2019. Distributed Data Summarization in Well-Connected Networks. In *33rd International Symposium on Distributed Computing (DISC 2019) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 146)*, Jukka Suomela (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 33:1–33:16. https://doi.org/10.4230/LIPIcs.DISC.2019.33