Minor Sparsifiers and the Distributed Laplacian Paradigm*

Sebastian Forster† Gramoz Goranci‡ Yang P. Liu§ Richard Peng¶ Xiaorui Sun‖
Mingquan Ye**

Abstract

We study distributed algorithms built around minor-based vertex sparsifiers, and give the first algorithm in the CONGEST model for solving linear systems in graph Laplacian matrices to high accuracy. Our Laplacian solver has a round complexity of $O(n^{o(1)}(\sqrt{n}+D))$, and thus almost matches the lower bound of $\widetilde{\Omega}(\sqrt{n}+D)$, where n is the number of nodes in the network and D is its diameter.

We show that our distributed solver yields new sublinear round algorithms for several cornerstone problems in combinatorial optimization. This is achieved by leveraging the powerful algorithmic framework of Interior Point Methods (IPMs) and the Laplacian paradigm in the context of distributed graph algorithms, which entails numerically solving optimization problems on graphs via a series of Laplacian systems. Problems that benefit from our distributed algorithmic paradigm include exact mincost flow, negative weight shortest paths, maxflow, and bipartite matching on sparse directed graphs. For the maxflow problem, this is the first exact distributed algorithm that applies to directed graphs, while the previous work by [Ghaffari et al. SICOMP'18] considered the approximate setting and works only for undirected graphs. For the mincost flow and the negative weight shortest path problems, our results constitute the first exact distributed algorithms running in a sublinear number of rounds. Given that the hybrid between IPMs and the Laplacian paradigm has proven useful for tackling numerous optimization problems in the centralized setting, we believe that our distributed solver will find future applications.

At the heart of our distributed Laplacian solver is the notion of spectral subspace sparsifiers of [Li, Schild FOCS'18]. We present a nontrivial distributed implementation of their construction by (i) giving a parallel variant of their algorithm that avoids the sampling of random spanning trees and uses approximate leverage scores instead, and (ii) showing that the algorithm still produces a high-quality subspace spectral sparsifier by carefully setting up and analyzing matrix martingales. Combining this vertex reduction recursively with both tree and elimination-based preconditioners leads to our algorithm for solving Laplacian systems. The construction of the elimination-based preconditioners is based on computing short random walks, and we introduce a new technique for reducing the congestion incurred by the simulation of these walks on weighted graphs.

*Accepted to the 62nd Annual Symposium on Foundations of Computer Science (FOCS 2021)
†University of Salzburg, Austria.
‡University of Toronto, Canada.
§Stanford University, USA.
¶Georgia Institute of Technology, USA.
‖University of Illinois at Chicago, USA.
**University of Illinois at Chicago, USA.

Contents

1 Introduction

The steady growth of data makes it increasingly important to control and reduce the communication of algorithms. The CONGEST model [Pel00] is a widely studied model for low communication algorithms on large graphs and sparse matrices. In this model, each vertex/variable occupies a separate machine, and communicates in synchronous rounds by sending messages of length $O(\log n)$ to its neighbors given by the edges of the underlying graph. This bandwidth restriction implies a polynomial lower bound in the round complexity for many fundamental graph problems [PR00, Elk06, DSHK+12]. While early work on efficient algorithms in this model has focused on the minimum spanning tree problem [GHS83, GKP98, KP98], extensive work over the past few years has led to efficient algorithms for several more fundamental graph problems, such as approximate and exact single-source shortest paths [Nan14, HKN16, Elk20, GL18, FN18, CM20], approximate and exact all-pairs shortest paths [HW12, HNS17, ARKP18, EN18, LPP19, AR19, BN19, AR20], approximate and exact minimum cut [GK13, NS14, DHNS19, GNT20, DEMN21], approximate maximum flow [GKK+15], bipartite maximum matching [AKO18], triangle counting [IG17, CPZ19, CS19], and single-source reachability [GU15, JLS19].

A major development in sequential and parallel graph algorithms is the development of hybrid algorithms that combine numerical and combinatorial building blocks. This line of work was initiated by the seminal work of Spielman and Teng [ST14], which showed that a Laplacian linear system on a graph can be solved in nearly linear time. Here, the Laplacian of a weighted undirected graph $G = (V, E)$ is defined as $\boldsymbol{L}(G) = \boldsymbol{D}(G) - \boldsymbol{A}(G)$, where $\boldsymbol{D}(G)$ is the diagonal weighted degree matrix, and $\boldsymbol{A}(G)$ is the weighted adjacency matrix. Equivalently, if $\vec{w} \in \mathbb{R}^m_{>0}$ are the edge weights,

$$\boldsymbol{L}(G)_{uv} = \begin{cases} \sum_{(u,z)\in E} \vec{w}_{uz} & \text{if } u = v, \\ -\vec{w}_{uv} & \text{otherwise.} \end{cases}.$$

Since then, there has been extensive work towards giving more efficient and simpler Laplacian system solvers sequentially [KMP10, KMP11, KOSA13, CKM+14, KS16], as well as parallel versions [PS14, KLP+16]. These results have in turn been used to give the state-of-the-art runtimes for a variety of graph problems, including exact maximum flows, bipartite matchings, and mincost flows [Mad16, LS20b, LS20a, CMSV17, AMV20, vdBLN+20], approximate maximum flows [KLOS14, She13, Pen16], and approximate parallel shortest paths [Li20, ASZ20]. Ideas from the latter works have found application in the distributed setting, giving nearly optimal algorithms for approximate maxflows [GKK+15] and approximate single-source shortest paths [BKKL17] in the CONGEST model.

Our main result is an algorithm for solving graph Laplacian linear systems in the CONGEST model in $O(n^{o(1)}(\sqrt{n} + D))$ rounds (Theorem 1), where n is the number of nodes in the underling graph and D is its diameter. This nearly matches a lower bound of $\widetilde{\Omega}(\sqrt{n} + D)$, which we show for completeness in Appendix A by reduction to [DSHK+12].

Theorem 1. *There is an algorithm in the* CONGEST *model that on a weighted graph $G = (V, E, \vec{w})$ with n vertices and diameter D, vector $\vec{b}$ on the vertices of G, and error $\epsilon < 0.1$, produces in $O(n^{o(1)}(n^{1/2} + D)\log(1/\epsilon))$ rounds a vector $\vec{x}$ distributed over the vertices such that*

$$\left\|\vec{x} - \boldsymbol{L}(G)^{\dagger}\vec{b}\right\|_{\boldsymbol{L}(G)} \leq \epsilon \cdot \left\|\vec{b}\right\|_{\boldsymbol{L}(G)}.$$

Theorem 2. *In the* CONGEST *model of computation, solving Laplacian systems to accuracy $\epsilon \leq \frac{1}{2}$ requires at least $\widetilde{\Omega}(n^{1/2} + D)$ rounds of communication.*

We give several applications towards designing hybrid algorithms for graph problems in the CONGEST model. Specifically, by combining our Laplacian solver with interior point methods [Mad16, CMSV17], we obtain the first algorithms for exact computation of maximum flows, bipartite matchings, and negative-weight shortest paths that run in a sublinear number of rounds in the CONGEST model on sparse graphs (Section 7).

At a high level, we build our CONGEST model algorithm by first building a parallel/PRAM algorithm for solving Laplacian systems that only works with minors[1] of the original graph, and show that one round of communication necessary in our algorithm (such as matrix vector multiplication) between neighbors on a minor can be simulated in the original graph in $\tilde{O}(\sqrt{n}+D)$ rounds. Previous methods for computing low stretch spanning trees and approximate maximum flows [GKK+15] use a similar notion of considering a graph on clusters of nodes in the original graph, however – to the best of our knowledge – we are the first to work explicitly with the notion of minors. We are optimistic that our approach based on minor vertex sparsifiers may provide a general framework for designing CONGEST model algorithms with near optimal complexities.

The main backbone of our algorithm for solving Laplacian systems that works with minors only is the parallel Laplacian solver of [KLP+16]. This solver relies on sparse spectral approximations of the Schur complements of an $n \times n$ matrix, which can be thought of as a smaller matrix that preserves the solutions of linear systems on a subset of coordinates in $[n]$. At a high level, the algorithm eliminates onto (sparse) Schur complements of the original graph while adding edges, leading to graphs that are not minors of the original graph. To resolve this, a major contribution of this paper is an efficient parallel algorithm to construct a spectral sparsifier for a Schur complement which is a minor of the original graph. While the existence of such a minor spectral sparsifier was known [LS18], the algorithm required sampling a random spanning tree, and hence could not be implemented in parallel. We instead show that a large batch of edges may be independently sampled at the same time using leverage scores (Definition 2.5), providing an arguably simpler and more direct analysis than [LS18].

1.1 Applications to Flow Problems

We briefly discuss how our Laplacian solver can be applied to achieve results on maximum flow, bipartite matching, mincost flows, and negative weight shortest paths, and compare to previous complexities. We achieve our bounds by combining our Laplacian system solver in Theorem 1 with recent interior point methods of [Mad16, CMSV17].

For unit capacity graphs, the runtimes we achieve in Theorems 7, 8, and 9 for the maximum flow problem, mincost flow, and negative weight shortest path problems are

$$O(m^{3/7+o(1)}(n^{1/2}D^{1/4}+D)).$$

For sparse unweighted graphs with $m = O(n)$, and polynomially small diameter $D = n^{2/7-\Omega(1)}$, the algorithms in Theorems 7, 8, and 9 run in a *sublinear* number of rounds, i.e. $n^{1-\Omega(1)}$ rounds. To our knowledge, these are the first exact sublinear round algorithms for unit maximum flows, bipartite matchings, and negative weight shortest paths for any regime of diameter D. Our distributed maximum flow algorithm extends to directed graphs, while the previous work by Ghaffari et al. [GKK+15] considered the approximate setting and works only for undirected graphs. In fact, for the maximum flow problem, our results

[1] In fact, our algorithm deals with ρ-minors (Definition 4.1), which can be thought of as minors with congestion ρ, where $\rho \geq 1$ is a parameter. However, for the sake of simplicity, we refer to them as minors throughout the informal discussions of our techniques in the introduction and overview.

are – to the best of our knowledge – the fastest known in the low-diameter regime; see Section 1.2 for further discussion.

At a high level, our runtime comes from two pieces. The results of [Mad16, CMSV17] show that in $\widetilde{O}(m^{3/7})$ rounds of an interior point method, in each round which involves solving a Laplacian system on the underlying graph with edge weights/resistances, we can reduce the amount of residual flow to $\widetilde{O}(m^{3/7})$. The residual flow can routed combinatorially with $\widetilde{O}(m^{3/7})$ rounds of an augmenting paths or shortest paths computation. Therefore, the total number of rounds required to implement the interior point method is $O(m^{3/7+o(1)}(n^{1/2}+D))$ using Theorem 1, and the shortest path computations can be done in $\widetilde{O}(m^{3/7}(n^{1/2}D^{1/4}+D))$ rounds using the results of [CM20]. Combining these gives the result.

1.2 Related Work

Distributed Graph Algorithms Previous works in distributed algorithms most related to our result and the corresponding techniques are the algorithms for simulating **random walks** and generating **random spanning trees** [DSNPT13, GB20]. On unweighted, undirected graphs with diameter D, the algorithms by Das Sarma, Nanongkai, Pandurangan, and Tetali [DSNPT13] generate an ℓ-step random walk in $\widetilde{O}(\sqrt{\ell D}+D)$ rounds, and a random spanning tree in $\widetilde{O}(\sqrt{mD})$ rounds, respectively. There are well known connections between sampling a large number of random walks and Laplacian solving [DST17, DGT17]. However, it is not clear how to utilize these methods in the context of our algorithms, since many of the intermediate graph structures we deal with involve dealing with weighted random walks, which in turn leads to congestion issues when trying to simulate these walks in the distributed setting. We discuss how to overcome such obstacles in Section 3.

There has also been work in the distributed setting relating to **spectral graph properties**. This in particular includes distributed sparsification [KX16], PageRank [DSMPU15], Laplacian solvers in well-mixing settings [GB20], and expander decomposition [CPZ19, CS19, CS20].

Continuous optimization methods have been used to give the state-of-the-art distributed algorithms for approximate max-flow [GKK+15] and approximate transshipment [BKKL17]. Note however that these approximation algorithms are tailored to undirected graphs and their running time depends polynomially on $1/\epsilon$ (for a desired accuracy of ϵ). Our max-flow routine also works on directed graphs and only depends polylogarithmically on $1/\epsilon$, which allows for computing a high-accuracy solution and rounding it to an exact one. Furthermore, these prior approaches for ℓ_∞ and ℓ_1-norm minimization, respectively, do not carry over to ℓ_2-norm minimization (as would be needed for solving Laplacian systems) as it is not known how to efficiently sample from a collection of trees when using an ℓ_2 variant of tree-based graph approximations to build oblivious routing schemes.

In addition to these works, there are many papers related to the three problems we solve by applying our distributed Laplacian solver. There have been numerous results on exact and approximate **shortest path** computation in the past decade [HW12, Nan14, HKN16, EN19a, Elk20, HNS17, GL18, ARKP18, EN18, FN18, LPP19, EN19b, AR19, BN19, AR20, CM20]. For the single-source shortest paths (SSSP) problem all of these works assume non-negative or positive edge weights. It is well-known that the SSSP problem in presence of negative edge weights can be solved in $O(n)$ rounds by a variant of the Bellman-Ford algorithm. To the best of our knowledge no algorithm that improves upon this bound has been formulated (or implied) in the CONGEST model so far.

For **distributed computations of maximum flows**, Ghaffari, Karrenbauer, Kuhn, Lenzen, and Patt-Shamir [GKK+15] designed an algorithm that returns an $(1+\epsilon)$-approximation in $O((\sqrt{n}+D)n^{o(1)}\epsilon^{-3})$ rounds. In terms of exact algorithms, we are not aware of any paper claiming a sublinear number of

rounds in the CONGEST model (cf. [GKK+15] for a detailed discussion of maximum flow for other distributed models). To the best of our knowledge, we need to compare ourselves with the following two approaches:

- The problem can trivially be solved in $O(m + D)$ rounds by collecting the whole graph topology in a single node and then solving the problem with internal computation.
- The Ford-Fulkerson algorithm [FF56] takes $|f^*|$ iterations (where $|f^*|$ is the value of a maximum flow) and the running time in each iteration is dominated by the time needed to perform an s-t reachability computation (on a directed graph). The latter problem can be solved in $\tilde{O}(\sqrt{n}D^{1/4} + D)$ [GU15] or $\tilde{O}(\sqrt{n}+n^{1/3+o(1)}D^{2/3})$ rounds, respectively, which yields total running time of $\tilde{O}(|f^*|(\sqrt{n}D^{1/4}+D))$ or $\tilde{O}(|f^*|(\sqrt{n} + n^{1/3+o(1)}D^{2/3}))$ rounds, respectively. In unit-capacity ("unweighted") graphs, where $|f^*| \leq n$, this gives a total running time of $\tilde{O}(n^{3/2}D^{1/4} + nD)$ or $\tilde{O}(n^{3/2} + n^{4/3+o(1)}D^{2/3})$, respectively.

Due to a well-known reduction to maximum flow, the **bipartite maximum matching** problem is intimately connected to the maximum flow problem. In the CONGEST model, the fastest known algorithm for computing a bipartite maximum matching (of an unweighted graph) takes $O(n \log n)$ rounds [AKO18] – more precisely the algorithm takes $O(s^* \log s^*)$ rounds, where s^* is the size of a maximum matching. Obtaining a subquadratic maximum matching algorithm for networks of arbitrary topology is a major open problem [AK20]. In addition, there are numerous works on computing approximate matchings, which are usually based on computing a maximal matching, using the framework of Hopcroft and Karp [HK73], or rounding a fractional matching (cf. [AK20] for an overview on approximate matching algorithms in the CONGEST model).

Laplacian Solvers Our algorithm combines both tree-based ultrasparsification algorithms [ST14, KMP10, KMP11, CKM+14] and elimination-based algorithms that utilize Schur complements [KLP+16, KS16, Kyn17]. Both types of algorithms were originally developed for the sequential model. The issue of round complexity was previously addressed in parallel Laplacian solving [BGK+14, PS14].

We believe that a variant of [BGK+14] tailored to the CONGEST model gives a round complexity of around $n^{3/4} + Dn^{1/4}$ as opposed to the bound in Theorem 1 to because the depth of the parallel algorithm of [BGK+14] is more than polylogarithmic. The polylogarithmic depth parallel algorithm from [PS14] is more difficult to convert to the CONGEST setting because it explicitly adds edges to the graph, which causes increased congestion.

The outer layer recursion of our algorithm is akin to the recursive construction of solvers and preconditioners present in Laplacian solving [Pen13, KLP+16], approximate max-flow [Pen16], and matrix sampling [CP15, CLM+15, CMM17].

Parallel Laplacian solvers and spectral algorithms have also motivated the study of (nearly) log space variants of these algorithms [MRSV17, MRSV19, AKM+20]. It's an intriguing question to formally connect these low space algorithms with distributed algorithms, both of which stem from works on low iteration count algorithms.

Vertex Sparsification Critical to our result is the construction of minor based Schur complements by Li and Schild [LS18]. Minor based sparsification has been studied for distances [CGH16, KNZ14], and implicitly for cuts via hierarchical routing schemes [Räc02, RST14]. A more systematic treatment of uses of such sparsifiers, in dynamic graph algorithms, can be found in [Gor19]. Some of the cut preserving

vertex sparsifiers [Moi09, LM10, CLLM10, EGK+14, KR13], as well as their recent variations in small cut settings [CDK+21] produce either minors or probability distribution over minors.

2 Preliminaries

We start by describing general notation we use throughout the paper.

General notation. Given a symmetric matrix $\boldsymbol{M}$, we let $\|\boldsymbol{M}\|_2 = \max_{\|x\|_2=1} |x^\top \boldsymbol{M} x|$ denote the maximum absolute value of any eigenvalue. For a vector v and matrix $\boldsymbol{M}$, we define $\|v\|_{\boldsymbol{M}} \stackrel{\text{def}}{=} \sqrt{v^\top \boldsymbol{M} v}$. For positive real numbers a, b we say that $a \approx_\epsilon b$ if $\exp(-\epsilon)a \le b \le \exp(\epsilon)a$. We say that a matrix $\boldsymbol{M} \in \mathbb{R}^{n\times n}$ is positive semidefinite if $x^\top \boldsymbol{M} x \ge 0$ for all $x \in \mathbb{R}^n$. For matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, we write $\boldsymbol{A} \preceq \boldsymbol{B}$ if $\boldsymbol{B}-\boldsymbol{A}$ is positive semidefinite. For positive semidefinite matrices $\boldsymbol{A}, \boldsymbol{B}$ we say that $\boldsymbol{A} \approx_\epsilon \boldsymbol{B}$ if $\exp(-\epsilon)\boldsymbol{A} \preceq \boldsymbol{B} \preceq \exp(\epsilon)\boldsymbol{A}$.

Schur complements and Cholesky factorization. Our algorithms are based on Schur complements and sparsified Cholesky factorization. At a high level, the Schur complement of an $n \times n$ matrix provides a matrix which is equivalent under linear system solves on a subset of coordinates in $[n]$.

Definition 2.1 (Schur complement)**.** For an $n \times n$ symmetric matrix $\boldsymbol{M}$ and subset of *terminals* $\mathcal{T} \subseteq [n]$, let $S = [n]\backslash\mathcal{T}$. Permute the rows/columns of $\boldsymbol{M}$ to write

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{M}_{[S,S]} & \boldsymbol{M}_{[S,\mathcal{T}]} \\ \boldsymbol{M}_{[\mathcal{T},S]} & \boldsymbol{M}_{[\mathcal{T},\mathcal{T}]}. \end{bmatrix}$$

Then the *Schur complement* of $\boldsymbol{M}$ onto $\mathcal{T}$ is denoted $\mathbf{SC}(\boldsymbol{M}, \mathcal{T}) \stackrel{\text{def}}{=} \boldsymbol{M}_{[\mathcal{T},\mathcal{T}]} - \boldsymbol{M}_{[\mathcal{T},S]}\boldsymbol{M}^{-1}_{[S,S]}\boldsymbol{M}_{[S,\mathcal{T}]}$.

For a graph G and subset $\mathcal{T} \subseteq V(G)$, for simplicity we write $\mathbf{SC}(G, \mathcal{T}) \stackrel{\text{def}}{=} \mathbf{SC}(\boldsymbol{L}_G, \mathcal{T})$. It is well-known that $\mathbf{SC}(G, \mathcal{T})$ is also a Laplacian.

Lemma 2.2 (Cholesky factorization)**.** *Given a matrix $\boldsymbol{M} \in \mathbb{R}^{n\times n}$, a subset $\mathcal{T} \subseteq [n]$, and $S = [n]\backslash\mathcal{T}$, we have*

$$\boldsymbol{M}^{-1} = \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{M}^{-1}_{[S,S]}\boldsymbol{M}_{[S,\mathcal{T}]} \\ 0 & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{M}^{-1}_{[S,S]} & 0 \\ 0 & \mathbf{SC}(\boldsymbol{M}, \mathcal{T})^{-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & 0 \\ -\boldsymbol{M}_{[\mathcal{T},S]}\boldsymbol{M}^{-1}_{[S,S]} & \boldsymbol{I} \end{bmatrix}.$$

The Cholesky factorization directly implies that the Schur complement represents the inverse of the Laplacian on a subset of the coordinates.

Lemma 2.3 (e.g. Fact 5.4 in [DKP+17])**.** *Let $\boldsymbol{I}$ be the identity matrix, and let $\boldsymbol{J}$ be the all 1 matrix. For any graph G, and subset $\mathcal{T} \subseteq V(G)$ we have that*

$$\mathbf{SC}(G, \mathcal{T})^\dagger = (\boldsymbol{I} - |\mathcal{T}|^{-1}\boldsymbol{J})(\boldsymbol{L}^\dagger_G)_{[\mathcal{T},\mathcal{T}]}(\boldsymbol{I} - |\mathcal{T}|^{-1}\boldsymbol{J}).$$

In addition, we have that

$$\mathbf{SC}(G, \mathcal{T})(\boldsymbol{L}^\dagger_G)_{[\mathcal{T},\mathcal{T}]}\mathbf{SC}(G, \mathcal{T}) = \mathbf{SC}(G, \mathcal{T}).$$

An equivalent view is that the quadratic form of the Schur complement gives the minimum energy extension of a vector on the terminals to the original vertex set, in the quadratic form of the original Laplacian [Gre96, MP13].

Lemma 2.4. *(Lemma B.2. of [MP13], matrix version in Appendix A.5.5 of [BBV04]) For a graph G and a $\mathcal{T} \subseteq V(G)$, the Schur complement of the Laplacian of G onto $\mathcal{T}$, $\mathbf{SC}(G, \mathcal{T})$ satisfies for all vectors $\vec{x}_{[\mathcal{T}]}$:*

$$\left\|\vec{x}_{[\mathcal{T}]}\right\|_{\mathbf{SC}(G,\mathcal{T})} = \min_{\vec{x}_{[V\setminus\mathcal{T}]} \in \mathbb{R}^{V\setminus\mathcal{T}}} \left\| \begin{bmatrix} \vec{x}_{[V\setminus\mathcal{T}]} \\ \vec{x}_{[\mathcal{T}]} \end{bmatrix} \right\|_{L(G)}.$$

Matrix Analysis Tools Our algorithm for computing Schur complement sparsifiers which are minors requires computing and sampling via leverage scores.

Definition 2.5 (Effective resistance and leverage scores)**.** For a graph G with resistances r_e, define $\mathrm{res}_G(e) \stackrel{\text{def}}{=} b_e^\top \boldsymbol{L}_G^\dagger b_e$ and $\mathrm{lev}_G(e) \stackrel{\text{def}}{=} \mathrm{res}_G(e)/r_e$.

Note that $0 \le \mathrm{lev}_G(e) \le 1$ and $\sum_{e\in E(G)} \mathrm{lev}_G(e) = n - 1$ for connected graphs G.

Let $G\backslash e$ and G/e denote the graphs resulting respectively from deleting and contracting edge e. Note that these correspond to setting the resistance of edge e to positive infinity or 0, respectively. The Woodbury matrix formula allows us to understand changes in the quadratic form when resistances of of the edges change.

Lemma 2.6 (Woodbury matrix formula)**.** *For matrices $\boldsymbol{A}, \boldsymbol{U}, \boldsymbol{C}, \boldsymbol{V}$ of compatible sizes we have*

$$(\boldsymbol{A} + \boldsymbol{UCV})^\dagger = \boldsymbol{A}^\dagger - \boldsymbol{A}^\dagger \boldsymbol{U}(\boldsymbol{C}^{-1} + \boldsymbol{V}\boldsymbol{A}^\dagger \boldsymbol{U})^{-1}\boldsymbol{V}\boldsymbol{A}^\dagger.$$

We use the following to understand the matrix martingales that arise in the analysis minor-based Schur complements.

Lemma 2.7 (Freedman's inequality for matrix martingales [Tro11])**.** *Consider a matrix martingale $(\boldsymbol{Y}^{(k)})_{k\ge 0}$ whose values are symmetric matrices with dimension d and let $(\boldsymbol{X}^{(k)})_{k\ge 1}$ be the difference sequence $\boldsymbol{X}^{(k)} \stackrel{\text{def}}{=} \boldsymbol{Y}^{(k)} - \boldsymbol{Y}^{(k-1)}$. Assume that the difference sequence is uniformly bounded in that $\|\boldsymbol{X}^{(k)}\|_2 \le R$ almost surely for $k \ge 1$. Define the predictable quadratic variation random matrix*

$$\boldsymbol{W}^{(k)} \stackrel{\text{def}}{=} \sum_{j=1}^{k} \mathbb{E}[(\boldsymbol{X}^{(j)})^2 | \boldsymbol{X}^{(j-1)}].$$

Then for all $\epsilon \ge 0$ and $\sigma^2 > 0$ we have that

$$\Pr\left[\exists k > 0 : \|\boldsymbol{Y}^{(k)} - \boldsymbol{Y}^{(0)}\|_2 \ge \epsilon \text{ and } \|\boldsymbol{W}^{(k)}\|_2 \le \sigma^2\right] \le 2d \cdot \exp\left(\frac{-\epsilon^2/3}{\sigma^2 + R\epsilon/3}\right).$$

The induced 2-norm of a symmetric matrix is bounded by its maximum row sum.

Lemma 2.8. *For a symmetric matrix $\boldsymbol{M} \in \mathbb{R}^{n\times n}$, we have that*

$$\|\boldsymbol{M}\|_2 \le \max_{i\in[n]} \sum_{j\in[n]} |\boldsymbol{M}_{ij}|.$$

Proof. For all vectors $\vec{x}$, note by the AM-GM inequality that

$$\vec{x}^\top \boldsymbol{M}\vec{x} = \sum_{1\le i,j\le n} \vec{x}_i \vec{x}_j \boldsymbol{M}_{ij} \le \sum_{1\le i,j\le n} \vec{x}_i^2 \left|\boldsymbol{M}_{ij}\right| \le \max_{i\in[n]} \sum_{j\in[n]} \left|\boldsymbol{M}_{ij}\right| \sum_{i\in[n]} \vec{x}_i^2 \le \max_{i\in[n]} \sum_{j\in[n]} \left|\boldsymbol{M}_{ij}\right| \|x\|_2^2.$$

□

CONGEST **model** In the CONGEST model [Pel00], we are given a communication network $\overline{G} = (\overline{V}, \overline{E})$ with $\overline{n}$ nodes modelling processors that have unique $O(\log n)$-bit IDs, $\overline{m}$ edges modelling bidirectional communication links between the processors, and diameter D. Initially, each node knows its own ID and the IDs of its neighbors as well as the value of n. Computation in this model is carried out in rounds synchronized by a global clock. In each round, every node sends to each of its neighbors an arbitrary $O(\log n)$-bit message, receives the messages of its neighbors, performs arbitrary internal computation, and stores arbitrary information for the next round. The main goal in this paper is to design algorithms for graph problems with a small number of rounds. For problems on directed graphs, the direction of each edge is known by both of its endpoints, but the corresponding communication link is still bidirectional. For problems on weighted graphs (involving, e.g., costs or capacities), the weight of each edge is known by both of its endpoints, but the corresponding communication link still allows for direct transmission of each message within a single round. In particular the diameter D always refers to the underlying undirected, unweighted communication network. Our running time bounds hold under the assumption that all weights are polynomial in n, which is a standard assumption in the CONGEST model literature.

3 Overview

Here we will give the main ideas behind our algorithm that efficiently solves Laplacians in a distributed setting (Theorem 1). We start by discussing elimination-based parallel Laplacian algorithms which remove a constant fraction of vertices to reduce the size of the graph. This naturally leads to requiring sparsifiers of the Schur complement that are minors of the original graph, whose existence is shown by [LS18]. Our key contribution is a nontrivial distributed implementation of their construction by giving a parallel variant of their algorithm that avoids the sampling of random spanning trees and samples by approximate leverage scores. We analyze this algorithm using matrix martingales. Finally, to achieve our main result we combine the algorithm with tree-based ultrasparsifiers, an alternate vertex reduction scheme that is not parallel but significantly reduces the size of the graph.

Parallel Laplacian Solvers via Elimination The starting point for our algorithm is based on the $\text{poly}(\log n)$ round Laplacian system solvers in the PRAM model, namely the sparsified Cholesky algorithm from [KLP+16]. This algorithm repeatedly finds a constant fraction of the vertices on which the block minor is "almost independent" and hence easy to solve. The inverse of this block then gives the result of eliminating these vertices, which is the Schur complement on the rest of the vertices, which we view as the terminal vertices $\mathcal{T}$.

More explicitly, this can be seen in the context of the Cholesky factorization in Lemma 2.2, where we let $\boldsymbol{M} = \boldsymbol{L}(G)$ be the Laplacian. We find an "almost independent" set of vertices S so that computing $\boldsymbol{M}^{-1}_{[S,S]}$ to high accuracy is simple using a preconditioned gradient descent method. Therefore, the remaining difficulty in computing $\boldsymbol{M}^{-1}$ is simply from computing and inverting the Schur complement: $\mathbf{SC}(\boldsymbol{M}, \mathcal{T})^{-1}$. To do this, we first approximately compute the Schur complement $\mathbf{SC}(\boldsymbol{M}, \mathcal{T})$, which is again a Laplacian, and then recursively apply a Cholesky factorization to it again.

However, this resulting Schur complement may be dense, even if the original graph is sparse. For example, eliminating the center of a star results in a complete graph on the peripheral leaf vertices. To make this more efficient, sparsified elimination algorithms [KLP+16, CGP+18, DPPR20] seek to directly construct a sparse approximation of this Schur complement. This can be done in a variety of ways, but algorithmically one of the simplest interpretations is through the sampling of random walks. Indeed,

matrix concentration bounds imply that the following procedure suffices for generating a good approximation of $\mathbf{SC}(G, \mathcal{T})$ with high probability:

Algorithm 1: Approximate Schur Complement using Random Walks

```
Set H ← ∅
for each edge e = uv in G do
    Repeat the following two steps O(ε^{-2} log n) times:
    Random walk both endpoints u and v until they are in 𝒯, to t_u and t_v respectively.
    Add an edge to the approximate Schur complement H between t_u and t_v, with weight as
      function of the original weight, and the number of steps the walk took.
return H
```

By picking $\mathcal{T}$ so that $V \backslash \mathcal{T}$ is almost independent, that is, each vertex not in $\mathcal{T}$ has a constant fraction of its weight going to $\mathcal{T}$, it can be ensured that the lengths of the walks don't exceed $O(\log n)$ with high probability. As a result, PRAM algorithms are able to construct low error Schur complements by sampling about $O(\epsilon^{-2} \log n)$ walks of length $O(\log n)$ per edge. As the number of vertices in the Schur complement decreases by a constant factor per step, this process yields a parallel solver with another $O(\log n)$ factor overhead in parallel depth.

Another contribution we make is introducing a new technique that reduces the congestion of these random walks by augmenting the terminal set $\mathcal{T}$. More concretely, recall that in the CONGEST model, each edge can only pass $O(\log n)$ bits per round. Random walks in weighted graphs on the other hand may severely congest some edges: consider for example, a star with one very heavily weighted edge, and rest lightly weighted. All the walks starting from the lightly weighted edges' end points will likely utilize the heavily weighted edges, leading to a congestion of $\Omega(n)$ in the worst case. To resolve this we use a procedure to *estimate* the congestion of an edge accumulated by such random walks. We use these estimates to add edges with high estimated congestion to $\mathcal{T}$ to ensure that remaining edges have low congestion.

However, a single elimination round only removes a constant fraction of the vertices, but performing $\Omega(\log n)$ elimination rounds would result in a significant blowup in the congestion (as each elimination round accumulates $\widetilde{O}(1)$ congestion). Hence, we only perform $\Theta((\log \log n)^2)$ rounds of elimination between sparsification steps. A formal statement of this elimination scheme is shown in Lemma 4.10.

Minor Sparsifiers and its Distributed Construction After that, the core component of our algorithm is that we must bring the Schur complement back to being a minor of the original graph, by constructing a spectral sparsifier of the Schur complement which is a minor of the original graph (Theorem 3). That is, the Schur complement results from contracting connected subsets of vertices in the original graph and reweighting edges. Minors are particularly useful for distributed algorithms because we can simulate one round of communication between neighbors on a minor, such as multiplying by the incidence matrix, in $\widetilde{O}(\sqrt{n} + D)$ rounds (Lemma 4.3). They interact particularly well with the parallel Laplacian solving algorithm which is a short sequence of matrix-vector multiplies on submatrices. Existence and efficient sequential constructions of these objects were first shown by Li and Schild [LS18].

A key contribution of our work is to give a simplified parallel variant of the algorithm of Li-Schild [LS18] which leads to an efficient distributed implementation. The algorithm of [LS18] works by contracting or deleting edges e with probability given by its leverage score. The main difference is that instead

of sampling edges using a random spanning tree, we identify a large subset of edges that can be sampled independent of each other, without affecting each edges's sampling probability too much. This is done via localization [SRS18], which provides an overall bound on the total influence of edges' effective resistances.

The algorithm then comprises of three main steps, and is analyzed via matrix martingales.

1. Calculating edges' influence on the Schur complement (Lemma 5.6).
2. Computing the mutual influence of edges's resistances, and picking a large set that has small mutual influence, which we term the steady set (Definition 5.1).
3. Among these steady edges, randomly contract/delete them with probability given by an approximation of their leverage scores.

Note that all steps actually require solving Laplacian systems in the original graph, which seems circular. We address this using the now well understood recursive approach of [Pen16], which we discuss below together with the overall algorithm.

Overall Recursive Scheme Given a graph G, the goal of the algorithm is to return a *chain* of approximate Schur complements of G, each with 0.99 as many vertices as the previous. This chain has length $O(\log n)$, and after built, can be applied in $O(\log n)$ steps and $\widetilde{O}(\sqrt{n} + D)$ rounds to solve a Laplacian system to high accuracy in the CONGEST model. The construction of the chain is as follows – pick $d = \Theta((\log \log n)^2)$ say, and run d rounds of the sparsified Cholesky elimination scheme (Lemma 4.10) to reduce the graph size to $0.99^d|V(G)|$. Now, use the minor Schur complement algorithm (Theorem 3) to build a minor of G which is a Schur complement sparsifier with respect to the remaining $0.99^d|V(G)|$ remaining vertices.

To compute the Schur complement sparsifier, we employ a separate recursion, because the Schur complement sparsifier construction requires Laplacian system solves to compute leverage scores (and other similar measures). To do this, we ultrasparsify the graph G, thus reducing the size by a factor of k (Lemma 4.9), and build a Schur complement chain on the ultrasparsifier. Now, we can use this solver on the ultrasparsifier to precondition a solver on G with $\widetilde{O}(\sqrt{k})$ steps of preconditioned conjugate gradient to compute the desired leverage scores. We want to emphasize the final Schur complement chain we output for G *does not* involve the ultrasparsifier, and hence can still be applied in parallel.

One final technical detail is that due to needing to solve submatrices of the Laplacian (Lemma 5.6) we require tracking graphs that embed with *low congestion* in the original graph, a slight generalization of minors (Definition 4.1). We ensure that the congestion stays as $n^{o(1)}$ throughout the algorithm, so it does not affect the final round complexity.

4 Full Algorithm and Analysis

The goal of this section is to formalize the notions and graph reduction algorithms described in Section 3, and provide a bound for the overall performance.

4.1 Distributed Communication on Minors of Overlay Networks

As described, we will work with graphs that are minors of the original graph, which doubles as the communication network. However, some of our linear systems reductions duplicate edges, leading to

minors with slightly larger congestion. So we will need to incorporate such congestion parameters into our definition of minors. The following definition is a direct extension of the distributed N-node cluster graph from [GKK+15], with congestion incorporated, and the connection with graph minors stated more explicitly.

Definition 4.1. Given a parameter $\rho \geq 1$, a graph G is a ρ-minor of H if we have the following mappings:

1. For each vertex of G, $u \in V(G)$:
   (a) A subset of vertices of H, which we term a supervertex, $S^{G\to H}(u) \subseteq V(H)$, with a root vertex $V_{map}^{G\to H}(u) \in S^{G\to H}(u)$.
   (b) A connected subgraph of H on $S^{G\to H}(u)$, which for simplicity we will keep as a tree, $T^{G\to H}(u)$. Note that this requires $S^{G\to H}(u)$ being connected in H.
2. A mapping of the edges of G onto edges of H, or self-loops on vertices of H, such that for any $u^G v^G = e^G \in E(G)$, the mapped edge $E_{map}^{G\to H}(e^G) = e^H = u^H v^H$ satisfies $u^H \in S^{G\to H}(u^G)$ and $v^H \in S^{G\to H}(v^G)$.

and additionally:

1. Each vertex of H is contained in at most ρ supervertices $V_{map}^{G\to H}(v^G)$ for some v^G.
2. Each edge of H appears as the image of the edge map $E_{map}^{G\to H}(\cdot)$, or in one of the trees connecting supervertices, $T^{G\to H}(v^G)$ for some v^G, at most ρ times.

When $\rho = 1$, then G is simply a minor of H.

Finally, we say a ρ-minor mapping is stored distributedly, or that G is ρ-minor distributed over H if it's stored by having all the images of the maps recording their sources. That is, each $v^H \in V(H)$ records

1. All v^G for which $v^H \in V_{map}^{G\to H}(v^G)$,
2. For each edge e^H incident to v^H (including self loops that may not exist in original H):
   (a) All vertices for which e^H is in the corresponding tree
   $$\left\{ v^G \mid e^H \in T^{G\to H}\left(v^G\right) \right\}$$
   (b) All edges e^G that map to it.

We will denote the original graph, which doubles as the overlay network, using $\overline{G}$.

Note that the vertex mappings, or even the neighborhoods of G, cannot be stored at one vertex in $\overline{G}$. This is because both of these sets may have size up to $\Omega(n)$, and passing that information to a single low degree vertex would incur too much communication.

We store vectors on G by putting the values at the root vertices of each of its corresponding supervertices. This notion of rooting can be made more explicit: we can compute directions for all edges in the spanning tree $T^{G\to\overline{G}}(v^G)$ that point to the corresponding root vertex $V_{map}^{G\to\overline{G}}(v^G)$.

Lemma 4.2. *Given a graph G that's ρ-minor distributed over a communication network $\overline{G} = (\overline{V}, \overline{E})$ with $\overline{n}$ vertices, $\overline{m}$ edges, and diameter D, we can compute in $O(\rho\sqrt{\overline{n}}\log\overline{n} + D)$ rounds of communication on $\overline{G}$, an orientation for each v^G and each edge $e \in T^{G\to\overline{G}}(v^G)$ such that each vertex other than the root has exactly one edge pointing away from it, and following these edges leads us to the root.*

We will make extensive usage of the following lemma, which we prove in Appendix B, about simulating communications on G using rounds of CONGEST communications in a graph that it ρ-minor distributes into.

Lemma 4.3. *Let $G = (V, E)$ be a graph with n vertices and m edges that ρ-minor distributes into a communication network $\overline{G} = (\overline{V}, \overline{E})$ with $\overline{n}$ vertices, $\overline{m}$ edges, and diameter D. In the* CONGEST *model, the following operations can be performed with high probability using $O(t\rho\sqrt{\overline{n}}\log\overline{n} + D)$ rounds of communication on $\overline{G}$:*

1. *Each $V_{map}^{G\to\overline{G}}(v^G)$ sends $O(t\log n)$ bits of information to all vertices in $S^{G\to\overline{G}}(v^G)$.*
2. *Simultaneously aggregate the sum/minimum of $O(t\log n)$ bits, from all vertices in $S^{G\to\overline{G}}(v^G)$ to $V_{map}^{G\to\overline{G}}(v^G)$ for all $v^G \in V(G)$.*

One direct use of this communication result is that it allows us to efficiently compute matrix-vector products.

Corollary 4.4. *Given a matrix $\boldsymbol{A}$ with nonzeroes supported on the edges of a graph G that's ρ-minor distributed over a communication network $\overline{G} = (\overline{V}, \overline{E})$ with $\overline{n}$ vertices, $\overline{m}$ edges, and diameter D, with values stored with endpoints of the corresponding edge, and a vector $\vec{x} \in \mathbb{R}^{|V(G)|}$ stored distributedly on the vertices $V_{map}^{G\to\overline{G}}(v^G)$, we can compute the vector $\boldsymbol{A}\vec{x}$, also stored at $V_{map}^{G\to\overline{G}}(v^G)$ using $O(t\rho\sqrt{\overline{n}}\log\overline{n} + D)$ of communication in the* CONGEST *model, with high probability.*

Proof. We first invoke Lemma 4.3 to pass $\vec{x}_{V^G}$ to all of $S^{G\to\overline{G}}(V^G)$. Then in $O(\rho)$ round of distributed communication, we can pass these entries (multiplied by the weights of $\boldsymbol{A}$) to the corresponding row index. That is, if $E_{map}^{G\to\overline{G}}(u^G v^G) = u^H v^H$, we pass

$$\boldsymbol{A}_{u^G v^G}\vec{x}_{v^G}$$

from v^H to u^H. Running Lemma 4.3 again to sum together the passed values over each super vertex then brings the values to the root vertex. □

Another implication is that that the Koutis-Xu distributed sparsification algorithm can also be simulated on $\overline{G}$ in the CONGEST model, with a round overhead of $\widetilde{O}(\sqrt{\overline{n}} + D)$ [KX16].

Corollary 4.5. *There is an algorithm,* SpectralSparsifyKX*, that for a graph G that ρ-minor distributes into $\overline{G}$, and some error $0 < \epsilon < 0.1$,* SpectralSparsifyKX$(G, \overline{G}, \epsilon)$ *with high probability returns in*

$$O\left(\left(\rho\sqrt{\overline{n}}\log\overline{n} + D\right)\log^8\overline{n}/\epsilon^2\right)$$

rounds, a graph $\widetilde{G}$, distributed as a ρ-minor in $\overline{G}$ such that:

1. *$\widetilde{G}$ is a (reweighted) subgraph of G,*

2. $\boldsymbol{L}(G) \approx_\epsilon \boldsymbol{L}(\widetilde{G})$,

3. $\widetilde{G}$ *has* $\widetilde{O}(|V(G)|/\epsilon^2)$ *edges.*

Proof. The algorithm by [KX16] is based on repeated spanner computations on subgraphs (which are obtained by removing edges from previous spanner computations and uniform sampling of edges). The spanner algorithm of [BS07], internally used in [KX16], iteratively grows clusters – organized as spanning trees rooted at center nodes – and adds edges to the spanner. In each iteration some of the existing clusters first are sampled at random, which is done by the respective center node who then forwards the information whether the cluster is sampled to all nodes in its cluster. Then each node decides whether it joins a cluster and if so which one and also decides which of its neighboring edges it adds to the spanner. These decisions are made by comparing the weights of its incident edges. Thus, all the operations performed by nodes in the algorithm of [BS07], and thus the algorithm of [KX16] fit the description of operations supported by Lemma 4.3. □

We will call this sparsification routine regularly, often as preprocessing. This is partly because subgraphs 1-minor distributes into itself trivially.

The minor property also compose naturally: a minor of a minor of G is also a minor of G. This holds with ρ-minors too, up to multiplications of the congestion parameters. We prove the following general composition result in Appendix B.

Lemma 4.6. *Given graphs G_1, and G_2 via a ρ_2-minor distribution of G_2 into $\overline{G}$, and a ρ_1-minor distribution of G_1 into G_2 stored on the root vertices of the supervertices of G_2, and images of G_2's edges in $\overline{G}$, we can, with high probability, compute using $\widetilde{O}(\rho_1\rho_2 \cdot (\sqrt{\overline{n}}+D))$ rounds of communication in the* CONGEST *model a $\rho_1 \cdot \rho_2$-minor distribution of G_1 into G.*

We will always work with congestions in the $n^{o(1)}$ range: this essentially means we can perform distributed algorithms on G, while paying an overhead of about $\sqrt{\overline{n}}+D$ in round complexity to simulate on the original graph.

The composition of minors from Lemma 4.6 implies, among others, that a subset of edges can be quickly contracted.

Corollary 4.7. *Given G that's ρ-minor distributed on $\overline{G}$, along with a subset of edges $F \subseteq E(G)$ then we can obtain a ρ-minor distribution of G/F (G with F contracted), into $\overline{G}$ in $\widetilde{O}(\rho(\sqrt{\overline{n}}+D))$ rounds, under the* CONGEST *model of computation.*

The proof of this requires running $O(\log n)$ rounds of parallel contraction on the edges of F. We defer it to Appendix B as well.

4.2 Laplacian Building Blocks

Some of our algorithm require working with submatrices of Laplacians, which may not be Laplacians anymore, but are still SDD. Fortunately, we can reduce solving an SDD matrix on a graph to solving a Laplacian on a 2-minor. The proof is straightforward and can be found in [Gre96] or [ST14].

Lemma 4.8 (Gremban [Gre96])**.** *Given an n-by-n SDD matrix $\boldsymbol{M}$ that ρ-minor distributes into $\overline{G}$, we can construct a graph H on $2n$ vertices, along with a 2ρ-minor distribution of H into $\overline{G}$ with vertices i and $n+i$'s*

roots mapping to the same root vertex, so that for any vector $\vec{b} \in \mathbb{R}^n$ *and any vectors* $\vec{x} \in \mathbb{R}^{2n}$ *such that*

$$\left\| \vec{x} - \boldsymbol{L}(H)^{\dagger} \begin{bmatrix} \vec{b} \\ \vec{b} \end{bmatrix} \right\|_{\boldsymbol{L}(H)} \leq \epsilon \left\| \begin{bmatrix} \vec{b} \\ \vec{b} \end{bmatrix} \right\|_{\boldsymbol{L}(H)^{\dagger}},$$

we have

$$\left\| \frac{\vec{x}_{1:n} - \vec{x}_{n+1:2n}}{2} - \boldsymbol{M}^{\dagger}\vec{b} \right\|_{\boldsymbol{M}} \leq \epsilon \left\| \vec{b} \right\|_{\boldsymbol{M}^{\dagger}}.$$

In this section we outline the main pieces needed to prove Theorem 1. As described in Section 3, we require three main graph reduction procedures: ultrasparsification (Lemma 4.9), sparsified Cholesky (Lemma 4.10), and minor-based Schur complements (Theorem 3).

The ultrasparsification procedure, based on [ST14, KMP10], allows us to significantly reduce the size of the graph and maintain congestion, but incurs a large approximation error.

Lemma 4.9. *There is a routine* UltraSparsify(G, k) *in the* CONGEST *model that given a graph* G *with* n *vertices and* m *edges, that* ρ*-minor distributes into the communication network* $\overline{G}$*, which has* $\overline{n}$ *vertices,* $\overline{m}$ *edges, and diameter* D*, along with a parameter* k*, produces in* $O(n^{o(1)}(\rho\sqrt{\overline{n}} + D))$ *rounds a graph* H *such that:*

1. H *is a subgraph of* G*,*
2. H *has at most* $n - 1 + m2^{O(\sqrt{\log n \log\log n})}/k$ *edges.*
3. $\boldsymbol{L}(G) \preceq \boldsymbol{L}(H) \preceq k\boldsymbol{L}(G)$*.*

Furthermore, the algorithm also gives $\widehat{G}, \boldsymbol{Z}_1, \boldsymbol{Z}_2, C$ *such that*

1. $\widehat{G}$ *1-minor distributes into* H *such that* $\widehat{G} = \mathbf{SC}(H, C)$ *with* $|C| = m2^{O(\sqrt{\log n \log\log n})}/k$*.*
2. *There are operators* $\boldsymbol{Z}_1$ *and* $\boldsymbol{Z}_2$ *evaluable with* $O(\rho\sqrt{\overline{n}}\log\overline{n}+D)$ *rounds of* CONGEST *communication on* $\overline{G}$ *such that:*

$$\boldsymbol{L}(H)^{\dagger} = \boldsymbol{Z}_1^{\top} \begin{bmatrix} \boldsymbol{Z}_2 & 0 \\ 0 & \boldsymbol{L}\left(\widehat{G}\right)^{\dagger} \end{bmatrix} \boldsymbol{Z}_1$$

The elimination procedure, based on [KLP+16], incurs small approximation error, but significantly increases the congestion.

Lemma 4.10. *There is a routine* Eliminate(G, d, ϵ) *in the* CONGEST *model that given a graph* G *that* ρ*-minor distributes into a communication network* $\overline{G}$*, along with step count* d *and error* ϵ*, produces in*

$$O((\epsilon^{-6}\log^{14} n)^d(\rho\sqrt{\overline{n}}\log\overline{n} + D))$$

rounds a subset $\mathcal{T}$ *and access to operators* $\boldsymbol{Z}_1$ *and* $\boldsymbol{Z}_2$ *such that*

1. $|\mathcal{T}| \leq (\frac{49}{50})^d|V(G)|$*.*
2. *The cost of applying* $\boldsymbol{Z}_1$*,* $\boldsymbol{Z}_1^{\top}$ *and* $\boldsymbol{Z}_2$ *to vectors is* $O((\epsilon^{-6}\log^{14}\overline{n})^d(\rho\sqrt{\overline{n}}\log\overline{n}+D))$ *rounds of communication on* $\overline{G}$*.*

3. $\boldsymbol{L}(G)^\dagger$ *is* $(1\pm\epsilon)^d$*-approximated by a composed operator built from* $\boldsymbol{Z}_1$, $\boldsymbol{Z}_2$, *and the inverse of the of the Schur complement of* $\boldsymbol{L}(G)$ *onto* C, $\mathbf{SC}(\boldsymbol{L}(G), C)$*:*

$$(1-\epsilon)^d \boldsymbol{L}(G)^\dagger \preceq \boldsymbol{Z}_1^\top \begin{bmatrix} \boldsymbol{Z}_2 & 0 \\ 0 & \mathbf{SC}(\boldsymbol{L}(G), C)^\dagger \end{bmatrix} \boldsymbol{Z}_1 \preceq (1+\epsilon)^d \boldsymbol{L}(G)^\dagger$$

Note that we cannot directly set $d = \Omega(\log n)$ to finish with $|\mathcal{T}|$ a constant: $(\log n)^{\log n}$ may be even larger than n. So we need to bring the structure back to a minor of the original communication network. For this we use the construction of spectral vertex sparsifiers that are minors [LS18], modified to not use random spanning trees.

Theorem 3. *There is a routine* ApproxSC$(G, \mathcal{T}, \epsilon)$ *in the* CONGEST *model that given a graph* G *with* n *vertices and* m *edges that* ρ*-minor distributes into the communication network* $\overline{G}$*, a subset of vertices* $\mathcal{T} \subseteq V(G)$*, an error parameter* $\epsilon < 0.1$*, and access to a (distributed) Laplacian solver* Solve*, it returns a graph* H*, represented as a distributed* ρ*-minor of* $\overline{G}$ *such that:*

1. $\mathcal{T} \subseteq V(H)$,
2. H *has* $O(|\mathcal{T}|\epsilon^{-2}\log^2 n)$ *edges (and hence at most that many vertices as well).*
3. *The Schur complements of* G *and* H *well approximate each other, i.e.,*

$$\mathbf{SC}(G, \mathcal{T}) \approx_\epsilon \mathbf{SC}(H, \mathcal{T}).$$

The cost of this computation consists of:

1. $O(\epsilon^{-3}\log^{10} n)$ *calls to* Solve *with accuracy* $1/\mathrm{poly}(n)$ *on graphs that* 2ρ*-distribute into* $\overline{G}$*.*
2. *An overhead of* $O(\rho(\overline{n}^{1/2} + D)\epsilon^{-3}\log^{11}\overline{n})$ *rounds.*

4.3 Schur complement chains and a proof of Theorem 1

In this section, we formally show how to combine Lemma 4.10, Lemma 4.9, and Theorem 3 to efficiently construct a Schur complement chain and prove Theorem 1.

Definition 4.11. For a graph G of n vertices, $\{(G_i, \boldsymbol{Z}_{i,1}, \boldsymbol{Z}_{i,2}, \mathcal{T}_i)\}_{i=1}^t$ is a (γ, ϵ)-Schur-complement solver chain of G if the following conditions hold.

1. $G_1 = G$.
2. $$(1-\epsilon)\boldsymbol{L}(G_i)^\dagger \preceq \boldsymbol{Z}_{i,1}^\top \begin{bmatrix} \boldsymbol{Z}_{i,2} & 0 \\ 0 & \mathbf{SC}(\boldsymbol{L}(G_i), \mathcal{T}_i)^\dagger \end{bmatrix} \boldsymbol{Z}_{i,1} \preceq (1+\epsilon)\boldsymbol{L}(G_i)^\dagger$$
3. $\mathcal{T}_i \subset V(G_{i+1}) \subset V(G_i)$ and $\mathbf{SC}(G_i, \mathcal{T}_i) \approx_\epsilon \mathbf{SC}(G_{i+1}, \mathcal{T}_i)$
4. $|V(G_i)| \geq \gamma \cdot |V(G_{i+1})|$ if $i < t$, and $|V(G_t)| \leq \gamma$.

Algorithm 2: PseudoinverseMulti $(\{(G_i, Z_{i,1}, Z_{i,2}, \mathcal{T}_i) \mid j \le i \le t\}, \vec{b})$

$\vec{u} \leftarrow Z_{j,1}\vec{b}$;
if $j = t$ **then**
  $\vec{v} \leftarrow \mathbf{SC}(G_t, \mathcal{T}_t)^\dagger \vec{u}_{[\mathcal{T}_t]}$;
  return $Z_{t,1}^T \begin{bmatrix} Z_{t,2}\vec{u}_{[V(G_t)\setminus\mathcal{T}_t]} \\ \vec{v} \end{bmatrix}$;
else
  $\vec{u}_1 \leftarrow \vec{u}_{[\mathcal{T}_j]} - \frac{\vec{u}_{[\mathcal{T}_j]}^T \vec{1}}{\|\vec{1}\|_2^2} \cdot \vec{1}$;
  $\vec{v} \leftarrow$ PseudoinverseMulti $\left(\{(G_i, Z_{i,1}, Z_{i,2}, \mathcal{T}_i) \mid j+1 \le i \le t\}, \begin{bmatrix} \vec{0} \\ \vec{u}_1 \end{bmatrix}\right)$;
  $\vec{v}_1 \leftarrow \vec{v}_{[\mathcal{T}_j]} - \frac{\vec{v}_{[\mathcal{T}_j]}^T \vec{1}}{\|\vec{1}\|_2^2} \cdot \vec{1}$;
  return $Z_{j,1}^T \begin{bmatrix} Z_{j,2}\vec{u}_{[V(G_j)\setminus\mathcal{T}_j]} \\ \vec{v}_1 \end{bmatrix}$;

Lemma 4.12. *Let $\overline{G}$ be a communication network with $\overline{n}$ vertices and $\overline{m}$ edges. Let $\{(G_i, Z_{i,1}, Z_{i,2}, \mathcal{T}_i)\}_{i=1}^t$ be a (γ, ϵ)-Schur-complement solver chain of graph G for some $\gamma \ge 2$ and $\epsilon \le \frac{1}{C \log n}$ for large constant C, satisfying the following conditions:*

1. *G_i ρ-minor distributes into $\overline{G}$.*
2. *Linear operators $Z_{i,1}$ and $Z_{i,2}$ can be evaluated in $O(\overline{n}^{o(1)}(\overline{n}^{1/2} + D))$ rounds.*

Then for a given vector $\vec{b}$, Algorithm PseudoinverseMulti *computes a vector $\vec{x}$ in $O(\rho\overline{n}^{o(1)}(\overline{n}^{1/2} + D))$ rounds such that*

$$\left\|\vec{x} - \boldsymbol{L}(G)^\dagger \vec{b}\right\|_{\boldsymbol{L}(G)} \le 2\epsilon \log n \cdot \left\|\vec{b}\right\|_{\boldsymbol{L}(G)^\dagger}.$$

The correctness proof rely heavily on the following conversion from operator guarantees to error guarantees.

Lemma 4.13. *(Lemma 1.6.7 of [Pen13]) If $\boldsymbol{A}$ and $\boldsymbol{B}$ are two symmetric PSD matrices such that $\boldsymbol{A} \approx_\delta \boldsymbol{B}^\dagger$ for some $0 < \delta < 1$, then for any vector $\vec{b}$, we have*

$$\left\|\boldsymbol{A}\vec{b} - \boldsymbol{B}^\dagger\vec{b}\right\|_{\boldsymbol{B}} \le \delta \left\|\vec{b}\right\|_{\boldsymbol{B}^\dagger}.$$

Proof of Lemma 4.12. We prove this lemma by induction on j. For the base case $j = t$, we have that

$$\begin{aligned} Z_{t,1}^T \begin{bmatrix} Z_{t,2}\vec{u}_{[V(G_t)\setminus\mathcal{T}_t]} \\ \vec{v} \end{bmatrix} &= Z_{t,1}^T \begin{bmatrix} Z_{t,2}\vec{u}_{[V(G_t)\setminus\mathcal{T}_t]} \\ \mathbf{SC}(G_t, \mathcal{T}_t)^\dagger \vec{u}_{[\mathcal{T}_t]} \end{bmatrix} \\ = Z_{t,1}^T \begin{bmatrix} Z_{t,2} & 0 \\ 0 & \mathbf{SC}(G_t, \mathcal{T}_t)^\dagger \end{bmatrix} \vec{u} &= Z_{t,1}^T \begin{bmatrix} Z_{t,2} & 0 \\ 0 & \mathbf{SC}(G_t, \mathcal{T}_t)^\dagger \end{bmatrix} Z_{t,1}\vec{b}, \end{aligned} \tag{1}$$

in which

$$(1-\epsilon)\boldsymbol{L}(G_t)^\dagger \preceq Z_{t,1}^T \begin{bmatrix} Z_{t,2} & 0 \\ 0 & \mathbf{SC}(G_t, \mathcal{T}_t)^\dagger \end{bmatrix} Z_{t,1} \preceq (1+\epsilon)\boldsymbol{L}(G_t)^\dagger. \tag{2}$$

By Lemma 4.13 and combining (1) and (2), we have

$$\left\| Z_{t,1}^T \begin{bmatrix} Z_{t,2}\vec{u}_{[V(G_t)\setminus\mathcal{T}_t]} \\ \vec{v} \end{bmatrix} - \boldsymbol{L}(G_t)^\dagger \vec{b} \right\|_{\boldsymbol{L}(G_t)} \leq \epsilon \left\| \vec{b} \right\|_{\boldsymbol{L}(G_t)^\dagger}.$$

We suppose that it holds for case $j+1$, i.e.,

$$\begin{aligned} &\left\| \textsc{PseudoinverseMulti}(\{(G_i, Z_{i,1}, Z_{i,2}, \mathcal{T}_i) \mid j+1 \leq i \leq t\}, \vec{b}) - \boldsymbol{L}(G_{j+1})^\dagger \vec{b} \right\|_{\boldsymbol{L}(G_{j+1})} \\ \leq & 2(t-j)\epsilon \left\| \vec{b} \right\|_{\boldsymbol{L}(G_{j+1})^\dagger}, \end{aligned}$$

then we have that

$$\left\| \vec{v} - \boldsymbol{L}(G_{j+1})^\dagger \begin{bmatrix} \vec{0} \\ \vec{u}_1 \end{bmatrix} \right\|_{\boldsymbol{L}(G_{j+1})} \leq 2(t-j)\epsilon \left\| \begin{bmatrix} \vec{0} \\ \vec{u}_1 \end{bmatrix} \right\|_{\boldsymbol{L}(G_{j+1})^\dagger}. \tag{3}$$

Lemma 2.4 gives

$$\left\| \vec{v}_{[\mathcal{T}_j]} - \left(\boldsymbol{L}(G_{j+1}^\dagger) \begin{bmatrix} \vec{0} \\ \vec{u}_1 \end{bmatrix} \right)_{[\mathcal{T}_j]} \right\|_{\mathbf{SC}(G_{j+1}, \mathcal{T}_j)} \leq \left\| \vec{v} - \boldsymbol{L}(G_{j+1})^\dagger \begin{bmatrix} \vec{0} \\ \vec{u}_1 \end{bmatrix} \right\|_{\boldsymbol{L}(G_{j+1})}. \tag{4}$$

Combining (3) and (4), we have

$$\left\| \vec{v}_{[\mathcal{T}_j]} - \left(\boldsymbol{L}(G_{j+1}^\dagger) \begin{bmatrix} \vec{0} \\ \vec{u}_1 \end{bmatrix} \right)_{[\mathcal{T}_j]} \right\|_{\mathbf{SC}(G_{j+1}, \mathcal{T}_j)} \leq 2(t-j)\epsilon \left\| \begin{bmatrix} \vec{0} \\ \vec{u}_1 \end{bmatrix} \right\|_{\boldsymbol{L}(G_{j+1})^\dagger}. \tag{5}$$

Now we prove the case j. By triangle inequality, we have that

$$\left\| Z_{j,1}^T \begin{bmatrix} Z_{j,2}\vec{u}_{[V(G_j)\setminus\mathcal{T}_j]} \\ \vec{v}_1 \end{bmatrix} - \boldsymbol{L}(G_j)^\dagger \vec{b} \right\|_{\boldsymbol{L}(G_j)} \tag{6}$$

$$\leq \left\| Z_{j,1}^T \begin{bmatrix} Z_{j,2}\vec{u}_{[V(G_j)\setminus\mathcal{T}_j]} \\ \vec{v}_1 \end{bmatrix} - Z_{j,1}^T \begin{bmatrix} Z_{j,2} & 0 \\ 0 & \mathbf{SC}(G_j, \mathcal{T}_j)^\dagger \end{bmatrix} Z_{j,1}\vec{b} \right\|_{\boldsymbol{L}(G_j)} \tag{7}$$

$$+ \left\| Z_{j,1}^T \begin{bmatrix} Z_{j,2} & 0 \\ 0 & \mathbf{SC}(G_j, \mathcal{T}_j)^\dagger \end{bmatrix} Z_{j,1}\vec{b} - \boldsymbol{L}(G_j)^\dagger \vec{b} \right\|_{\boldsymbol{L}(G_j)}. \tag{8}$$

Obviously, for (8), Lemma 4.12 gives

$$\left\| Z_{j,1}^T \begin{bmatrix} Z_{j,2} & 0 \\ 0 & \mathbf{SC}(G_j, \mathcal{T}_j)^\dagger \end{bmatrix} Z_{j,1}\vec{b} - \boldsymbol{L}(G_j)^\dagger \vec{b} \right\|_{\boldsymbol{L}(G_j)} \leq \epsilon \left\| \vec{b} \right\|_{\boldsymbol{L}(G_j)^\dagger}. \tag{9}$$

Now our task is to bound (7),

$$\begin{aligned}
&\left\| Z_{j,1}^T \begin{bmatrix} Z_{j,2}\vec{u}_{[V(G_j)\setminus\mathcal{T}_j]} \\ \vec{v}_1 \end{bmatrix} - Z_{j,1}^T \begin{bmatrix} Z_{j,2} & 0 \\ 0 & \mathbf{SC}(G_j,\mathcal{T}_j)^\dagger \end{bmatrix} Z_{j,1}\vec{b} \right\|_{L(G_j)} \\
=&\left\| Z_{j,1}^T \begin{bmatrix} Z_{j,2}\vec{u}_{[V(G_j)\setminus\mathcal{T}_j]} \\ \vec{v}_1 \end{bmatrix} - Z_{j,1}^T \begin{bmatrix} Z_{j,2} & 0 \\ 0 & \mathbf{SC}(G_j,\mathcal{T}_j)^\dagger \end{bmatrix} \vec{u} \right\|_{L(G_j)} \\
=&\left\| Z_{j,1}^T \begin{bmatrix} Z_{j,2}\vec{u}_{[V(G_j)\setminus\mathcal{T}_j]} \\ \vec{v}_1 \end{bmatrix} - Z_{j,1}^T \begin{bmatrix} Z_{j,2}\vec{u}_{[V(G_j)\setminus\mathcal{T}_j]} \\ \mathbf{SC}(G_j,\mathcal{T}_j)^\dagger\vec{u}_{[\mathcal{T}_j]} \end{bmatrix} \right\|_{L(G_j)} \\
=&\left\| Z_{j,1}^T \begin{bmatrix} \vec{0}_{[V(G_j)\setminus\mathcal{T}_j]} \\ \vec{v}_1 - \mathbf{SC}(G_j,\mathcal{T}_j)^\dagger\vec{u}_{[\mathcal{T}_j]} \end{bmatrix} \right\|_{L(G_j)} = \left\| \begin{bmatrix} \vec{0}_{[V(G_j)\setminus\mathcal{T}_j]} \\ \vec{v}_1 - \mathbf{SC}(G_j,\mathcal{T}_j)^\dagger\vec{u}_{[\mathcal{T}_j]} \end{bmatrix} \right\|_{Z_{j,1}L(G_j)Z_{j,1}^T}.
\end{aligned} \tag{10}$$

By triangle inequality, (10) gives

$$\begin{aligned}
&\left\| \begin{bmatrix} \vec{0}_{[V(G_j)\setminus\mathcal{T}_j]} \\ \vec{v}_1 - \mathbf{SC}(G_j,\mathcal{T}_j)^\dagger\vec{u}_{[\mathcal{T}_j]} \end{bmatrix} \right\|_{Z_{j,1}L(G_j)Z_{j,1}^T} \\
\le&\left\| \begin{bmatrix} \vec{0}_{[V(G_j)\setminus\mathcal{T}_j]} \\ \vec{v}_1 - \mathbf{SC}(G_{j+1},\mathcal{T}_j)^\dagger\vec{u}_{[\mathcal{T}_j]} \end{bmatrix} \right\|_{Z_{j,1}L(G_j)Z_{j,1}^T} \\
+&\left\| \begin{bmatrix} \vec{0}_{[V(G_j)\setminus\mathcal{T}_j]} \\ \mathbf{SC}(G_{j+1},\mathcal{T}_j)^\dagger\vec{u}_{[\mathcal{T}_j]} - \mathbf{SC}(G_j,\mathcal{T}_j)^\dagger\vec{u}_{[\mathcal{T}_j]} \end{bmatrix} \right\|_{Z_{j,1}L(G_j)Z_{j,1}^T},
\end{aligned} \tag{11}$$

in which

$$\begin{aligned}
&\left\| \begin{bmatrix} \vec{0}_{[V(G_j)\setminus\mathcal{T}_j]} \\ \vec{v}_1 - \mathbf{SC}(G_{j+1},\mathcal{T}_j)^\dagger\vec{u}_{[\mathcal{T}_j]} \end{bmatrix} \right\|_{Z_{j,1}L(G_j)Z_{j,1}^T} \\
=&\left\| \begin{bmatrix} \vec{0}_{[V(G_j)\setminus\mathcal{T}_j]} \\ \vec{v}_1 - \boldsymbol{P}(\boldsymbol{L}(G_{j+1})^\dagger))_{[\mathcal{T}_j,\mathcal{T}_j]}\boldsymbol{P}\vec{u}_{[\mathcal{T}_j]} \end{bmatrix} \right\|_{Z_{j,1}L(G_j)Z_{j,1}^T} \\
=&\left\| \begin{bmatrix} \vec{0}_{[V(G_j)\setminus\mathcal{T}_j]} \\ \vec{v}_1 - \boldsymbol{P}(\boldsymbol{L}(G_{j+1})^\dagger))_{[\mathcal{T}_j,\mathcal{T}_j]}\vec{u}_1 \end{bmatrix} \right\|_{Z_{j,1}L(G_j)Z_{j,1}^T} \\
=&\left\| \begin{bmatrix} \vec{0}_{[V(G_j)\setminus\mathcal{T}_j]} \\ \boldsymbol{P}\vec{v}_{[\mathcal{T}_j]} - \boldsymbol{P}\left(\boldsymbol{L}(G_{j+1})^\dagger \begin{bmatrix} \vec{0} \\ \vec{u}_1 \end{bmatrix}\right)_{[\mathcal{T}_j]} \end{bmatrix} \right\|_{Z_{j,1}L(G_j)Z_{j,1}^T} \\
=&\left\| \vec{v}_{[\mathcal{T}_j]} - \left(\boldsymbol{L}(G_{j+1})^\dagger \begin{bmatrix} \vec{0} \\ \vec{u}_1 \end{bmatrix}\right)_{[\mathcal{T}_j]} \right\|_{\boldsymbol{P}(Z_{j,1}L(G_j)Z_{j,1}^T)_{[\mathcal{T}_j,\mathcal{T}_j]}\boldsymbol{P}},
\end{aligned} \tag{12}$$

where $\boldsymbol{P}$ is the projection matrix of the space spanned by $\mathbf{SC}(G_{j+1},\mathcal{T}_j)$. Furthermore, the vectors $\vec{u}_1$ and $\vec{v}_1$ are the projections of the vectors $\vec{u}_{[\mathcal{T}_j]}$ and $\vec{v}_{[\mathcal{T}_j]}$ onto $\boldsymbol{P}$ respectively.

By the given condition, we have that

$$\boldsymbol{L}(G_j)^\dagger \approx_\epsilon Z_{j,1}^T \begin{bmatrix} Z_{j,2} & 0 \\ 0 & \mathbf{SC}(G_j,\mathcal{T}_j)^\dagger \end{bmatrix} Z_{j,1}. \tag{13}$$

Multiplying the both sides of the LHS and RHS of (13) by $\boldsymbol{L}(G_j)$ gives

$$\boldsymbol{L}(G_j) \approx_\epsilon \boldsymbol{L}(G_j)\boldsymbol{Z}_{j,1}^T \begin{bmatrix} \boldsymbol{Z}_{j,2} & 0 \\ 0 & \mathbf{SC}(G_j, \mathcal{T}_j)^\dagger \end{bmatrix} \boldsymbol{Z}_{j,1}\boldsymbol{L}(G_j). \tag{14}$$

Multiplying the left (resp. right) side of the LHS and RHS of (14) by $\boldsymbol{Z}_{j,1}$ (resp. $\boldsymbol{Z}_{j,1}^T$) gives

$$\boldsymbol{Z}_{j,1}\boldsymbol{L}(G_j)\boldsymbol{Z}_{j,1}^T \approx_\epsilon \boldsymbol{Z}_{j,1}\boldsymbol{L}(G_j)\boldsymbol{Z}_{j,1}^T \begin{bmatrix} \boldsymbol{Z}_{j,2} & 0 \\ 0 & \mathbf{SC}(G_j, \mathcal{T}_j)^\dagger \end{bmatrix} \boldsymbol{Z}_{j,1}\boldsymbol{L}(G_j)\boldsymbol{Z}_{j,1}^T, \tag{15}$$

which implies that

$$\begin{bmatrix} \boldsymbol{Z}_{j,2} & 0 \\ 0 & \mathbf{SC}(G_j, \mathcal{T}_j)^\dagger \end{bmatrix} \approx_\epsilon (\boldsymbol{Z}_{j,1}\boldsymbol{L}(G_j)\boldsymbol{Z}_{j,1}^T)^\dagger \tag{16}$$

and

$$\boldsymbol{Z}_{j,1}\boldsymbol{L}(G_j)\boldsymbol{Z}_{j,1}^T \approx_\epsilon \begin{bmatrix} \boldsymbol{Z}_{j,2}^\dagger & 0 \\ 0 & \mathbf{SC}(G_j, \mathcal{T}_j) \end{bmatrix}. \tag{17}$$

Moreover, (17) gives

$$(\boldsymbol{Z}_{j,1}\boldsymbol{L}(G_j)\boldsymbol{Z}_{j,1}^T)_{[\mathcal{T}_j,\mathcal{T}_j]} \approx_\epsilon \mathbf{SC}(G_j, \mathcal{T}_j). \tag{18}$$

Multiplying the both sides of the LHS and RHS of (18) by $\boldsymbol{P}$ gives

$$\boldsymbol{P}(\boldsymbol{Z}_{j,1}\boldsymbol{L}(G_j)\boldsymbol{Z}_{j,1}^T)_{[\mathcal{T}_j,\mathcal{T}_j]}\boldsymbol{P} \approx_\epsilon \mathbf{SC}(G_j, \mathcal{T}_j). \tag{19}$$

In addition, $\mathbf{SC}(G_{j+1}, \mathcal{T}_j) \approx_\epsilon \mathbf{SC}(G_j, \mathcal{T}_j)$. Combining with (19), we have

$$\boldsymbol{P}(\boldsymbol{Z}_{j,1}\boldsymbol{L}(G_j)\boldsymbol{Z}_{j,1}^T)_{[\mathcal{T}_j,\mathcal{T}_j]}\boldsymbol{P} \approx_{2\epsilon} \mathbf{SC}(G_{j+1}, \mathcal{T}_j). \tag{20}$$

Getting back to (12), by (20) we have that

$$\begin{aligned} &\left\| \vec{v}_{[\mathcal{T}_j]} - \left(\boldsymbol{L}(G_{j+1})^\dagger \begin{bmatrix} \vec{0} \\ \vec{u}_1 \end{bmatrix} \right)_{[\mathcal{T}_j]} \right\|_{\boldsymbol{P}(\boldsymbol{Z}_{j,1}\boldsymbol{L}(G_j)\boldsymbol{Z}_{j,1}^T)_{[\mathcal{T}_j,\mathcal{T}_j]}\boldsymbol{P}} \\ \leq &e^\epsilon \left\| \vec{v}_{[\mathcal{T}_j]} - \left(\boldsymbol{L}(G_{j+1})^\dagger \begin{bmatrix} \vec{0} \\ \vec{u}_1 \end{bmatrix} \right)_{[\mathcal{T}_j]} \right\|_{\mathbf{SC}(G_{j+1},\mathcal{T}_j)}. \end{aligned} \tag{21}$$

Combining (21) with (5) gets

$$\left\| \vec{v}_{[\mathcal{T}_j]} - \left(\boldsymbol{L}(G_{j+1})^\dagger \begin{bmatrix} \vec{0} \\ \vec{u}_1 \end{bmatrix} \right)_{[\mathcal{T}_j]} \right\|_{\boldsymbol{P}(\boldsymbol{Z}_{j,1}\boldsymbol{L}(G_j)\boldsymbol{Z}_{j,1}^T)_{[\mathcal{T}_j,\mathcal{T}_j]}\boldsymbol{P}} \leq 2(t-j)\epsilon e^\epsilon \left\| \begin{bmatrix} \vec{0} \\ \vec{u}_1 \end{bmatrix} \right\|_{\boldsymbol{L}(G_{j+1})^\dagger}. \tag{22}$$

Now consider bounding $\left\|\begin{bmatrix}\vec{0}\\ \vec{u}_1\end{bmatrix}\right\|_{\boldsymbol{L}(G_{j+1})^\dagger}$,

$$\begin{aligned}\left\|\begin{bmatrix}\vec{0}\\ \vec{u}_1\end{bmatrix}\right\|_{\boldsymbol{L}(G_{j+1})^\dagger} &= \|\vec{u}_1\|_{(\boldsymbol{L}(G_{j+1})^\dagger)_{[\mathcal{T}_j,\mathcal{T}_j]}} = \left\|\boldsymbol{P}\vec{u}_{[\mathcal{T}_j]}\right\|_{(\boldsymbol{L}(G_{j+1})^\dagger)_{[\mathcal{T}_j,\mathcal{T}_j]}}\\ &= \left\|\vec{u}_{[\mathcal{T}_j]}\right\|_{\boldsymbol{P}(\boldsymbol{L}(G_{j+1})^\dagger)_{[\mathcal{T}_j,\mathcal{T}_j]}\boldsymbol{P}} = \left\|\vec{u}_{[\mathcal{T}_j]}\right\|_{\mathrm{SC}(G_{j+1},\mathcal{T}_j)^\dagger}\\ &\le e^{\epsilon/2}\left\|\vec{u}_{[\mathcal{T}_j]}\right\|_{\mathrm{SC}(G_j,\mathcal{T}_j)^\dagger} = e^{\epsilon/2}\|\vec{u}\|_{\boldsymbol{A}},\end{aligned} \tag{23}$$

where $\boldsymbol{A} = \begin{bmatrix}0 & 0\\ 0 & \mathrm{SC}(G_j,\mathcal{T}_j)^\dagger\end{bmatrix} \preceq \begin{bmatrix}\boldsymbol{Z}_{j,2} & 0\\ 0 & \mathrm{SC}(G_j,\mathcal{T}_j)^\dagger\end{bmatrix}$, then

$$\left\|\begin{bmatrix}\vec{0}\\ \vec{u}_1\end{bmatrix}\right\|_{\boldsymbol{L}(G_{j+1})^\dagger} \le e^{\epsilon/2}\left\|\boldsymbol{Z}_{j,1}\vec{b}\right\|_{\boldsymbol{A}} = e^{\epsilon/2}\left\|\vec{b}\right\|_{\boldsymbol{Z}_{j,1}^T\boldsymbol{A}\boldsymbol{Z}_{j,1}} \le e^{\epsilon}\left\|\vec{b}\right\|_{\boldsymbol{L}(G_j)^\dagger}. \tag{24}$$

Combining (22), (24) with (12), we have that

$$\left\|\begin{bmatrix}\vec{0}_{[V(G_j)\setminus\mathcal{T}_j]}\\ \vec{v}_1 - \mathrm{SC}(G_{j+1},\mathcal{T}_j)^\dagger\vec{u}_{[\mathcal{T}_j]}\end{bmatrix}\right\|_{\boldsymbol{Z}_{j,1}\boldsymbol{L}(G_j)\boldsymbol{Z}_{j,1}^T} \le 2(t-j)\epsilon e^{2\epsilon}\left\|\vec{b}\right\|_{\boldsymbol{L}(G_j)^\dagger}. \tag{25}$$

Another item in (11) is

$$\begin{aligned}&\left\|\begin{bmatrix}\vec{0}_{[V(G_j)\setminus\mathcal{T}_j]}\\ \mathrm{SC}(G_{j+1},\mathcal{T}_j)^\dagger\vec{u}_{[\mathcal{T}_j]} - \mathrm{SC}(G_j,\mathcal{T}_j)^\dagger\vec{u}_{[\mathcal{T}_j]}\end{bmatrix}\right\|_{\boldsymbol{Z}_{j,1}\boldsymbol{L}(G_j)\boldsymbol{Z}_{j,1}^T}\\ =&\left\|\mathrm{SC}(G_{j+1},\mathcal{T}_j)^\dagger\vec{u}_{[\mathcal{T}_j]} - \mathrm{SC}(G_j,\mathcal{T}_j)^\dagger\vec{u}_{[\mathcal{T}_j]}\right\|_{(\boldsymbol{Z}_{j,1}\boldsymbol{L}(G_j)\boldsymbol{Z}_{j,1}^T)_{[\mathcal{T}_j,\mathcal{T}_j]}}.\end{aligned} \tag{26}$$

By (18), we have

$$\begin{aligned}&\left\|\mathrm{SC}(G_{j+1},\mathcal{T}_j)^\dagger\vec{u}_{[\mathcal{T}_j]} - \mathrm{SC}(G_j,\mathcal{T}_j)^\dagger\vec{u}_{[\mathcal{T}_j]}\right\|_{(\boldsymbol{Z}_{j,1}\boldsymbol{L}(G_j)\boldsymbol{Z}_{j,1}^T)_{[\mathcal{T}_j,\mathcal{T}_j]}}\\ \le& e^{\epsilon/2}\left\|\mathrm{SC}(G_{j+1},\mathcal{T}_j)^\dagger\vec{u}_{[\mathcal{T}_j]} - \mathrm{SC}(G_j,\mathcal{T}_j)^\dagger\vec{u}_{[\mathcal{T}_j]}\right\|_{\mathrm{SC}(G_j,\mathcal{T}_j)}.\end{aligned} \tag{27}$$

By the fact $\mathrm{SC}(G_{j+1},\mathcal{T}_j) \approx_\epsilon \mathrm{SC}(G_j,\mathcal{T}_j)$ and applying Lemma 4.13, we have

$$\left\|\mathrm{SC}(G_{j+1},\mathcal{T}_j)^\dagger\vec{u}_{[\mathcal{T}_j]} - \mathrm{SC}(G_j,\mathcal{T}_j)^\dagger\vec{u}_{[\mathcal{T}_j]}\right\|_{\mathrm{SC}(G_j,\mathcal{T}_j)} \le \epsilon\left\|\vec{u}_{[\mathcal{T}_j]}\right\|_{\mathrm{SC}(G_j,\mathcal{T}_j)^\dagger}. \tag{28}$$

Recall that in (23) and (24) we have

$$\left\|\vec{u}_{[\mathcal{T}_j]}\right\|_{\mathrm{SC}(G_j,\mathcal{T}_j)^\dagger} = \|\vec{u}\|_{\boldsymbol{A}} = \left\|\boldsymbol{Z}_{j,1}\vec{b}\right\|_{\boldsymbol{A}} = \left\|\vec{b}\right\|_{\boldsymbol{Z}_{j,1}^T\boldsymbol{A}\boldsymbol{Z}_{j,1}} \le e^{\epsilon/2}\left\|\vec{b}\right\|_{\boldsymbol{L}(G_j)^\dagger}. \tag{29}$$

Combining (26), (27), (28) and (29) gives

$$\left\|\begin{bmatrix}\vec{0}_{[V(G_j)\setminus\mathcal{T}_j]}\\ \mathrm{SC}(G_{j+1},\mathcal{T}_j)^\dagger\vec{u}_{[\mathcal{T}_j]} - \mathrm{SC}(G_j,\mathcal{T}_j)^\dagger\vec{u}_{[\mathcal{T}_j]}\end{bmatrix}\right\|_{\boldsymbol{Z}_{j,1}\boldsymbol{L}(G_j)\boldsymbol{Z}_{j,1}^T} \le \epsilon e^{\epsilon}\left\|\vec{b}\right\|_{\boldsymbol{L}(G_j)^\dagger}. \tag{30}$$

Finally, combining (25), (30) with (9), we have that for the case j,

$$\left\| Z_{t,1}^T \begin{bmatrix} Z_{t,2}\vec{u}_{[V(G_t)\setminus\mathcal{T}_t]} \\ \vec{v} \end{bmatrix} - \boldsymbol{L}(G_t)^\dagger \vec{b} \right\|_{\boldsymbol{L}(G_t)} \leq \left[2(t-j)\epsilon e^{2\epsilon} + \epsilon e^{\epsilon} + \epsilon)\right] \left\|\vec{b}\right\|_{\boldsymbol{L}(G_j)^\dagger}$$
$$\lesssim 2(t-j+1)\epsilon \left\|\vec{b}\right\|_{\boldsymbol{L}(G_j)^\dagger}.$$

Since $t \leq \log n$, we can prove that

$$\left\| Z_{1,1}^T \begin{bmatrix} Z_{1,2}\vec{u}_{[V(G_1)\setminus\mathcal{T}_1]} \\ \vec{v} \end{bmatrix} - \boldsymbol{L}(G_1)^\dagger \vec{b} \right\|_{\boldsymbol{L}(G_1)} \leq 2\epsilon \log n \left\|\vec{b}\right\|_{\boldsymbol{L}(G_1)^\dagger},$$

that is,

$$\left\|\vec{x} - \boldsymbol{L}(G)^\dagger \vec{b}\right\|_{\boldsymbol{L}(G)} \leq 2\epsilon \log n \left\|\vec{b}\right\|_{\boldsymbol{L}(G)^\dagger}.$$

□

Algorithm 3: Distributed Laplacian Solver

procedure SOLVE(G)
  $G' \leftarrow$ SPECTRALSPARSIFYKX(G)
  $(G_1, Z_{1,1}, Z_{1,2}, \mathcal{T}_1, G_2) \leftarrow$ ULTRASPARSIFY(G', k)
  $\{(G_i, Z_{i,1}, Z_{i,2}, \mathcal{T}_i)\}_{i=2}^t \leftarrow$ BUILDCHAIN(G_2, d, ϵ, k)
  solve $\boldsymbol{L}(G)\vec{x} = \vec{b}$ by preconditioned Chebyshev with G_1 as preconditioner s.t. $\boldsymbol{L}(G_1)\vec{y} = \vec{c}$ is approximated by PSEUDOINVERSEMULTI($\{(G_i, Z_{i,1}, Z_{i,2}, \mathcal{T}_i)\}_{i=1}^t, \vec{c}$).
procedure BUILDCHAIN(G, d, ϵ, k)
  if $|V(G)| \leq k$ **then**
    return ;
  $(Z_1, Z_2, C) \leftarrow$ ELIMINATE(G, d, ϵ).
  $H \leftarrow$ APPROXSC(G, C, ϵ)
  return $(G, Z_1, Z_1, C) \cup$ BUILDCHAIN(H, d, ϵ, k)

Proof. (of Theorem 1) The parameters are set as follow:

- $\epsilon = (\frac{1}{\log \overline{n}})^{10}$.
- $d = (\log\log \overline{n})^2$
- $k = 2^{(\log \overline{n})^{2/3}}$

The correctness of the algorithm is obtained by Lemma 4.3, Lemma 4.9, Lemma 4.10, Theorem 3 and Lemma 4.12.

Now we bound the number of rounds required. By Lemma 4.10, Theorem 3 we have that the Schur-complement chain obtained for graph G satisfying the following conditions

$$|V(G)| = |V(G_1)| \geq |V(G_2)| 2^{O(\sqrt{\log n \log\log n})}/k = |V(G_2)|/k^{1-o(1)}$$

and

$$|V(G_i)|/|V(G_{i+1})| \geq \epsilon^{-2}\log^2 n/0.99^d = 2^{\Theta((\log\log\overline{n})^2)}.$$

Hence, the Schur-complement chain obtained by the BuildChain algorithm is a $(2^{\Theta((\log\log\overline{n})^2)}, \epsilon)$-Schur-complement chain of length $O(\log\overline{n}/(\log\log\overline{n})^2)$. By Lemma 4.12, PseudoinverseMulti for the Schur-complement chain takes $O(\rho\overline{n}^{o(1)}(\overline{n}^{1/2}+D))$ rounds.

Let $f(n,\rho)$ denote the number of rounds required by Algorithms Solve and BuildChain on a graph with n vertices that is ρ-minor distributes to $\overline{G}$, and let $g(n,\rho)$ denote the number of rounds of BuildChain with n vertices that is ρ-minor distributes to $\overline{G}$.

Since preconditioned Chebyshev needs to call the Laplacian solver of the preconditioner $O(\sqrt{k})$ times, by Lemma 4.3, Lemma 4.9, Lemma 4.10, Theorem 3, and Lemma 4.12 we have

$$\begin{aligned} f(n,\rho) &= O\left(\left(\log^{14} n\log^{60}\overline{n}\right)^{(\log\log\overline{n})^2}\left(\rho\overline{n}^{1/2}\log\overline{n}+D\right)\right)+ \\ &\qquad g\left(n/k^{1-o(1)},\rho\right)+O\left(\sqrt{k}\rho\overline{n}^{o(1)}\left(\overline{n}^{1/2}+D\right)\right) \\ &= O\left(\overline{n}^{o(1)}\left(\rho\overline{n}^{1/2}\log\overline{n}+D\right)\right)+g\left(n/k^{1-o(1)},\rho\right). \end{aligned}$$

and

$$\begin{aligned} g(n,\rho) &= O\left(\left(\log^{14} n\log^{60}\overline{n}\right)^{(\log\log\overline{n})^2}\left(\rho\overline{n}^{1/2}\log\overline{n}+D\right)\right)+f(n,2\rho)\log^{10} n\cdot\epsilon^{-3}+g(n/k,2\rho) \\ &= O\left(\overline{n}^{o(1)}\left(\rho\overline{n}^{1/2}+D\right)\right)+\text{polylog}(\overline{n})\,f(n,2\rho)+g(n/k,2\rho). \end{aligned}$$

Since the depth of the recursion is $O(\log\overline{n}/(\log\log\overline{n})^2)$, the overall increase in congestion is at most

$$2^{O\left(\log\overline{n}/(\log\log\overline{n})^2\right)}\rho \leq n^{o(1)}\rho$$

so all the graphs constructed $\overline{n}^{o(1)}\rho$-minor distribute into $\overline{G}$.

Hence, the algorithm Solve takes $\rho\overline{n}^{o(1)}(\overline{n}^{1/2}+D)$ rounds. □

5 Minor Schur Complement

In this section we give the algorithm for constructing minor based approximate Schur complements. Due to the recursive invocation of this routine and solver constructions in Section 4, we can treat the calls to solvers for SDD or Laplacian matrices as black-boxes. The formal guarantees of our constructions are stated in Theorem 3, which is restated below.

Theorem 3. *There is a routine* ApproxSC$(G,\mathcal{T},\epsilon)$ *in the* CONGEST *model that given a graph G with n vertices and m edges that ρ-minor distributes into the communication network $\overline{G}$, a subset of vertices $\mathcal{T}\subseteq V(G)$, an error parameter $\epsilon < 0.1$, and access to a (distributed) Laplacian solver* Solve*, it returns a graph H, represented as a distributed ρ-minor of $\overline{G}$ such that:*

1. *$\mathcal{T}\subseteq V(H)$,*
2. *H has $O(|\mathcal{T}|\epsilon^{-2}\log^2 n)$ edges (and hence at most that many vertices as well).*

3. *The Schur complements of G and H well approximate each other, i.e.,*

$$\mathbf{SC}(G, \mathcal{T}) \approx_\epsilon \mathbf{SC}(H, \mathcal{T}).$$

The cost of this computation consists of:

1. $O(\epsilon^{-3}\log^{10} n)$ *calls to* Solve *with accuracy* $1/\mathrm{poly}(n)$ *on graphs that* 2ρ*-distribute into* $\overline{G}$.
2. *An overhead of* $O(\rho(\overline{n}^{1/2} + D)\epsilon^{-3}\log^{11}\overline{n})$ *rounds.*

Before delving into technical details, we first discuss the high-level connections and differences between our algorithm and that of [LS18].

Comparison to [LS18] Our starting point is the same as [LS18], that is, randomly contracting an edge with probability being equal to its leverage score (and deleting otherwise) is exactly a matrix martingale on the spectral form of the graph. It gives a natural algorithm – iteratively computing leverage scores of edges and sampling them until the variance having been accumulated. The correctness of the algorithm is proved via the matrix martingale concentration inequality.

The main difference lies in the way of obtaining a nearly-linear running time. The leverage scores of all the edges keep changing as some edges get sampled, so a fast algorithm is needed to do better than recomputing the sampling probabilities of all the edges after each edge gets sampled. Li and Schild [LS18] address this issue by showing that a random spanning tree has the correct marginals, and use the fast random spanning tree sampling algorithm [LS18, ALGV20] to obtain such trees. While there are distributed algorithms for sampling spanning trees from unweighted graphs [DSNPT13], partial states of elimination algorithms, namely Schur complements, are naturally weighted. Furthermore, we are unable to directly extend fast random walk simulations to weighted graphs due to the higher congestion of weighted random walks.

Instead, we devise a parallel version of this algorithm based on sampling large subsets of edges independently. We compute a large subset of *steady edges* Z that are mostly uncorrelated, which is obtained by the localization of electrical flows [SRS18]. We then identify such subsets, as well as compute all their effective resistances, using standard sketching methods that are also highly parallel. By ensuring that the size of these sets is at least $1/\mathrm{poly}(\log n)$ of the total number of edges, we are able to ensure the rapid convergence of this process.

In general, we track the cost of our algorithms via three quantities. The first is the number of Laplacian solvers to principal minors of $\boldsymbol{L}(G)$ that we must call, and the second is how many additional rounds of communication between neighbors of G that are necessary, each of which can be simulated in $O(\rho\sqrt{\overline{n}}\log\overline{n} + D)$ rounds in $\overline{G}$ by Lemma 4.3. Finally, we must also ensure that the local computations on vertices $v \in V(G)$ are actually simple minimum/sum aggregations, as each vertex $v \in V(G)$ actually corresponds to a connected component in $\overline{G}$. These can also be simulated in $O(\rho\sqrt{\overline{n}}\log\overline{n} + D)$ rounds in $\overline{G}$ by Lemma 4.3. We note that the computations for solving Laplacian systems and computing leverage scores, etc. only involve matrix-vector multiplications and sampling Bernoulli/Cauchy random variables, which can all be aggregated in a distributed manner.

Distributed storage conventions. In this section, we work with graph G that ρ-minor distributes into the original graph/communication network $\overline{G}$ and is stored distributedly (see Definition 4.1). We work with vertex vectors $\vec{x} \in \mathbb{R}^{V(G)}$. In this case, for a vertex $v \in V(G)$ (corresponding to a connected

component in $\overline{G}$), we assume that the root $V_{map}^{G\to\overline{G}}(v) \in V(\overline{G})$ stores the value of x_v. We also work with edge vectors $\vec{w} \in \mathbb{R}^{E(G)}$ of edge resistances or leverage scores. For an edge $e^G = (u^G, v^G) \in E(G)$, it corresponds to an edge in $\overline{G}$ with endpoints u^H and v^H that store the weight w_{e^G}. When an algorithm is said to compute vertex vectors or edge vectors, it means that these conditions are satisfied.

The remaining part of this section is organized as follows.

1. In Section 5.1, we give the formal definition of steady edges, and present the algorithm for minor based approximate Schur complement.
2. In Section 5.2, we give the algorithm for finding the set of steady edges.
3. In Section 5.3, we prove the correctness of the algorithm in Section 5.2 via matrix martingales, and Theorem 3.

5.1 Sparsification Algorithm

We start by defining the key notion *steady edges*, which are edges that intuitively do not interact with each other much. Here, we emphasize that these steady edges are stochastic, not deterministic.

Definition 5.1. A stochastic subset of edges $Z \subseteq E(H)$ is (α, δ)-steady if

1. (Quadratic form) $\mathbb{E}\left[\sum_{e\in Z} r_e^{-1}\vec{b}_e\vec{b}_e^T\right] \preceq \alpha \boldsymbol{L}(H)$;
2. (Localization) For each edge $e \in Z$, $\sum_{f\neq e\in Z} \frac{|\vec{b}_e^T \boldsymbol{L}(H)^\dagger \vec{b}_f|}{\sqrt{r_e r_f}} \leq \delta$;
3. (Variance) For each edge $e \in Z$,

$$r_e^{-1}\vec{b}_e^T \boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{SC}(H, \mathcal{T}) \end{bmatrix} \boldsymbol{L}(H)^\dagger \vec{b}_e \leq \frac{18|\mathcal{T}|}{|E(H)|}.$$

Intuitively, the *Quadratic form* constraint guarantees that no edge is picked in the set of steady edges with a high probability. The *Localization* constraint bounds the "correlation" between edges, by restricting the electrical flow of each edge e putting on the remaining edges in Z. Finally, the *Variance* constraint says that the induced leverage score of edge e on the Schur complement is bounded, and allows us to control the variance in the matrix martingale analysis.

Now we describe the algorithm for computing a minor Schur complement. First, identify a set of steady edges and approximately compute their leverage scores by the Johnson-Lindenstrauss lemma. Then for each steady edge, contract it with probability being its approximate leverage score, and delete it otherwise. Repeat this process until the size of the resulting graph is small enough. The algorithm pseudocode is shown in algorithm ApproxSC. For this algorithm we have the following theorem.

Theorem 4. *Given a graph G with m edges and a set of terminals $\mathcal{T} \subseteq V(G)$, and parameter $\epsilon \in (0, 1)$, the algorithm* ApproxSC$(G, \mathcal{T}, \epsilon)$ *returns a graph H such that $|E(H)| \leq O(|\mathcal{T}|\epsilon^{-2}\log^2 m)$ and $\mathbf{SC}(H, \mathcal{T}) \approx_\epsilon \mathbf{SC}(G, \mathcal{T})$ with probability at least $1 - 1/\mathrm{poly}(m)$.*

The proof of Theorem 4 is deferred to Section 5.3. Now we describe the subroutines in algorithm ApproxSC.

Algorithm 4: Finding sparsifier of Schur complement onto terminals, but with extra Steiner vertices

procedure ApproxSC($G, \mathcal{T}, \epsilon$)
  Initialize $G^{(0)} \leftarrow G$ and $i \leftarrow 0$.
  Set $\delta \leftarrow \frac{\epsilon}{C \log^2 m}$ with $m = |E(G)|$. ▷ C is a large constant
  while $|E(G^{(i)})| \geq \frac{C|\mathcal{T}|\log^2 m}{\epsilon^2}$ **do**
    $H^{(i)} \leftarrow \textsc{Split}(G^{(i)}, \textsc{LevApx}(e, G^{(i)}, 0.01))$. ▷ Lemmas 5.4 and 5.2
    $Z^{(i)} \leftarrow \textsc{FindSteady}(H^{(i)}, \mathcal{T}, \delta)$. ▷ Lemma 5.5
    For each edge $e \in Z^{(i)}$, set $p_e \leftarrow \textsc{LevApx}(e, H^{(i)}, \delta)$. ▷ Lemma 5.2
    For each edge $e \in Z^{(i)}$, contract e with probability p_e and delete e with probability $1 - p_e$. Perform the contractions and deletions via Corollary 4.7 and let the resulting graph be $I^{(i)}$.
    $G^{(i+1)} \leftarrow \textsc{Unsplit}(I^{(i)})$. ▷ Lemma 5.4
    $i \leftarrow i + 1$.
  $H \leftarrow G^{(i)}$.
  return H.

The Split and sampling process depend on the leverage score of each edge. Instead of computing the leverage scores precisely, we use LevApx to compute approximate leverage scores following the standard random projection scheme devised by Spielman and Srivastava [SS11]. Specifically, the subroutine LevApx satisfies the following guarantees.

Lemma 5.2 (Approximate leverage scores)**.** *Given a graph G' that ρ-minor distributes into $\overline{G}$, an error parameter $\delta > 0$ and the distributed Laplacian solver* Solve*, for each edge $e \in E(G')$, the algorithm* LevApx(e, G', δ) *returns the approximation of* $\mathrm{lev}_{G'}(e) = r_e^{-1} \vec{b}_e^T \boldsymbol{L}(G')^\dagger \vec{b}_e$ *to within a factor of* $1 + \delta$ *with high probability. Furthermore, it takes*

1. $O(\delta^{-2} \log |V(G')|)$ *calls to* Solve *with accuracy* $1/\mathrm{poly}(|V(G)|)$ *on graphs that ρ-minor distribute into $\overline{G}$;*
2. *An additional* $O(\rho\delta^{-2}\sqrt{\overline{n}} \log \overline{n} \log |V(G')| + D)$ *rounds of communication in $\overline{G}$.*

Before proving Lemma 5.2, we present the Johnson-Lindenstrauss lemma which is essential for proving Lemma 5.2.

Lemma 5.3 (Johnson-Lindenstrauss Lemma)**.** *Given n vectors $\vec{v}_1, \cdots, \vec{v}_n \in \mathbb{R}^d$ and a parameter $\delta > 0$, let $\boldsymbol{Q} \in \mathbb{R}^{k \times d}$ with $k \geq 24\delta^{-2} \log n$ be a random $\pm 1/\sqrt{k}$ matrix with each entry being an independent Bernoulli random variable. Then with probability at least $1 - 1/n$,*

$$\|\boldsymbol{Q}(\vec{v}_i - \vec{v}_j)\|_2^2 \approx_\delta \|\vec{v}_i - \vec{v}_j\|_2^2,$$

for all $i, j \in [n]$.

Proof of Lemma 5.2. Recall that the effective resistance of $e = (u, v) \in E(G')$ is defined by $\mathrm{res}_{G'}(e) = \vec{b}_e^T \boldsymbol{L}(G')^\dagger \vec{b}_e$. More specifically, we have

$$\begin{aligned}\mathrm{res}_{G'}(e) &= \vec{b}_e^T \boldsymbol{L}(G')^\dagger \vec{b}_e = \vec{b}_e^T \boldsymbol{L}(G')^\dagger \boldsymbol{L}(G') \boldsymbol{L}(G')^\dagger \vec{b}_e \\ &= \vec{b}_e^T \boldsymbol{L}(G')^\dagger \boldsymbol{B}^T \boldsymbol{R}^{-1} \boldsymbol{B} \boldsymbol{L}(G')^\dagger \vec{b}_e \qquad \text{(Setting } \boldsymbol{L}(G') = \boldsymbol{B}^T \boldsymbol{R}^{-1} \boldsymbol{B}\text{)} \\ &= \left\| \boldsymbol{R}^{-1/2} \boldsymbol{B} \boldsymbol{L}(G')^\dagger \vec{b}_e \right\|_2^2,\end{aligned}$$

which is equal to the squared Euclidean distance between the u-th and v-th column vectors of the matrix $\boldsymbol{R}^{-1/2} \boldsymbol{B} \boldsymbol{L}(G')^\dagger$. To compute the effective resistance for each edge, it suffices to compute the pairwise distances among the column vectors of matrix $\boldsymbol{R}^{-1/2} \boldsymbol{B} \boldsymbol{L}(G')^\dagger$. Applying Johnson-Lindenstruass lemma, we generate a random matrix $\boldsymbol{Q} \in \mathbb{R}^{t \times |E(G')|}$ with $t = O(\delta^{-2} \log |V(G')|)$ such that with high probability

$$\left\| \boldsymbol{Q} \boldsymbol{R}^{-1/2} \boldsymbol{B} \boldsymbol{L}(G')^\dagger \vec{b}_e \right\|_2^2 \approx_\delta \mathrm{res}_{G'}(e),$$

where computing the matrix $\boldsymbol{Q} \boldsymbol{R}^{-1/2} \boldsymbol{B} \boldsymbol{L}(G')^\dagger$ requires matrix multiplication of $\boldsymbol{Q} \boldsymbol{R}^{-1/2} \boldsymbol{B}$ and solving $O(\delta^{-2} \log |V(G')|)$ Laplacian linear systems.

Now we implement the above operations in the distributed settings. First generate $\boldsymbol{Q}$ on the endpoints of the images of the edges in G'. Note that $\boldsymbol{R}^{-1/2}$ is a rescaling of the resistances of the edges, which are also stored together with their endpoints. Let each edge $e \in E(G')$ store the corresponding column of $\boldsymbol{Q} \boldsymbol{R}^{-1/2}$ on both of its endpoints, i.e., both endpoints of $E_{map}^{G' \to \overline{G}}(e)$ store $(\boldsymbol{Q} \boldsymbol{R}^{-1/2})_{:,e}$. Computing the matrix $\boldsymbol{Q} \boldsymbol{R}^{-1/2} \boldsymbol{B}$ is reduced to computing the matrix-vector multiplication $\boldsymbol{Q} \boldsymbol{R}^{-1/2} \boldsymbol{B}_{:,v}$ for each vertex $v \in V(G')$. Note that each edge can choose its direction arbitrarily, as the direction factor backs in when we apply the multiplication by $\vec{b}_e$ at the end, which is also a local step. Lemma 4.3 allows us to perform this process in $O(\rho \delta^{-2} \sqrt{\overline{n}} \log \overline{n} \log |V(G')| + D)$ rounds. □

The algorithm ApproxSC requires that the leverage scores of all the edges are bounded away from 0 and 1, which can be done by the two subroutines, Split and Unsplit that have the following guarantees.

Lemma 5.4 (Split and Unsplit, see Proposition 3.4 and 3.5 in [LS18])**.** *Given a graph G' that ρ-minor distributes into $\overline{G}$ and the approximate leverage score $\mathrm{lev}'_{G'}(e) \approx_{0.01} \mathrm{lev}_{G'}(e)$ for each $e \in E(G')$, the algorithm $\textsc{Split}(G', \mathrm{lev}'_{G'}(e))$ returns a graph H' in $\widetilde{O}(\rho(\sqrt{\overline{n}} + D))$ rounds such that*

1. *H' is electrically equivalent to G';*
2. *H' 2ρ-minor distributes into $\overline{G}$;*
3. *For each edge $e' \in E(H')$, $\mathrm{lev}_{H'}(e') \in [3/16, 13/16]$.*

The algorithm Unsplit *returns a graph resulting from collapsing paths, parallel edges, and removing non-terminal leaves, along with a ρ-minor distribution into $\overline{G}$.*

Proof. In algorithm Split, for each edge $e \in E(G')$, if $\mathrm{lev}'_{G'}(e) \le 1/2$, replace e by a path of two edges e_1 and e_2 with resistance $r_e/2$; if $\mathrm{lev}'_{G'}(e) \ge 1/2$, replace e by two parallel edges e_1 and e_2 with resistance $2r_e$. In the first case, both edges have leverage score $\mathrm{lev}_{H'}(e_1) = \mathrm{lev}_{H'}(e_2) = \frac{1}{2} + \frac{\mathrm{lev}_{G'}(e)}{2}$, in which $\mathrm{lev}_{G'}(e) \in$

$\left[0, \frac{1}{2(1-0.01)}\right]$; in the second case, both edges have leverage score $\text{lev}_{H'}(e_1) = \text{lev}_{H'}(e_2) = \frac{\text{lev}_{G'}(e)}{2}$, in which $\text{lev}_{G'}(e) \in \left[\frac{1}{2(1+0.01)}, 1\right]$. It is easy to verify that each edge $e' \in E(H')$ has $\text{lev}_{H'}(e') \in [3/16, 13/16]$.

The bound on the cost and embeddability follows since each edge is turned into a path of at most two edges. The new vertex can be placed at either endpoints of $E_{map}^{G \to \overline{G}}(e)$, and the congestion on both edge e and the endpoints of e goes up by a factor of 2. This 2-minor embedding of the new graph into G' then meets the definition of Lemma 4.6, which means that H' 2ρ-minor distributes into $\overline{G}$ with an overhead of $\widetilde{O}(\rho(\sqrt{\overline{n}} + D))$ rounds.

The execution of Unsplit is straightforward because the resulting graph is a minor of G', and all changes happen on $O(1)$ neighbors, and only involve local endpoints of edges of G'. Therefore, they can be implemented using $O(\rho)$ rounds of communications among neighbors of G'. □

5.2 Algorithm for Finding Steady Edges

In this section, we present the subroutine FindSteady, shown in algorithm 5, that returns the set of steady edges in algorithm ApproxSC.

Algorithm 5: Given a graph H with a set of terminals $\mathcal{T}$ and parameter δ, return the set of steady edges

procedure FindSteady$(H, \mathcal{T}, \delta)$
  Set $\alpha \leftarrow \frac{\delta}{46 C_{\text{local}} \log^2 |E(H)|}$.
  For each $e \in E(H)$, let $v_e \leftarrow$ DiffApx$(e, H, \mathcal{T})$. ▷ Lemma 5.6
  For each $e \in E(H)$, let $s_e \leftarrow$ ColumnApx$(e, H, E(H))$. ▷ Lemma 5.7
  $Z_1 \leftarrow \{e \in E(H) \mid v_e \le 16|\mathcal{T}|/|E(H)|,\ s_e \le 16 C_{\text{local}} \log^2 |E(H)|\}$.
  Let Z_2 be the set of sampled edges from Z_1 such that each $e \in Z_1$ is sampled with probability α.
  For each $e \in Z_2$, let $s'_e \leftarrow$ ColumnApx(e, H, Z_2). ▷ Lemma 5.6
  $Z \leftarrow \{e \in Z_2 \mid s'_e \le \delta/1.1\}$.
  return Z.

The algorithm FindSteady has the following lemma.

Lemma 5.5. *Given a graph H that ρ-minor distributes into $\overline{G}$, a set of terminals $\mathcal{T} \subseteq V(H)$ and constant $\delta \in (0, 1)$, the algorithm* FindSteady$(H, \mathcal{T}, \delta)$ *has access to the distributed Laplacian solver* Solve *and returns an edge set Z with at least $\alpha|E(H)|/2$ edges in expectation that is (α, δ)-steady. Furthermore, it takes*

1. *$O(\log^2 |V(H)|)$ calls to* Solve *with $1/\text{poly}(|V(H)|)$ error on graphs that ρ-minor distribute into $\overline{G}$;*
2. *An additional $O((\rho\sqrt{\overline{n}} \log \overline{n} \log |V(H)| + D) \log |V(H)|)$ rounds of communication in $\overline{G}$.*

Before proving Lemma 5.5, we first introduce the subroutines DiffApx and ColumnApx.

Lemma 5.6 (Difference sketch, Lemma 1.4 in [LS18])**.** *Given a graph H that ρ-minor distributes into $\overline{G}$ and a set of terminals $\mathcal{T} \subseteq V(H)$, for each edge $e \in E(H)$, the algorithm* DiffApx$(e, H, \mathcal{T})$ *returns an*

approximation to

$$r_e^{-1}\vec{b}_e^T \boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{SC}(H,\mathcal{T}) \end{bmatrix} \boldsymbol{L}(H)^\dagger \vec{b}_e$$

within a factor of 1.1 *with high probability. Furthermore, it requires*

1. $O(\log |V(H)|)$ *calls to* Solve *with accuracy* $1/\mathrm{poly}(|V(H)|)$ *on graphs that* ρ*-minor distribute into* $\overline{G}$*;*
2. *An additional* $O(\rho\sqrt{\overline{n}}\log\overline{n}\log|V(H)|+D)$ *rounds of communication in* $\overline{G}$*.*

Proof. By Lemma 2.3, we have that

$$\begin{bmatrix} 0 & 0 \\ 0 & \mathrm{SC}(H,\mathcal{T}) \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{SC}(H,\mathcal{T}) \end{bmatrix} \boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{SC}(H,\mathcal{T}) \end{bmatrix},$$

and then

$$\begin{aligned} & r_e^{-1}\vec{b}_e^T \boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{SC}(H,\mathcal{T}) \end{bmatrix} \boldsymbol{L}(H)^\dagger \vec{b}_e \\ =& r_e^{-1}\vec{b}_e^T \boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{SC}(H,\mathcal{T}) \end{bmatrix} \boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{SC}(H,\mathcal{T}) \end{bmatrix} \boldsymbol{L}(H)^\dagger \vec{b}_e. \end{aligned} \tag{31}$$

Using the fact $\boldsymbol{L}(H)^\dagger = \boldsymbol{L}(H)^\dagger \boldsymbol{L}(H) \boldsymbol{L}(H)^\dagger$ and setting $\boldsymbol{L}(H) = \boldsymbol{B}^T \boldsymbol{R}^{-1} \boldsymbol{B}$, (31) becomes

$$r_e^{-1}\vec{b}_e^T \boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{SC}(H,\mathcal{T}) \end{bmatrix} \boldsymbol{L}(H)^\dagger \boldsymbol{B}^T \boldsymbol{R}^{-1} \boldsymbol{B} \boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{SC}(H,\mathcal{T}) \end{bmatrix} \boldsymbol{L}(H)^\dagger \vec{b}_e. \tag{32}$$

Formulating (32) in another way, it becomes

$$r_e^{-1} \left\| \boldsymbol{R}^{-1/2} \boldsymbol{B} \boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{SC}(H,\mathcal{T}) \end{bmatrix} \boldsymbol{L}(H)^\dagger \vec{b}_e \right\|_2^2. \tag{33}$$

Combining (31), (32) and (33), we have

$$r_e^{-1}\vec{b}_e^T \boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{SC}(H,\mathcal{T}) \end{bmatrix} \boldsymbol{L}(H)^\dagger \vec{b}_e = r_e^{-1} \left\| \boldsymbol{R}^{-1/2} \boldsymbol{B} \boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{SC}(H,\mathcal{T}) \end{bmatrix} \boldsymbol{L}(H)^\dagger \vec{b}_e \right\|_2^2.$$

By Lemma 5.3, we generate a random matrix $\boldsymbol{Q} \in \mathbb{R}^{t \times |E(H)|}$ with $t = O(\log |V(H)|)$ such that

$$\begin{aligned} & r_e^{-1} \left\| \boldsymbol{Q}\boldsymbol{R}^{-1/2} \boldsymbol{B} \boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{SC}(H,\mathcal{T}) \end{bmatrix} \boldsymbol{L}(H)^\dagger \vec{b}_e \right\|_2^2 \\ \approx_{0.1} & r_e^{-1}\vec{b}_e^T \boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{SC}(H,\mathcal{T}) \end{bmatrix} \boldsymbol{L}(H)^\dagger \vec{b}_e. \end{aligned}$$

Therefore, the round complexity is determined by computing the matrix

$$\boldsymbol{Q}\boldsymbol{R}^{-1/2} \boldsymbol{B} \boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{SC}(H,\mathcal{T}) \end{bmatrix} \boldsymbol{L}(H)^\dagger.$$

Firstly, we can compute the matrix $\boldsymbol{A}_1 = \boldsymbol{Q}\boldsymbol{R}^{-1/2}\boldsymbol{B}$ as Lemma 5.2, which takes

$$O(\rho\sqrt{\overline{n}}\log\overline{n}\log|V(H)|+D)$$

rounds. Then multiplying by $\boldsymbol{L}(H)^\dagger$ requires calling $O(\log|V(H)|)$ times Solve on $\boldsymbol{L}(H)$, which corresponds to the t rows of the matrix $\boldsymbol{A}_1$.

Recall that

$$\mathrm{SC}(H,\mathcal{T}) = \boldsymbol{L}(H)_{[\mathcal{T},\mathcal{T}]} - \boldsymbol{L}(H)_{[\mathcal{T},V(H)\setminus\mathcal{T}]}\left(\boldsymbol{L}(H)_{[V(H)\setminus\mathcal{T},V(H)\setminus\mathcal{T}]}\right)^{-1}\boldsymbol{L}(H)_{[V(H)\setminus\mathcal{T},\mathcal{T}]},$$

then we have

$$\begin{aligned}&\boldsymbol{A}_1\begin{bmatrix}0 & 0\\ 0 & \mathrm{SC}(H,\mathcal{T})\end{bmatrix} = \boldsymbol{A}_{1[:,\mathcal{T}]}\mathrm{SC}(H,\mathcal{T})\\ =&\boldsymbol{A}_{1[:,\mathcal{T}]}\boldsymbol{L}(H)_{[\mathcal{T},\mathcal{T}]} - \boldsymbol{A}_{1[:,\mathcal{T}]}\boldsymbol{L}(H)_{[\mathcal{T},V(H)\setminus\mathcal{T}]}\left(\boldsymbol{L}(H)_{[V(H)\setminus\mathcal{T},V(H)\setminus\mathcal{T}]}\right)^{-1}\boldsymbol{L}(H)_{[V(H)\setminus\mathcal{T},\mathcal{T}]},\end{aligned}$$

which can be computed by calling t times Solve on $\boldsymbol{L}(H)_{[V(H)\setminus\mathcal{T},V(H)\setminus\mathcal{T}]}$ and three matrix-matrix multiplications; each matrix-matrix consists of t matrix-vector multiplications.

Let the matrix $\boldsymbol{A}_2 = \boldsymbol{A}_1\begin{bmatrix}0 & 0\\ 0 & \mathrm{SC}(H,\mathcal{T})\end{bmatrix}$, then computing $\boldsymbol{A}_2\boldsymbol{L}(H)^\dagger$ requires calling t times Solve on $\boldsymbol{L}(H)$.

Therefore, the algorithm DiffApx$(e,H,\mathcal{T})$ requires calling $O(\log|V(H)|)$ times Solve on graphs that ρ-minor distributes into $\overline{G}$ and additional $O(\rho\sqrt{\overline{n}}\log\overline{n}\log|V(H)|+D)$ rounds of communication. □

Lemma 5.7 (Analog to Proposition 4.3 in [LS18]). *Given a graph H that ρ-minor distributes into $\overline{G}$ and a subset $W\subseteq E(H)$, for each edge $e\in W$, the algorithm* ColumnApx(e,H,W) *returns an approximation to*

$$\sum_{f\neq e\in W}\frac{|\vec{b}_e^T\boldsymbol{L}(H)^\dagger\vec{b}_f|}{\sqrt{r_e r_f}}$$

within a factor of 1.1 *with high probability. Furthermore, it takes*

1. $O(\log^2|V(H)|)$ *calls to* Solve *with accuracy* $1/\mathrm{poly}(|V(H)|)$ *on graphs that ρ-minor distribute into $\overline{G}$;*
2. *An additional* $O((\rho\sqrt{\overline{n}}\log\overline{n}\log|V(H)|+D)\log|V(H)|)$ *rounds of communication in $\overline{G}$.*

The proof of Lemma 5.7 depends on the following ℓ_1 sketch.

Lemma 5.8 (Theorem 3 in [Ind06]). *Given an integer $d\geq 1$ and two constants $0<\delta,\epsilon<1$, there exists a matrix $\boldsymbol{C}\in\mathbb{R}^{t\times d}$ with $t=O(\epsilon^{-2}\log(1/\delta))$ and an algorithm* Recover$(\vec{u},d,\delta,\epsilon)$ *such that*

1. *The entries of $\boldsymbol{C}$ are independently sampled from a Cauchy distribution;*
2. *For any vector $\vec{v}\in\mathbb{R}^d$, the algorithm* Recover$(\boldsymbol{C}\vec{v},d,\delta,\epsilon)$ *outputs an estimator r such that*

$$r\approx_\epsilon\|\vec{v}\|_1,$$

with probability $1-\delta$.

Proof of Lemma 5.7. We use the ℓ_1 sketch in Lemma 5.8 in a way analogous to the ℓ_2 resistance estimation procedure in Lemma 5.2. Randomly partitioning the set W such that $W = U \cup (W \setminus U)$ and $\Pr[e \in U] = 1/2$ for each $e \in W$, then for each $e \in U$, we have

$$\mathbb{E}\left[\sum_{f \in W \setminus U} \frac{|\vec{b}_e^T \boldsymbol{L}(H)^\dagger \vec{b}_f|}{\sqrt{r_e r_f}}\right] = \frac{1}{2} \sum_{f \neq e \in W} \frac{|\vec{b}_e^T \boldsymbol{L}(H)^\dagger \vec{b}_f|}{\sqrt{r_e r_f}}.$$

Denote the random variable $X_i = \sum_{f \in W \setminus U} \frac{|\vec{b}_e^T \boldsymbol{L}(H)^\dagger \vec{b}_f|}{\sqrt{r_e r_f}}$, and repeat $t_1 = O(\log |V(H)|)$ times to obtain $X_1, \cdots, X_{t_1}$. Let X be $X = \sum_{i=1}^{t_1} X_i$, then by Chernoff bound we have that with high probability,

$$\frac{2X}{t_1} \approx_{0.1} \sum_{f \neq e \in W} \frac{|\vec{b}_e^T \boldsymbol{L}(H)^\dagger \vec{b}_f|}{\sqrt{r_e r_f}}.$$

Let $\boldsymbol{R}_{W \setminus U}$ and $\boldsymbol{B}_{W \setminus U}$ be the diagonal resistance matrix and incidence matrix restricted to $W \setminus U$, and $\vec{v} = r_e^{-1/2} \boldsymbol{R}_{W \setminus U}^{-1/2} \boldsymbol{B}_{W \setminus U} \boldsymbol{L}(H)^\dagger \vec{b}_e$. Then we have

$$\|\vec{v}\|_1 = \sum_{f \in W \setminus U} \frac{|\vec{b}_e^T \boldsymbol{L}(H)^\dagger \vec{b}_f|}{\sqrt{r_e r_f}}.$$

By Lemma 5.8, setting the matrix $\boldsymbol{C} \in \mathbb{R}^{t_2 \times |W \setminus U|}$ with $t_2 = O(\log |V(H)|)$, then with probability $1 - 1/\mathrm{poly}(|V(H)|)$ the algorithm $\textsc{Recover}(\boldsymbol{C}\vec{v}, |W \setminus U|, 1/\mathrm{poly}(|V(H)|), 0.01)$ outputs a 0.01-approximation of the quantity $\|\vec{v}\|_1$.

Now we analyze the round complexity which is analogous to the ℓ_2 sketch presented in the proof of Lemma 5.2. Computing the matrix $\boldsymbol{C}\boldsymbol{R}_{W \setminus U}^{-1/2} \boldsymbol{B}_{W \setminus U} \boldsymbol{L}(H)^\dagger$ consists of computing $\boldsymbol{C}\boldsymbol{R}_{W \setminus U}^{-1/2} \boldsymbol{B}_{W \setminus U}$, which takes $O(\rho \sqrt{\overline{n}} \log \overline{n} \log |V(H)| + D)$ rounds, and solving t_2 Laplacian linear systems in $\boldsymbol{L}(H)$. Note that we repeat that for t_1 times, therefore, the algorithm $\textsc{ColumnApx}(e, H, W)$ requires calling $O(\log^2 |V(H)|)$ times $\textsc{Solve}$ with accuracy $1/\mathrm{poly}(|V(H)|)$ on graphs that ρ-minor distribute into $\overline{G}$, and an additional $O((\rho \sqrt{\overline{n}} \log \overline{n} \log |V(H)| + D) \log |V(H)|)$ rounds of communication in $\overline{G}$. □

Lemma 5.9. *The graph H with a set of terminals $\mathcal{T} \subseteq V(H)$ satisfies that*

$$\sum_{e \in E(H)} r_e^{-1} \vec{b}_e^T \boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{SC}(H, \mathcal{T}) \end{bmatrix} \boldsymbol{L}(H)^\dagger \vec{b}_e \leq |\mathcal{T}|. \tag{34}$$

Proof. Since the LHS of (34) is a scalar, it holds that

$$\begin{aligned} &\sum_{e \in E(H)} r_e^{-1} \vec{b}_e^T \boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{SC}(H, \mathcal{T}) \end{bmatrix} \boldsymbol{L}(H)^\dagger \vec{b}_e \\ = &\sum_{e \in E(H)} \mathrm{Tr}\left(r_e^{-1} \vec{b}_e^T \boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{SC}(H, \mathcal{T}) \end{bmatrix} \boldsymbol{L}(H)^\dagger \vec{b}_e\right). \end{aligned} \tag{35}$$

By the properties of the trace operation, (35) becomes

$$\begin{aligned}&\sum_{e\in E(H)} \mathrm{Tr}\left(\boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{SC}(H,\mathcal{T}) \end{bmatrix} \boldsymbol{L}(H)^\dagger r_e^{-1}\vec{b}_e\vec{b}_e^T\right)\\ =&\mathrm{Tr}\left(\boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{SC}(H,\mathcal{T}) \end{bmatrix} \boldsymbol{L}(H)^\dagger \sum_{e\in E(H)} r_e^{-1}\vec{b}_e\vec{b}_e^T\right).\end{aligned} \tag{36}$$

By the fact $\sum_{e\in E(H)} r_e^{-1}\vec{b}_e\vec{b}_e^T = \boldsymbol{L}(H)$ and the properties of trace operation, (36) becomes

$$\begin{aligned}&\mathrm{Tr}\left(\boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{SC}(H,\mathcal{T}) \end{bmatrix} \boldsymbol{L}(H)^\dagger \boldsymbol{L}(H)\right) = \mathrm{Tr}\left(\boldsymbol{L}(H)^\dagger \boldsymbol{L}(H)\boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{SC}(H,\mathcal{T}) \end{bmatrix}\right)\\ =&\mathrm{Tr}\left(\boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{SC}(H,\mathcal{T}) \end{bmatrix}\right) = \mathrm{Tr}\left(\left(\boldsymbol{L}(H)^\dagger\right)_{[\mathcal{T},\mathcal{T}]} \mathrm{SC}(H,\mathcal{T})\right)\\ =&\mathrm{Tr}\left(\left(\boldsymbol{L}(H)^\dagger\right)_{[\mathcal{T},\mathcal{T}]} \mathrm{SC}(H,\mathcal{T})\mathrm{SC}(H,\mathcal{T})^\dagger\mathrm{SC}(H,\mathcal{T})\right)\\ =&\mathrm{Tr}\left(\mathrm{SC}(H,\mathcal{T})\left(\boldsymbol{L}(H)^\dagger\right)_{[\mathcal{T},\mathcal{T}]} \mathrm{SC}(H,\mathcal{T})\mathrm{SC}(H,\mathcal{T})^\dagger\right).\end{aligned} \tag{37}$$

Lemma 2.3 gives $\mathrm{SC}(H,\mathcal{T})\left(\boldsymbol{L}(H)^\dagger\right)_{[\mathcal{T},\mathcal{T}]}\mathrm{SC}(H,\mathcal{T}) = \mathrm{SC}(H,\mathcal{T})$, so (37) gets

$$\mathrm{Tr}\left(\mathrm{SC}(H,\mathcal{T})\mathrm{SC}(H,\mathcal{T})^\dagger\right) = \mathrm{Tr}(\boldsymbol{P}) = |\mathcal{T}| - 1 \le |\mathcal{T}|,$$

where $\boldsymbol{P}$ is the projection matrix of the space spanned by $\mathrm{SC}(H,\mathcal{T})$.

This completes the proof. □

Now we review the *flow localization* theorem.

Theorem 5 (Flow localization [SRS18])**.** *For a graph H, let*

$$s_e = \sum_{f\in E(H)} \frac{|\vec{b}_e^T \boldsymbol{L}(H)^\dagger \vec{b}_f|}{\sqrt{r_e r_f}},$$

then there exists an universal constant C_{local} such that $\sum_{e\in E(H)} s_e \le C_{\mathrm{local}}|E(H)|\log^2|E(H)|$.

Now we are ready to prove Lemma 5.5.

Proof of Lemma 5.5. We prove that $Z \subseteq E(H)$ is (α,δ)-steady according to the Definition 5.1.

1. (Quadratic form)

$$\mathbb{E}\left[\sum_{e\in Z} r_e^{-1}\vec{b}_e\vec{b}_e^T\right] \preceq \mathbb{E}\left[\sum_{e\in Z_2} r_e^{-1}\vec{b}_e\vec{b}_e^T\right] = \alpha \sum_{e\in Z_1} r_e^{-1}\vec{b}_e\vec{b}_e^T \preceq \alpha\boldsymbol{L}(H).$$

2. (Localization) By Lemma 5.7, we have that for each $e \in Z_2$, $s'_e \approx_{0.1} \sum_{f \neq e \in Z_2} \frac{|\vec{b}_e^T \boldsymbol{L}(H)^\dagger \vec{b}_f|}{\sqrt{r_e r_f}}$, which gives $\sum_{f \neq e \in Z_2} \frac{|\vec{b}_e^T \boldsymbol{L}(H)^\dagger \vec{b}_f|}{\sqrt{r_e r_f}} \leq 1.1 s'_e$. Combining with line 8 in algorithm FindSteady, we know that for each edge $e \in Z$,

$$\sum_{f \neq e \in Z} \frac{|\vec{b}_e^T \boldsymbol{L}(H)^\dagger \vec{b}_f|}{\sqrt{r_e r_f}} \leq \delta.$$

3. (Variance) Lemma 5.6 gives that for each $e \in E(H)$,

$$v_e \approx_{0.1} r_e^{-1} \vec{b}_e^T \boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{SC}(H, \mathcal{T}) \end{bmatrix} \boldsymbol{L}(H)^\dagger \vec{b}_e,$$

which implies that $r_e^{-1} \vec{b}_e^T \boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{SC}(H, \mathcal{T}) \end{bmatrix} \boldsymbol{L}(H)^\dagger \vec{b}_e \leq 1.1 v_e$. Combining with line 5 in algorithm FindSteady, we have that for each edge $e \in Z$,

$$r_e^{-1} \vec{b}_e^T \boldsymbol{L}(H)^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{SC}(H, \mathcal{T}) \end{bmatrix} \boldsymbol{L}(H)^\dagger \vec{b}_e \leq \frac{18|\mathcal{T}|}{|E(H)|}.$$

Now we bound $|Z|$. By Lemma 5.6 and Lemma 5.9, we know that at most $1.1|E(H)|/16$ edges for $e \in E(H)$ satisfy that

$$v_e \geq 16|\mathcal{T}|/|E(H)|.$$

Similarly, Lemma 5.7 and Theorem 5 tell us that at most $1.1|E(H)|/16$ edges satisfy that

$$s_e \geq 16 C_{\text{local}} \log^2 |E(H)|.$$

We conclude that

$$|Z_1| \geq (1 - 2.2/16)|E(H)| \geq 13|E(H)|/16.$$

In addition,

$$\mathbb{E}[|Z_2|] = \alpha |Z_1| \geq 13\alpha |E(H)|/16.$$

By the definition of Z_2, we know that for each $e \in Z_2$,

$$\mathbb{E}[s'_e] \leq 16\alpha C_{\text{local}} \log^2 |E(H)|.$$

By Markov's inequality, we have that

$$\Pr[s'_e \geq \delta/1.1] \leq \frac{\mathbb{E}[s'_e]}{\delta/1.1} \leq \frac{16\alpha C_{\text{local}} \log^2 |E(H)|}{\delta/1.1} = \frac{44}{115}.$$

Therefore,

$$\mathbb{E}[|Z|] \geq \left(1 - \frac{44}{115}\right) \mathbb{E}[|Z_2|] \geq \alpha |E(H)|/2.$$

The round complexity mainly comes from Lemma 5.6 and Lemma 5.7, and the remaining steps in algorithm FindSteady can be trivially implemented. □

Finally, we bound the number of while loops, which is a key ingredient to prove Theorem 4 and Theorem 3.

Lemma 5.10. *The while loop in algorithm* ApproxSC *executes* $O(\alpha^{-1}\log m)$ *times with*

$$\alpha = \frac{\delta}{46C_{\text{local}}\log^2 m} = \frac{\epsilon}{46C_{\text{local}}C\log^4 m}$$

with probability at least $1 - 1/\text{poly}(m)$.

Proof. To bound the number of iterations, it suffices to argue that

$$\mathbb{E}[|E(G^{(i+1)})|] \le (1 - \Omega(\alpha))|E(G^{(i)})|.$$

Recall that in Lemma 5.5, we have $\mathbb{E}[|Z^{(i)}|] \ge \alpha|E(G^{(i)})|/2$, so it suffices to argue that each original edge of $G^{(i)}$ is removed with at least a constant probability, even considering the Split operation. We break the analysis into two cases.

Case 1: e is split into two parallel edges e_1 and e_2. Recall that by the definition of Split, both e_1 and e_2 have leverage score in $[3/16, 13/16]$. Therefore, if $e_1 \in Z^{(i)}$, then it will be contracted with probability at least $3/16$. In that case both e_1 and e_2 disappear, as desired.

Case 2: e is split into a path consisting of e_1 and e_2. Recall that by the definition of Split, both e_1 and e_2 have leverage score in $[3/16, 13/16]$. Therefore, if $e_1 \in Z^{(i)}$, the it will be deleted with probability at least $3/16$. Then e_2 becomes a leaf, so it will be removed during the Unsplit operation, as desired. □

5.3 Matrix Martingale Analysis of Approximation

In this section, we prove Theorem 4 by defining several stochastic sequences of matrices that capture the change of the quadratic form of the Schur complement. We also prove Theorem 3.

Let τ denote the final value of i in algorithm ApproxSC. Recall that Lemma 5.10 gives $\tau = O(\frac{\log m}{\alpha})$. Let the hidden constant be C', i.e., $\tau = \frac{C'\log m}{\alpha}$. For $0 \le i \le \tau$ and $0 \le t \le |Z^{(i)}|$, let $e_{i,t}$ be the t-th edge in $Z^{(i)}$ under an arbitrary ordering, and

$$\widehat{Y}^{(i,0)} = S_0 \boldsymbol{L}(H^{(i)})^\dagger S_0^T, \tag{38}$$

with $S_0 = [0, \text{SC}(G,\mathcal{T})^{1/2}]$, then we have the following iteration equation

$$\widehat{Y}^{(i,t+1)} = \begin{cases} \widehat{Y}^{(i,t)} + r_{e_{i,t}}^{-1}(1-p_{e_{i,t}})^{-1} S_0 \boldsymbol{L}(H^{(i)})^\dagger \vec{b}_{e_{i,t}} \vec{b}_{e_{i,t}}^T \boldsymbol{L}(H^{(i)})^\dagger S_0^T & \text{if } e_{i,t} \text{ is deleted,} \\ \widehat{Y}^{(i,t)} - r_{e_{i,t}}^{-1} p_{e_{i,t}}^{-1} S_0 \boldsymbol{L}(H^{(i)})^\dagger \vec{b}_{e_{i,t}} \vec{b}_{e_{i,t}}^T \boldsymbol{L}(H^{(i)})^\dagger S_0^T & \text{if } e_{i,t} \text{ is contracted.} \end{cases} \tag{39}$$

Especially for $\widehat{Y}^{(0,0)}$, by Lemma 2.3 we have

$$\begin{aligned} \widehat{Y}^{(0,0)} &= S_0 \boldsymbol{L}(G)^\dagger S_0^T = \text{SC}(G,\mathcal{T})^{1/2} \left(\boldsymbol{L}(G)^\dagger\right)_{[\mathcal{T},\mathcal{T}]} \text{SC}(G,\mathcal{T})^{1/2} \\ &= \text{SC}(G,\mathcal{T})^{\dagger/2} \text{SC}(G,\mathcal{T}) \text{SC}(G,\mathcal{T})^{\dagger/2} = \boldsymbol{P}, \end{aligned}$$

where $\boldsymbol{P}$ is the projection matrix of the space spanned by $\text{SC}(G,\mathcal{T})$.

In the proof of Theorem 4, if we assume that $\text{SC}(H^{(i)},\mathcal{T}) \approx_{0.1} \text{SC}(G,\mathcal{T})$ for all (i,t), then we have the following claim to bound $\left\|S_0 \boldsymbol{L}(H^{(i)})^\dagger S_0^T\right\|_2$.

Claim 5.11. $\left\|S_0 \boldsymbol{L}(H^{(i)})^\dagger S_0^T\right\|_2 \le 1.1$.

Proof. The fact $\mathrm{SC}(H^{(i)}, \mathcal{T}) \approx_{0.1} \mathrm{SC}(G, \mathcal{T})$ gives us $\mathrm{SC}(H^{(i)}, \mathcal{T})^\dagger \approx_{0.1} \mathrm{SC}(G, \mathcal{T})^\dagger$, i.e.,

$$\mathrm{SC}(H^{(i)}, \mathcal{T})^\dagger \preceq 1.1 \cdot \mathrm{SC}(G, \mathcal{T})^\dagger.$$

By Lemma 2.3, it holds that

$$\boldsymbol{P}\left(\boldsymbol{L}(H^{(i)})^\dagger\right)_{[\mathcal{T},\mathcal{T}]}\boldsymbol{P} \preceq 1.1 \cdot \boldsymbol{P}\left(\boldsymbol{L}(G)^\dagger\right)_{[\mathcal{T},\mathcal{T}]}\boldsymbol{P}.$$

Moreover, we have

$$\mathrm{SC}(G,\mathcal{T})^{1/2}\left(\boldsymbol{L}(H^{(i)})^\dagger\right)_{[\mathcal{T},\mathcal{T}]}\mathrm{SC}(G,\mathcal{T})^{1/2} \preceq 1.1 \cdot \mathrm{SC}(G,\mathcal{T})^{1/2}\left(\boldsymbol{L}(G)^\dagger\right)_{[\mathcal{T},\mathcal{T}]}\mathrm{SC}(G,\mathcal{T})^{1/2}. \tag{40}$$

Lemma 2.3 has

$$\mathrm{SC}(G,\mathcal{T})\left(\boldsymbol{L}(G)^\dagger\right)_{[\mathcal{T},\mathcal{T}]}\mathrm{SC}(G,\mathcal{T}) = \mathrm{SC}(G,\mathcal{T}), \tag{41}$$

which implies that

$$\mathrm{SC}(G,\mathcal{T})^{1/2}\left(\boldsymbol{L}(G)^\dagger\right)_{[\mathcal{T},\mathcal{T}]}\mathrm{SC}(G,\mathcal{T})^{1/2} = \mathrm{SC}(G,\mathcal{T})^{\dagger/2}\mathrm{SC}(G,\mathcal{T})\mathrm{SC}(G,\mathcal{T})^{\dagger/2} \tag{42}$$

by multiplying the both sides of the LHS and RHS of (41) by $\mathrm{SC}(G,\mathcal{T})^{\dagger/2}$. Combing (40) and (42), we have

$$\mathrm{SC}(G,\mathcal{T})^{1/2}\left(\boldsymbol{L}(H^{(i)})^\dagger\right)_{[\mathcal{T},\mathcal{T}]}\mathrm{SC}(G,\mathcal{T})^{1/2} \preceq 1.1 \cdot \mathrm{SC}(G,\mathcal{T})^{\dagger/2}\mathrm{SC}(G,\mathcal{T})\mathrm{SC}(G,\mathcal{T})^{\dagger/2}. \tag{43}$$

Furthermore,

$$\begin{aligned}\text{LHS of (43)} &= S_0 \boldsymbol{L}(H^{(i)})^\dagger S_0^T,\\ \text{RHS of (43)} &= 1.1 \cdot \boldsymbol{P}\end{aligned}$$

Therefore, $S_0 \boldsymbol{L}(H^{(i)})^\dagger S_0^T \preceq 1.1 \cdot \boldsymbol{P}$ and $\left\|S_0 \boldsymbol{L}(H^{(i)})^\dagger S_0^T\right\|_2 \le 1.1$. □

Now we define the difference sequence for $\widehat{Y}^{(i,t)}$ by

$$\boldsymbol{X}^{(i,t)} = \begin{cases} 0 & \text{if } t = 0,\\ \widehat{Y}^{(i,t)} - \widehat{Y}^{(i,t-1)} & \text{if } t > 0.\end{cases}$$

The operator norm of $\boldsymbol{X}^{(i,t)}$ has the following bound.

Lemma 5.12. *For all (i,t), it holds that*

$$\left\|\boldsymbol{X}^{(i,t)}\right\|_2 \le \frac{162|\mathcal{T}|}{|E(H^{(i)})|}.$$

Proof. By the definition of $\boldsymbol{X}^{(i,t)}$ and equation (39), we have

$$\begin{aligned}\left\|\boldsymbol{X}^{(i,t)}\right\|_2 =& \left\|\widehat{\boldsymbol{Y}}^{(i,t)} - \widehat{\boldsymbol{Y}}^{(i,t-1)}\right\|_2 \\ \leq& (r_{e_{i,t}} \cdot \min\{1-p_{e_{i,t}}, p_{e_{i,t}}\})^{-1} \left\|\boldsymbol{S}_0 \boldsymbol{L}(H^{(i)})^\dagger \vec{b}_{e_{i,t}} \vec{b}_{e_{i,t}}^T \boldsymbol{L}(H^{(i)})^\dagger \boldsymbol{S}_0^T\right\|_2.\end{aligned} \tag{44}$$

By Lemma 5.4 and Lemma 5.2, we have that for each $e_{i,t} \in Z^{(i)}$, $\mathrm{lev}_{H^{(i)}}(e_{i,t}) \in [3/16, 13/16]$ and $p_{e_{i,t}} \approx_\delta \mathrm{lev}_{H^{(i)}}(e_{i,t})$. Since $\delta \leq 0.01$, we have $p_{e_{i,t}} \in [1/8, 7/8]$. Then (44) becomes

$$\begin{aligned}\left\|\boldsymbol{X}^{(i,t)}\right\|_2 \leq& 8 r_{e_{i,t}}^{-1} \vec{b}_{e_{i,t}}^T \boldsymbol{L}(H^{(i)})^\dagger \boldsymbol{S}_0^T \boldsymbol{S}_0 \boldsymbol{L}(H^{(i)})^\dagger \vec{b}_{e_{i,t}} \\ \leq& 8 r_{e_{i,t}}^{-1} \vec{b}_{e_{i,t}}^T \boldsymbol{L}(H^{(i)})^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{SC}(G,\mathcal{T}) \end{bmatrix} \boldsymbol{L}(H^{(i)})^\dagger \vec{b}_{e_{i,t}}.\end{aligned}$$

The assumption $\mathbf{SC}(H^{(i)},\mathcal{T}) \approx_{0.1} \mathbf{SC}(G,\mathcal{T})$ implies that

$$\begin{bmatrix} 0 & 0 \\ 0 & \mathbf{SC}(G,\mathcal{T}) \end{bmatrix} \leq 1.1 \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{SC}(H^{(i)},\mathcal{T}) \end{bmatrix},$$

which gives

$$\left\|\boldsymbol{X}^{(i,t)}\right\|_2 \leq 9 r_{e_{i,t}}^{-1} \vec{b}_{e_{i,t}}^T \boldsymbol{L}(H^{(i)})^\dagger \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{SC}(H^{(i)},\mathcal{T}) \end{bmatrix} \boldsymbol{L}(H^{(i)})^\dagger \vec{b}_{e_{i,t}}.$$

Combining with condition 3 in Definition 5.1, we have

$$\left\|\boldsymbol{X}^{(i,t)}\right\|_2 \leq \frac{162|\mathcal{T}|}{|E(H^{(i)})|}.$$

□

Claim 5.13. *For each fixed $0 \leq i \leq \tau$, the sequence $\widehat{\boldsymbol{Y}}^{(i,0)}, \cdots, \widehat{\boldsymbol{Y}}^{(i,|Z^{(i)}|)}$ is a martingale.*

Proof. By the definition of $\boldsymbol{X}^{(i,t)}$ and Lemma 5.12, we have

$$\left\|\widehat{\boldsymbol{Y}}^{(i,t)}\right\|_2 \leq \left\|\widehat{\boldsymbol{Y}}^{(i,t-1)}\right\|_2 + \left\|\boldsymbol{X}^{(i,t)}\right\|_2 \leq \left\|\widehat{\boldsymbol{Y}}^{(i,t-1)}\right\|_2 + \frac{162|\mathcal{T}|}{|E(H^{(i)})|},$$

which implies that for a fixed i and $0 \leq t \leq |Z^{(i)}|$,

$$\mathbb{E}\left[\left\|\widehat{\boldsymbol{Y}}^{(i,t)}\right\|_2\right] < \infty. \tag{45}$$

Now considering the quantity $\mathbb{E}\left[\widehat{\boldsymbol{Y}}^{(i,t+1)} \middle| \widehat{\boldsymbol{Y}}^{(i,0)}, \cdots, \widehat{\boldsymbol{Y}}^{(i,t)}\right]$, we have

$$\begin{aligned}&\mathbb{E}\left[\widehat{\boldsymbol{Y}}^{(i,t+1)} \middle| \widehat{\boldsymbol{Y}}^{(i,0)}, \cdots, \widehat{\boldsymbol{Y}}^{(i,t)}\right] = \mathbb{E}\left[\widehat{\boldsymbol{Y}}^{(i,t+1)} \middle| \widehat{\boldsymbol{Y}}^{(i,t)}\right] \\ =& (1-p_{e_{i,t}}) \cdot \left(\widehat{\boldsymbol{Y}}^{(i,t)} + r_{e_{i,t}}^{-1}(1-p_{e_{i,t}})^{-1} \boldsymbol{S}_0 \boldsymbol{L}(H^{(i)})^\dagger \vec{b}_{e_{i,t}} \vec{b}_{e_{i,t}}^T \boldsymbol{L}(H^{(i)})^\dagger \boldsymbol{S}_0^T\right) \\ &+ p_{e_{i,t}} \cdot \left(\widehat{\boldsymbol{Y}}^{(i,t)} - r_{e_{i,t}}^{-1} p_{e_{i,t}}^{-1} \boldsymbol{S}_0 \boldsymbol{L}(H^{(i)})^\dagger \vec{b}_{e_{i,t}} \vec{b}_{e_{i,t}}^T \boldsymbol{L}(H^{(i)})^\dagger \boldsymbol{S}_0^T\right) \\ =& \widehat{\boldsymbol{Y}}^{(i,t)}.\end{aligned} \tag{46}$$

Putting (45) and (46) together proves this claim. □

However, the UnSplit and Split operations lead to $\widehat{Y}^{(i+1,0)} \neq \widehat{Y}^{(i,|Z^{(i)}|)}$. In order to treat the τ sequences as a whole, we define the new sequence $Y^{(i,t)}$ such that

$$\begin{cases} Y^{(0,0)} = \widehat{Y}^{(0,0)} = P, \\ Y^{(i,t)} - Y^{(i,t-1)} = X^{(i,t)}, \\ Y^{(i,0)} = Y^{(i-1,|Z^{(i)}|)}. \end{cases}$$

We prove Theorem 4 by considering two parts: the martingale $Y^{(i,t)}$ and the errors resulting from $\widehat{Y}^{(i+1,0)} - \widehat{Y}^{(i,|Z^{(i)}|)}$. Before that, we prove that the sequence $Y^{(i,t)}$ is also a martingale following the proof of Claim 5.13.

- Note that $\mathbb{E}\left[\left\|Y^{(i,t)}\right\|_2\right] < \infty$ since

$$\left\|Y^{(i,t)}\right\|_2 \leq \left\|Y^{(i,t-1)}\right\|_2 + \left\|X^{(i,t)}\right\|_2 \leq \left\|Y^{(i,t-1)}\right\|_2 + \frac{162|\mathcal{T}|}{|E(H^{(i)})|}.$$

- For the special case, $\mathbb{E}\left[Y^{(i+1,0)} \middle| Y^{(i,0)}, \cdots Y^{(i,|Z^{(i)}|)}\right] = \mathbb{E}\left[Y^{(i,Z^{(i)})}\right]$. More generally,

$$\begin{aligned} &\mathbb{E}\left[Y^{(i,t+1)} \middle| Y^{(i,0)}, \cdots, Y^{(i,t)}\right] = \mathbb{E}\left[Y^{(i,t+1)} \middle| Y^{(i,t)}\right] \\ =&Y^{(i,t)} + \mathbb{E}\left[X^{(i,t+1)}\right] = Y^{(i,t)} + \mathbb{E}\left[\widehat{Y}^{(i,t+1)} - \widehat{Y}^{(i,t)}\right] = Y^{(i,t)}. \end{aligned}$$

This completes the proof.

Lemma 5.14. *Let C be a sufficiently large constant in algorithm* ApproxSC. *Then for all (i,t), it holds that*

$$\left\|W^{(i,t)}\right\|_2 \leq \frac{\epsilon^2}{100\log m}.$$

Proof. Recall that in Lemma 2.7

$$W^{(k)} = \sum_{j=1}^{k} \mathbb{E}\left[\left(X^{(j)}\right)^2 \middle| X^{(j-1)}\right].$$

Here we have

$$W^{(i+1,0)} - W^{(i,0)} = \sum_{1\leq t\leq |Z^{(i)}|} \mathbb{E}\left[\left(X^{(i,t)}\right)^2 \middle| X^{(i,t-1)}\right] = \sum_{1\leq t\leq |Z^{(i)}|} \mathbb{E}\left[\left(X^{(i,t)}\right)^2\right], \tag{47}$$

and set $W^{(0,0)} = 0$. By the fact $\left(X^{(i,t)}\right)^2 \leq \left\|X^{(i,t)}\right\|_2 \cdot X^{(i,t)}$, we have

$$\sum_{1\leq t\leq |Z^{(i)}|} \mathbb{E}\left[\left(X^{(i,t)}\right)^2\right] \leq \sum_{1\leq t\leq |Z^{(i)}|} \left\|X^{(i,t)}\right\|_2 \mathbb{E}\left[X^{(i,t)}\right]. \tag{48}$$

By Lemma 5.12, it holds that

$$\sum_{1\le t\le|Z^{(i)}|}\left\|\boldsymbol{X}^{(i,t)}\right\|_2\mathbb{E}\left[\boldsymbol{X}^{(i,t)}\right]\preceq\frac{162|\mathcal{T}|}{|E(H^{(i)})|}\sum_{1\le t\le|Z^{(i)}|}\mathbb{E}\left[\boldsymbol{X}^{(i,t)}\right]. \tag{49}$$

By the definition of $\boldsymbol{X}^{(i,t)}$ and the fact $p_{e_{i,t}}\in[1/8,7/8]$, we have

$$\begin{aligned}\sum_{1\le t\le|Z^{(i)}|}\mathbb{E}\left[\boldsymbol{X}^{(i,t)}\right]&\preceq 8\sum_{1\le t\le|Z^{(i)}|}\mathbb{E}\left[r_{e_{i,t}}^{-1}\boldsymbol{S}_0\boldsymbol{L}(H^{(i)})^\dagger\vec{b}_{e_{i,t}}\vec{b}_{e_{i,t}}^T\boldsymbol{L}(H^{(i)})^\dagger\boldsymbol{S}_0^T\right]\\&=8\cdot\boldsymbol{S}_0\boldsymbol{L}(H^{(i)})^\dagger\mathbb{E}\left[\sum_{1\le t\le|Z^{(i)}|}r_{e_{i,t}}^{-1}\vec{b}_{e_{i,t}}\vec{b}_{e_{i,t}}^T\right]\boldsymbol{L}(H^{(i)})^\dagger\boldsymbol{S}_0^T.\end{aligned} \tag{50}$$

By the condition 1 in Definition 5.1, (51) becomes

$$\sum_{1\le t\le|Z^{(i)}|}\mathbb{E}\left[\boldsymbol{X}^{(i,t)}\right]\preceq 8\alpha\cdot\boldsymbol{S}_0\boldsymbol{L}(H^{(i)})^\dagger\boldsymbol{L}(H^{(i)})\boldsymbol{L}(H^{(i)})^\dagger\boldsymbol{S}_0^T=8\alpha\cdot\boldsymbol{S}_0\boldsymbol{L}(H^{(i)})^\dagger\boldsymbol{S}_0^T. \tag{51}$$

Combining (51), (49), (48) with (47), we have

$$\boldsymbol{W}^{(i+1,0)}-\boldsymbol{W}^{(i,0)}\preceq\frac{1296|\mathcal{T}|\alpha}{|E(H^{(i)})|}\boldsymbol{S}_0\boldsymbol{L}(H^{(i)})^\dagger\boldsymbol{S}_0^T.$$

and

$$\left\|\boldsymbol{W}^{(i+1,0)}-\boldsymbol{W}^{(i,0)}\right\|_2\le\frac{1296|\mathcal{T}|\alpha}{|E(H^{(i)})|}\left\|\boldsymbol{S}_0\boldsymbol{L}(H^{(i)})^\dagger\boldsymbol{S}_0^T\right\|_2\le\frac{1426|\mathcal{T}|\alpha}{|E(H^{(i)})|},$$

where the last inequality follows from Claim 5.11.

For any i,t, we have

$$\begin{aligned}\left\|\boldsymbol{W}^{(i,t)}\right\|_2=\left\|\boldsymbol{W}^{(i,t)}-\boldsymbol{W}^{(0,0)}\right\|_2&\le\sum_{0\le k\le i}\left\|\boldsymbol{W}^{(k+1,0)}-\boldsymbol{W}^{(k,0)}\right\|_2\le\tau\cdot\frac{1426|\mathcal{T}|\alpha}{|E(H^{(i)})|}\\&=\frac{C'\log m}{\alpha}\cdot\frac{1426|\mathcal{T}|\alpha}{|E(H^{(i)})|}\le\frac{C'\log m}{\alpha}\cdot\frac{1426|\mathcal{T}|\alpha\epsilon^2}{C|\mathcal{T}|\log^2 m}\le\frac{\epsilon^2}{100\log m}.\end{aligned}$$

□

Now combining Lemma 5.12 and Lemma 5.14, we have the following lemma.

Lemma 5.15. *With probability at least* $1-1/\mathrm{poly}(m)$*, for all* (i,t)*, it holds that* $\|\boldsymbol{Y}^{(i,t)}-\boldsymbol{P}\|_2\le\epsilon/2$.

Proof. Lemma 5.12 gives $\left\|\boldsymbol{X}^{(i,t)}\right\|_2\le\frac{162|\mathcal{T}|}{|E(H^{(i)})|}$. The algorithm ApproxSC implies that $|E(H^{(i)})|=\Omega\left(\frac{|\mathcal{T}|\log^2 m}{\epsilon^2}\right)$. Then we have $\left\|\boldsymbol{X}^{(i,t)}\right\|_2=O\left(\frac{\epsilon^2}{\log^2 m}\right)$. Setting $R=\frac{\epsilon^2}{100\log^2 m}$ and $\sigma^2=\frac{\epsilon^2}{100\log m}$, and applying Lemma 2.7, we obtain

$$\begin{aligned}&\Pr\left[\exists(i,t)\middle|\left\|\boldsymbol{Y}^{(i,t)}-\boldsymbol{Y}^{(0,0)}\right\|_2\ge\epsilon/2,\ \left\|\boldsymbol{W}^{(i,t)}\right\|_2\le\sigma^2\right]\\\le&2|\mathcal{T}|\cdot\exp\left(\frac{-\epsilon^2/12}{\sigma^2+R\epsilon/6}\right)\le 2|\mathcal{T}|\cdot\exp\left(\frac{-\epsilon^2/12}{\frac{\epsilon^2}{100\log m}+\frac{\epsilon^3}{600\log^2 m}}\right)=1/\mathrm{poly}(m).\end{aligned}$$

Moreover, since $\left\|\boldsymbol{W}^{(i,t)}\right\|_2 \leq \sigma^2$, we have

$$\Pr\left[\left\|\boldsymbol{Y}^{(i,t)} - \boldsymbol{Y}^{(0,0)}\right\|_2 \geq \epsilon/2\right] \leq 1/\mathrm{poly}(m),$$

which gives that

$$\Pr\left[\left\|\boldsymbol{Y}^{(i,t)} - \boldsymbol{P}\right\|_2 \leq \epsilon/2\right] \geq 1 - 1/\mathrm{poly}(m).$$

□

Since Split and Unsplit preserve the Schur complement,we conclude that $\widehat{\boldsymbol{Y}}^{(i+1,0)}$ satisfies

$$\begin{aligned}
\widehat{\boldsymbol{Y}}^{(i+1,0)} &= \boldsymbol{S}_0 \boldsymbol{L}(H^{(i+1)})^\dagger \boldsymbol{S}_0^T \\
&= \mathrm{SC}(G,\mathcal{T})^{1/2} \left(\boldsymbol{L}(H^{(i+1)})^\dagger\right)_{[\mathcal{T},\mathcal{T}]} \mathrm{SC}(G,\mathcal{T})^{1/2} \\
&= \mathrm{SC}(G,\mathcal{T})^{1/2} \left(\boldsymbol{L}(I^{(i)})^\dagger\right)_{[\mathcal{T},\mathcal{T}]} \mathrm{SC}(G,\mathcal{T})^{1/2} \\
&= \boldsymbol{S}_0 \boldsymbol{L}(I^{(i)})^\dagger \boldsymbol{S}_0^T,
\end{aligned}$$

where $I^{(i)}$ corresponds to $H^{(i)}$ in algorithm ApproxSC. The Laplacian matrices $\boldsymbol{L}(I^{(i)})$ and $\boldsymbol{L}(H^{(i)})$ satisfy the following relation

$$\boldsymbol{L}(I^{(i)}) = \boldsymbol{L}(H^{(i)}) + \boldsymbol{U}\boldsymbol{C}^{(i)}\boldsymbol{U}^T,$$

where $\boldsymbol{C}^{(i)}$ is a diagonal matrix such that

$$\boldsymbol{C}^{(i)}_{ff} = \begin{cases} -1 & \text{if } f \in Z^{(i)} \text{ is deleted} \\ \infty & \text{if } f \in Z^{(i)} \text{ is contracted} \end{cases}$$

and $\boldsymbol{U} = \boldsymbol{B}^T_{Z^{(i)}} \boldsymbol{R}^{-1/2}_{Z^{(i)}}$, where $\boldsymbol{B}_{Z^{(i)}}$ and $\boldsymbol{R}_{Z^{(i)}}$ are the matrices with restriction of $\boldsymbol{B}$ and $\boldsymbol{R}$ to the indices corresponding to the set $Z^{(i)}$. Then by Woodbury matrix formula (see Lemma 2.6) we have

$$\boldsymbol{L}(I^{(i)})^\dagger = \boldsymbol{L}(H^{(i)})^\dagger - \boldsymbol{L}(H^{(i)})^\dagger \boldsymbol{U} \left((\boldsymbol{C}^{(i)})^{-1} + \boldsymbol{U}^T \boldsymbol{L}(H^{(i)})^\dagger \boldsymbol{U}\right)^{-1} \boldsymbol{U}^T \boldsymbol{L}(H^{(i)})^\dagger. \tag{52}$$

Furthermore, multiplying the left side and right side of (52) by $\boldsymbol{S}_0$ and $\boldsymbol{S}_0^T$ respectively, one can obtain

$$\widehat{\boldsymbol{Y}}^{(i+1,0)} - \widehat{\boldsymbol{Y}}^{(i,0)} = -\boldsymbol{S}_0 \boldsymbol{L}(H^{(i)})^\dagger \boldsymbol{U} \left((\boldsymbol{C}^{(i)})^{-1} + \boldsymbol{U}^T \boldsymbol{L}(H^{(i)})^\dagger \boldsymbol{U}\right)^{-1} \boldsymbol{U}^T \boldsymbol{L}(H^{(i)})^\dagger \boldsymbol{S}_0^T. \tag{53}$$

For the quantity $\widehat{\boldsymbol{Y}}^{(i,|Z^{(i)}|)} - \widehat{\boldsymbol{Y}}^{(i,0)}$, by virtue of the equation (39), we have

$$\begin{aligned}
\widehat{\boldsymbol{Y}}^{(i,|Z^{(i)}|)} - \widehat{\boldsymbol{Y}}^{(i,0)} &= \sum_{0 \leq t < |Z^{(i)}|} \widehat{\boldsymbol{Y}}^{(i,t+1)} - \widehat{\boldsymbol{Y}}^{(i,t)} \\
&= \boldsymbol{S}_0 \boldsymbol{L}(H^{(i)})^\dagger \boldsymbol{U} \boldsymbol{P}^{(i)} \boldsymbol{U}^T \boldsymbol{L}(H^{(i)})^\dagger \boldsymbol{S}_0^T,
\end{aligned} \tag{54}$$

where $\boldsymbol{P}^{(i)}$ is a diagonal matrix such that

$$P^{(i)}_{ff} = \begin{cases} (1-p_f)^{-1} & \text{if } f \text{ is deleted,} \\ -p_f^{-1} & \text{if } f \text{ is contracted.} \end{cases}$$

Subtracting (54) from (53) gives

$$\begin{aligned} &\widehat{Y}^{(i+1,0)} - \widehat{Y}^{(i,|Z^{(i)}|)} \\ =& -S_0 L(H^{(i)})^\dagger U \left[\left((C^{(i)})^{-1} + U^T L(H^{(i)})^\dagger U \right)^{-1} + P^{(i)} \right] U^T L(H^{(i)})^\dagger S_0^T. \end{aligned}$$

Define $\widehat{X}^{(i)} = \widehat{Y}^{(i+1,0)} - \widehat{Y}^{(i,|Z^{(i)}|)}$. Before bounding $\left\| \widehat{X}^{(i)} \right\|_2$, we have the following lemma.

Lemma 5.16. $\left\| \left((C^{(i)})^{-1} + U^T L(H^{(i)})^\dagger U \right)^{-1} + P^{(i)} \right\|_2 \le 36\delta.$

Proof. Let $D^{(i)}$ be the diagonal matrix with entries being the diagonal entries of the matrix $(C^{(i)})^{-1} + U^T L(H^{(i)})^\dagger U$, specifically,

$$D^{(i)}_{ff} = \begin{cases} -1 + \mathrm{lev}_{H^{(i)}}(f) & \text{if } f \text{ is deleted,} \\ \mathrm{lev}_{H^{(i)}}(f) & \text{if } f \text{ is contracted.} \end{cases}$$

Define another matrix $Q^{(i)}$ by

$$Q^{(i)} = D^{(i)} - \left((C^{(i)})^{-1} + U^T L(H^{(i)})^\dagger U \right).$$

Note that all the diagonal entries of $Q^{(i)}$ are 0. Considering the summation of non-diagonal entries of $Q^{(i)}$, which is equal to the summation of non-diagonal entries of $U^T L(H^{(i)})^\dagger U$, we have

$$\sum_{f \neq g} \left| Q^{(i)}_{fg} \right| = \sum_{f \neq g \in Z^{(i)}} \frac{|\vec{b}_e^T L(H^{(i)})^\dagger \vec{b}_f|}{\sqrt{r_f r_g}} \le \delta,$$

where the inequality follows from the localization condition of Definition 5.1. Lemma 2.8 gives

$$\left\| Q^{(i)} \right\|_2 \le \sum_{f \neq g} \left| Q^{(i)}_{fg} \right|,$$

and thus $\left\| Q^{(i)} \right\|_2 \le \delta$.

Consider the left side of the target inequality,

$$\begin{aligned} &\left\| \left((C^{(i)})^{-1} + U^T L(H^{(i)})^\dagger U \right)^{-1} + P^{(i)} \right\|_2 \\ =& \left\| \left(D^{(i)} - Q^{(i)} \right)^{-1} + P^{(i)} \right\|_2 \le \left\| \left(D^{(i)} - Q^{(i)} \right)^{-1} - (D^{(i)})^{-1} \right\|_2 + \left\| (D^{(i)})^{-1} + P^{(i)} \right\|_2, \end{aligned} \tag{55}$$

in which

$$\begin{aligned}\left\|\left(\boldsymbol{D}^{(i)}-\boldsymbol{Q}^{(i)}\right)^{-1}-(\boldsymbol{D}^{(i)})^{-1}\right\|_2 &= \left\|\left(\boldsymbol{D}^{(i)}-\boldsymbol{Q}^{(i)}\right)^{-1}\boldsymbol{Q}^{(i)}(\boldsymbol{D}^{(i)})^{-1}\right\|_2\\ \leq \left\|\left(\boldsymbol{D}^{(i)}-\boldsymbol{Q}^{(i)}\right)^{-1}\right\|_2 \left\|\boldsymbol{Q}^{(i)}\right\|_2 \left\|(\boldsymbol{D}^{(i)})^{-1}\right\|_2 &\leq \left(\frac{3}{16}-\delta\right)^{-1}\cdot\delta\cdot\frac{16}{3}\leq 30\delta,\end{aligned} \tag{56}$$

and

$$\left\|(\boldsymbol{D}^{(i)})^{-1}+\boldsymbol{P}^{(i)}\right\|_2 = \max_{f\in Z^{(i)}}\left\{\left|\frac{1}{-1+\mathrm{lev}_{H^{(i)}}(f)}+\frac{1}{1-p_f}\right|,\left|\frac{1}{\mathrm{lev}_{H^{(i)}}(f)}-\frac{1}{p_f}\right|\right\}\leq 6\delta, \tag{57}$$

where $p_f \approx_\delta \mathrm{lev}_{H^{(i)}}(f)$.

Substituting (56) and (57) to (55) completes the proof. □

Based on Lemma 5.16, we can give the following bounds on $\widehat{\boldsymbol{X}}^{(i)}$.

Lemma 5.17. *For all (i,t), it holds that*

$$\begin{aligned}\left\|\widehat{\boldsymbol{X}}^{(i)}\right\|_2 &\leq 40\delta,\\ \left\|\mathbb{E}_{Z^{(i)}}\left[\widehat{\boldsymbol{X}}^{(i)}\right]\right\|_2 &\leq 40\alpha\delta,\\ \left\|\mathbb{E}_{Z^{(i)}}\left[\left(\widehat{\boldsymbol{X}}^{(i)}\right)^2\right]\right\|_2 &\leq 1600\alpha\delta^2.\end{aligned}$$

Proof. Recall that

$$\begin{aligned}\widehat{\boldsymbol{X}}^{(i)} &= \widehat{\boldsymbol{Y}}^{(i+1,0)}-\widehat{\boldsymbol{Y}}^{(i,|Z^{(i)}|)}\\ &= -\boldsymbol{S}_0\boldsymbol{L}(H^{(i)})^\dagger\boldsymbol{U}\left[\left((\boldsymbol{C}^{(i)})^{-1}+\boldsymbol{U}^T\boldsymbol{L}(H^{(i)})^\dagger\boldsymbol{U}\right)^{-1}+\boldsymbol{P}^{(i)}\right]\boldsymbol{U}^T\boldsymbol{L}(H^{(i)})^\dagger\boldsymbol{S}_0^T,\end{aligned}$$

then

$$\begin{aligned}\left\|\widehat{\boldsymbol{X}}^{(i)}\right\|_2 &= \left\|\boldsymbol{S}_0\boldsymbol{L}(H^{(i)})^\dagger\boldsymbol{U}\left[\left((\boldsymbol{C}^{(i)})^{-1}+\boldsymbol{U}^T\boldsymbol{L}(H^{(i)})^\dagger\boldsymbol{U}\right)^{-1}+\boldsymbol{P}^{(i)}\right]\boldsymbol{U}^T\boldsymbol{L}(H^{(i)})^\dagger\boldsymbol{S}_0^T\right\|_2\\ &\leq \left\|\left((\boldsymbol{C}^{(i)})^{-1}+\boldsymbol{U}^T\boldsymbol{L}(H^{(i)})^\dagger\boldsymbol{U}\right)^{-1}+\boldsymbol{P}^{(i)}\right\|_2\left\|\boldsymbol{S}_0\boldsymbol{L}(H^{(i)})^\dagger\boldsymbol{U}\boldsymbol{U}^T\boldsymbol{L}(H^{(i)})^\dagger\boldsymbol{S}_0^T\right\|_2\\ &\leq 36\delta\cdot\left\|\boldsymbol{S}_0\boldsymbol{L}(H^{(i)})^\dagger\boldsymbol{L}(H^{(i)})\boldsymbol{L}(H^{(i)})^\dagger\boldsymbol{S}_0^T\right\|_2\\ &= 36\delta\cdot\left\|\boldsymbol{S}_0\boldsymbol{L}(H^{(i)})\boldsymbol{S}_0^T\right\|_2\\ &\leq 40\delta, \qquad \text{(By Claim 5.11)}\end{aligned}$$

where the second inequality follows from Lemma 5.16 and the fact $\boldsymbol{U}\boldsymbol{U}^T\preceq\boldsymbol{L}(H^{(i)})$.

For $\mathbb{E}_{Z^{(i)}}\left[\widehat{\boldsymbol{X}}^{(i)}\right]$, we have

$$\begin{aligned}&\left\|\mathbb{E}_{Z^{(i)}}\left[\widehat{\boldsymbol{X}}^{(i)}\right]\right\|_2\\ =&\left\|\mathbb{E}_{Z^{(i)}}\left[\boldsymbol{S}_0\boldsymbol{L}(H^{(i)})^\dagger\boldsymbol{U}\left[\left((\boldsymbol{C}^{(i)})^{-1}+\boldsymbol{U}^T\boldsymbol{L}(H^{(i)})^\dagger\boldsymbol{U}\right)^{-1}+\boldsymbol{P}^{(i)}\right]\boldsymbol{U}^T\boldsymbol{L}(H^{(i)})^\dagger\boldsymbol{S}_0^T\right]\right\|_2.\end{aligned}$$

Since $\left((C^{(i)})^{-1}+U^TL(H^{(i)})^\dagger U\right)^{-1}+P^{(i)} \preceq \left\|\left((C^{(i)})^{-1}+U^TL(H^{(i)})^\dagger U\right)^{-1}+P^{(i)}\right\|_2 \cdot I$, we have

$$\begin{aligned}
&\mathbb{E}_{Z^{(i)}}\left[S_0L(H^{(i)})^\dagger U\left[\left((C^{(i)})^{-1}+U^TL(H^{(i)})^\dagger U\right)^{-1}+P^{(i)}\right]U^TL(H^{(i)})^\dagger S_0^T\right]\\
\preceq&\left\|\left((C^{(i)})^{-1}+U^TL(H^{(i)})^\dagger U\right)^{-1}+P^{(i)}\right\|_2\mathbb{E}_{Z^{(i)}}\left[S_0L(H^{(i)})^\dagger UU^TL(H^{(i)})^\dagger S_0^T\right]
\end{aligned}$$

and

$$\begin{aligned}
&\left\|\mathbb{E}_{Z^{(i)}}\left[\widehat{X}^{(i)}\right]\right\|_2\\
\le&\left\|\left((C^{(i)})^{-1}+U^TL(H^{(i)})^\dagger U\right)^{-1}+P^{(i)}\right\|_2\left\|\mathbb{E}_{Z^{(i)}}\left[S_0L(H^{(i)})^\dagger UU^TL(H^{(i)})^\dagger S_0^T\right]\right\|_2 \qquad (58)\\
=&\left\|\left((C^{(i)})^{-1}+U^TL(H^{(i)})^\dagger U\right)^{-1}+P^{(i)}\right\|_2\left\|S_0L(H^{(i)})^\dagger\mathbb{E}_{Z^{(i)}}\left[UU^T\right]L(H^{(i)})^\dagger S_0^T\right\|_2.
\end{aligned}$$

By Lemma 5.16 and $U=B^T_{Z^{(i)}}R^{-1/2}_{Z^{(i)}}$, (58) becomes

$$\begin{aligned}
\left\|\mathbb{E}_{Z^{(i)}}\left[\widehat{X}^{(i)}\right]\right\|_2\le&36\delta\left\|S_0L(H^{(i)})^\dagger\mathbb{E}_{Z^{(i)}}\left[\sum_{f\in Z^{(i)}}r_f^{-1}\vec{b}_f\vec{b}_f^T\right]L(H^{(i)})^\dagger S_0^T\right\|_2\\
\le&36\alpha\delta\left\|S_0L(H^{(i)})S_0^T\right\|_2\\
\le&40\alpha\delta,
\end{aligned}$$

where the second inequality follows from the condition 1 of Definition 5.1 and the last inequality follows from Claim 5.11.

For $\mathbb{E}_{Z^{(i)}}\left[\left(\widehat{X}^{(i)}\right)^2\right]$, using the fact $\left(\widehat{X}^{(i)}\right)^2 \preceq \left\|\widehat{X}^{(i)}\right\|_2\cdot\widehat{X}^{(i)}$, we have

$$\left\|\mathbb{E}_{Z^{(i)}}\left[\left(\widehat{X}^{(i)}\right)^2\right]\right\|_2 \le \left\|\widehat{X}^{(i)}\right\|_2\left\|\mathbb{E}_{Z^{(i)}}\left[\widehat{X}^{(i)}\right]\right\|_2 \le 40\delta\cdot40\alpha\delta = 1600\alpha\delta^2.$$

□

Now we bound $\left\|\widehat{Y}^{(i,0)}-Y^{(i,0)}\right\|_2$.

Lemma 5.18. *With probability at least* $1-1/\mathrm{poly}(m)$, $\left\|\widehat{Y}^{(i,0)}-Y^{(i,0)}\right\|_2\le\epsilon/2$.

Proof. We first consider the difference matrix $\widehat{Y}^{(i,0)}-\widehat{Y}^{(0,0)}$, which can be decomposed into two parts: the summation of $\widehat{Y}^{(j+1,0)}-\widehat{Y}^{(j,|Z^{(j)}|)}$ and the summation of $\sum_{k=1}^{|Z^{(j)}|}\left(\widehat{Y}^{(j,k)}-\widehat{Y}^{(j,k-1)}\right)$ for $j=0,\cdots,i-1$,

$$\widehat{Y}^{(i,0)}-\widehat{Y}^{(0,0)}=\sum_{j=0}^{i-1}\left(\widehat{Y}^{(j+1,0)}-\widehat{Y}^{(j,|Z^{(j)}|)}\right)+\sum_{j=0}^{i-1}\sum_{k=1}^{|Z^{(j)}|}\left(\widehat{Y}^{(j,k)}-\widehat{Y}^{(j,k-1)}\right).$$

Recall that $\widehat{X}^{(j)} = \widehat{Y}^{(j+1,0)} - \widehat{Y}^{(j,|Z^{(j)}|)}$ and $\widehat{Y}^{(j,k)} - \widehat{Y}^{(j,k-1)} = Y^{(j,k)} - Y^{(j,k-1)}$, then

$$\widehat{Y}^{(i,0)} - \widehat{Y}^{(0,0)} = \sum_{j=0}^{i-1} \widehat{X}^{(j)} + \sum_{j=0}^{i-1} \sum_{k=1}^{|Z^{(j)}|} \left(Y^{(j,k)} - Y^{(j,k-1)} \right). \tag{59}$$

Recall that $Y^{(j,0)} = Y^{(j-1,|Z^{(j-1)}|)}$, so we have

$$\begin{aligned} \sum_{j=0}^{i-1} \sum_{k=1}^{|Z^{(j)}|} \left(Y^{(j,k)} - Y^{(j,k-1)} \right) &= \sum_{j=0}^{i-1} \sum_{k=1}^{|Z^{(j)}|} \left(Y^{(j,k)} - Y^{(j,k-1)} \right) + \sum_{j=1}^{i} \left(Y^{(j,0)} - Y^{(j-1,|Z^{(j-1)}|)} \right) \\ &= Y^{(i,0)} - Y^{(0,0)}. \end{aligned} \tag{60}$$

Substituting (60) to (59) gives

$$\widehat{Y}^{(i,0)} - \widehat{Y}^{(0,0)} = \sum_{j=0}^{i-1} \widehat{X}^{(j)} + Y^{(i,0)} - Y^{(0,0)}.$$

Recall that $\widehat{Y}^{(0,0)} = Y^{(0,0)}$, then we can obtain

$$\widehat{Y}^{(i,0)} - Y^{(i,0)} = \sum_{j=0}^{i-1} \widehat{X}^{(j)}. \tag{61}$$

Define the new sequence $U^{(i)}$ such that

$$U^{(i)} = \widehat{X}^{(i)} - \mathbb{E}_{Z^{(i)}} \left[\widehat{X}^{(i)} \right], \tag{62}$$

and $\{V^{(i)}\}$ to be the martingale with difference sequence $U^{(i)}$ and $V^{(0)} = 0$. In order to apply Lemma 2.7 to martingale $\{V^{(i)}\}$, we first give the bounds of $\left\|U^{(i)}\right\|_2$ and $\left\|\sum_i \mathbb{E}_{Z^{(i)}} \left[(U^{(i)})^2 \middle| U^{(i-1)} \right]\right\|_2$. By the definition of $U^{(i)}$ and Lemma 5.17, we have

$$\begin{aligned} \left\|U^{(i)}\right\|_2 = \left\|\widehat{X}^{(i)} - \mathbb{E}_{Z^{(i)}} \left[\widehat{X}^{(i)} \right]\right\|_2 &\le \left\|\widehat{X}^{(i)}\right\|_2 + \left\|\mathbb{E}_{Z^{(i)}} \left[\widehat{X}^{(i)} \right]\right\|_2 \\ &\le 40\delta + 40\alpha\delta \le 80\delta. \end{aligned}$$

In addition,

$$\begin{aligned} \left\|\mathbb{E}_{Z^{(i)}} \left[(U^{(i)})^2 \middle| U^{(i-1)} \right]\right\|_2 = \left\|\mathbb{E}_{Z^{(i)}} \left[(U^{(i)})^2 \right]\right\|_2 &= \left\|\mathbb{V} \left[\widehat{X}^{(i)} \right]\right\|_2 \\ &\le \left\|\mathbb{E}_{Z^{(i)}} \left[\left(\widehat{X}^{(i)} \right)^2 \right]\right\|_2 \le 1600\alpha\delta^2, \end{aligned}$$

where the first inequality follows from the fact $\mathbb{V} \left[\widehat{X}^{(i)} \right] \le \mathbb{E}_{Z^{(i)}} \left[\left(\widehat{X}^{(i)} \right)^2 \right]$. Moreover, by triangle inequality, we have

$$\begin{aligned} \left\|\sum_i \mathbb{E}_{Z^{(i)}} \left[(U^{(i)})^2 \middle| U^{(i-1)} \right]\right\|_2 &\le \sum_i \left\|\mathbb{E}_{Z^{(i)}} \left[(U^{(i)})^2 \middle| U^{(i-1)} \right]\right\|_2 \le \tau \cdot 1600\alpha\delta^2 \\ &\le \frac{C' \log m}{\alpha} \cdot 1600\alpha\delta^2 = 1600C'\delta^2 \log m. \end{aligned}$$

Setting $R = 80\delta$ and $\sigma^2 = 1600C'\delta^2 \log m$, Lemma 2.7 gives that

$$\begin{aligned}&\Pr\left[\exists i \left| \left\|\boldsymbol{V}^{(i)}\right\|_2 \geq \epsilon/4, \left\|\sum_i \mathbb{E}_{Z^{(i)}}\left[(\boldsymbol{U}^{(i)})^2 \middle| \boldsymbol{U}^{(i-1)}\right]\right\|_2 \leq \sigma^2\right.\right]\\ \leq& 2|\mathcal{T}| \cdot \exp\left(\frac{-\epsilon^2/48}{1600C'\delta^2 \log m + 20\delta\epsilon/3}\right) = 1/\mathrm{poly}(m),\end{aligned}$$

that is, with probability at least $1 - 1/\mathrm{poly}(m)$,

$$\left\|\boldsymbol{V}^{(i)}\right\|_2 \leq \epsilon/4. \tag{63}$$

Now we finish the proof. By equality (61) and the definition of $\boldsymbol{U}^{(i)}$ (see (62)), we have

$$\begin{aligned}&\left\|\widehat{\boldsymbol{Y}}^{(i,0)} - \boldsymbol{Y}^{(i,0)}\right\|_2 = \left\|\sum_{j=0}^{i-1} \widehat{\boldsymbol{X}}^{(j)}\right\|_2 = \left\|\sum_{j=0}^{i-1}\left(\boldsymbol{U}^{(i)} + \mathbb{E}_{Z^{(i)}}\left[\widehat{\boldsymbol{X}}^{(j)}\right]\right)\right\|_2\\ =& \left\|\sum_{j=0}^{i-1} \boldsymbol{U}^{(i)} + \sum_{j=0}^{i-1}\mathbb{E}_{Z^{(i)}}\left[\widehat{\boldsymbol{X}}^{(j)}\right]\right\|_2 = \left\|\boldsymbol{V}^{(i)} + \sum_{j=0}^{i-1}\mathbb{E}_{Z^{(i)}}\left[\widehat{\boldsymbol{X}}^{(j)}\right]\right\|_2\\ \leq& \left\|\boldsymbol{V}^{(i)}\right\|_2 + \sum_{j=1}^{i-1}\left\|\mathbb{E}_{Z^{(i)}}\left[\widehat{\boldsymbol{X}}^{(j)}\right]\right\|_2. \qquad \text{(By triangle inequality)}\end{aligned}$$

By inequality (63) and Lemma 5.17, we have that with probability at least $1 - 1/\mathrm{poly}(m)$,

$$\left\|\widehat{\boldsymbol{Y}}^{(i,0)} - \boldsymbol{Y}^{(i,0)}\right\|_2 \leq \frac{\epsilon}{4} + \frac{C'\log m}{\alpha} \cdot 40\alpha\delta = \frac{\epsilon}{4} + \frac{40C'}{C\log m} \cdot \epsilon \leq \frac{\epsilon}{2}.$$

□

Now we prove Theorem 4. By algorithm ApproxSC, the returned graph H satisfies that $|E(H)| = O(|\mathcal{T}|\epsilon^{-2}\log^2 m)$. Therefore, it remains to prove that $\mathrm{SC}(H,\mathcal{T}) \approx_\epsilon \mathrm{SC}(G,\mathcal{T})$ with probability at least $1 - 1/\mathrm{poly}(m)$. Specifically, we bound the ℓ_2 norm of the difference matrix $\widehat{\boldsymbol{Y}}^{(i,0)} - \boldsymbol{P}$ by considering two parts: the martingale $\boldsymbol{Y}^{(i,t)}$ and the errors accumulated by $\widehat{\boldsymbol{Y}}^{(i,0)} - \widehat{\boldsymbol{Y}}^{(i-1,|Z^{(i-1)}|)}$, which correspond to Lemma 5.15 and Lemma 5.18 respectively.

Proof of Theorem 4. By Lemma 5.15 and Lemma 5.18, we have that

$$\left\|\widehat{\boldsymbol{Y}}^{(\tau,0)} - \boldsymbol{P}\right\|_2 \leq \left\|\widehat{\boldsymbol{Y}}^{(\tau,0)} - \boldsymbol{Y}^{(\tau,0)}\right\|_2 + \left\|\boldsymbol{Y}^{(\tau,0)} - \boldsymbol{P}\right\|_2 \leq \epsilon/2 + \epsilon/2 = \epsilon. \tag{64}$$

Note that

$$\widehat{\boldsymbol{Y}}^{(\tau,0)} = \boldsymbol{S}_0\boldsymbol{L}(H^{(\tau)})^\dagger\boldsymbol{S}_0^T = \boldsymbol{S}_0\boldsymbol{L}(H)^\dagger\boldsymbol{S}_0^T = \mathrm{SC}(G,\mathcal{T})^{1/2}\left(\boldsymbol{L}(H)^\dagger\right)_{[\mathcal{T},\mathcal{T}]}\mathrm{SC}(G,\mathcal{T})^{1/2},$$

and $\boldsymbol{P} = \mathrm{SC}(G,\mathcal{T})^{\dagger/2}\mathrm{SC}(G,\mathcal{T})\mathrm{SC}(G,\mathcal{T})^{\dagger/2}$, then inequality (64) tells us that

$$\mathrm{SC}\left(G,\mathcal{T}\right)^{1/2}\left(\boldsymbol{L}\left(H\right)^\dagger\right)_{[\mathcal{T},\mathcal{T}]}\mathrm{SC}\left(G,\mathcal{T}\right)^{1/2} \approx_\epsilon \mathrm{SC}\left(G,\mathcal{T}\right)^{\dagger/2}\mathrm{SC}\left(G,\mathcal{T}\right)\mathrm{SC}\left(G,\mathcal{T}\right)^{\dagger/2}. \tag{65}$$

Multiplying the both sides of the LHS and RHS of (65) by $\mathbf{SC}(G,\mathcal{T})^{\dagger/2}$ gives

$$\boldsymbol{P}\left(\boldsymbol{L}(H)^{\dagger}\right)_{[\mathcal{T},\mathcal{T}]}\boldsymbol{P} \approx_{\epsilon} \mathbf{SC}(G,\mathcal{T})^{\dagger}.$$

By Lemma 2.3, it holds that $\boldsymbol{P}\left(\boldsymbol{L}(H)^{\dagger}\right)_{[\mathcal{T},\mathcal{T}]}\boldsymbol{P} = \mathbf{SC}(H,\mathcal{T})^{\dagger}$, therefore,

$$\mathbf{SC}(H,\mathcal{T})^{\dagger} \approx_{\epsilon} \mathbf{SC}(G,\mathcal{T})^{\dagger},$$

that is,

$$\mathbf{SC}(H,\mathcal{T}) \approx_{\epsilon} \mathbf{SC}(G,\mathcal{T}).$$

□

Finally, we prove Theorem 3. Since the algorithm ApproxSC only applies deletions and contractions on the input graph, it follows that the resulting sparsifier is a minor. The correctness and the bound on the number of edges follow from Theorem 4. Therefore, it remains to bound the computation cost.

Proof of Theorem 3. By Lemma 5.10, the number of iterations in the main while loop of the algorithm ApproxSC (Line 4 of algorithm 4) is $O(\alpha^{-1}\log m) = O(\epsilon^{-1}\log^5 n)$. By Lemma 5.2, 5.4, 5.5, the communication cost of each iteration is dominated by line 6 and line 7, which require solving $O(\delta^{-2}\log n) = O(\epsilon^{-2}\log^5 n)$ Laplacian linear systems, and $O(\rho\epsilon^{-2}\sqrt{\overline{n}}\log\overline{n}\log^5 n + D\log n)$ rounds of communication in $\overline{G}$.

Therefore, the total number of required Laplacian solvers is

$$O\left(\epsilon^{-1}\log^5 n\cdot\epsilon^{-2}\log^5 n\right) = O\left(\epsilon^{-3}\log^{10} n\right).$$

The total overhead cost of communication in $\overline{G}$ can be bounded in the same way, that is,

$$O\left(\epsilon^{-1}\log^5 n\left(\rho\epsilon^{-2}\sqrt{\overline{n}}\log\overline{n}\log^5 n + D\log n\right)\right) = O\left(\rho\epsilon^{-3}\sqrt{\overline{n}}\log\overline{n}\log^{10} n + D\epsilon^{-1}\log^6 n\right).$$

□

6 Vertex and Edge Reductions

Here we show our reductions via tree and elimination based preconditioners in Section 6.1 and Section 6.2 respectively. This will prove Lemmas 4.9 and 4.10.

6.1 Ultra-Sparsifier

We prove the high error reduction routine as stated in Lemma 4.9

Lemma 4.9. *There is a routine* UltraSparsify(G,k) *in the* CONGEST *model that given a graph G with n vertices and m edges, that ρ-minor distributes into the communication network $\overline{G}$, which has $\overline{n}$ vertices, $\overline{m}$ edges, and diameter D, along with a parameter k, produces in $O(n^{o(1)}(\rho\sqrt{\overline{n}}+D))$ rounds a graph H such that:*

1. *H is a subgraph of G,*

2. *H has at most $n - 1 + m2^{O(\sqrt{\log n \log\log n})}/k$ edges.*

3. *$\boldsymbol{L}(G) \preceq \boldsymbol{L}(H) \preceq k\boldsymbol{L}(G)$.*

Furthermore, the algorithm also gives $\widehat{G}, \boldsymbol{Z}_1, \boldsymbol{Z}_2, C$ such that

1. *$\widehat{G}$ 1-minor distributes into H such that $\widehat{G} = \mathbf{SC}(H, C)$ with $|C| = m2^{O(\sqrt{\log n \log\log n})}/k$.*

2. *There are operators $\boldsymbol{Z}_1$ and $\boldsymbol{Z}_2$ evaluable with $O(\rho\sqrt{\overline{n}}\log\overline{n}+D)$ rounds of* CONGEST *communication on $\overline{G}$ such that:*

$$\boldsymbol{L}(H)^{\dagger} = \boldsymbol{Z}_1^{\top} \begin{bmatrix} \boldsymbol{Z}_2 & 0 \\ 0 & \boldsymbol{L}\left(\widehat{G}\right)^{\dagger} \end{bmatrix} \boldsymbol{Z}_1$$

We follow the construction from [KMP10], which samples off-tree edges with any upper bound on their stretches. To find the tree, we utilize the distributed version of the Alon-Karp-Peleg-West (AKPW) low stretch spanning tree, due to Ghaffari, Karrenbauer, Kuhn, Lenzen, and Patt-Shamir [GKK+15]. They work with a definition of distributed N-node cluster graphs that was the basis of our definition of distributed ρ-minor. We start by restating this definition, and describe how we simulate it when G is itself embedded.

Definition 6.1. A distributed N-node cluster graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{L}, \mathcal{T}, \psi)$ is defined by a set of N clusters $\mathcal{V} = \{S_1, \ldots, S_N\}$ partitioning the vertex set V, a set of weighted multiedges, a set of cluster leaders $\mathcal{L}$, a set of cluster trees $\mathcal{T}$, as well as a function ψ that maps the edges $\mathcal{E}$ of the cluster graph to edges in E. Formally, the tuple $(\mathcal{V}, \mathcal{E}, \mathcal{L}, \mathcal{T}, \psi)$ has to satisfy the following conditions.

1. The clusters $\mathcal{V} = (S_1, \ldots, S_N)$ form a partition of the set of vertices V.

2. For each cluster S_i, $|S_i \cap \mathcal{L}| = 1$. Hence, each cluster has exactly one cluster leader $\ell_i \in \mathcal{L} \cap S_i$. The ID of the node ℓ_i also serves as the ID of the cluster S_i and for the purpose of distributed computations, we assume that all nodes $v \in S_i$ know the cluster ID and the size $n_i := |S_i|$ of their cluster S_i.

3. Each cluster tree $T_i = (S_i, E_i)$ is a rooted spanning tree of the subgraph $G[S_i]$ of G induced by S_i. The root of T_i is the cluster leader $\ell_i \in S_i \cap \mathcal{L}$.

4. The function $\psi : \mathcal{E} \to E$ maps each edge of $\mathcal{E}$ to an (actual) edge of E connecting the clusters.

As a consequence of Lemma 4.3, we get that shortest paths can be ran on distributed N-node cluster graphs of G

Lemma 6.2. *Let $G = (V, E)$ be a graph with n vertices and m edges that ρ-embed into the communication network $\overline{G} = (\overline{V}, \overline{E})$, and $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{L}, \mathcal{T}, \psi)$ be a distributed cluster graph for G.*

Then we have the following algorithms:

1. *For each cluster S_i, the cluster leader ℓ_i broadcasts $O(\log\overline{n})$ bit message s_i to each vertex of S_i in $O(\rho\overline{n}^{1/2}\log\overline{n} + D)$ rounds.*

2. *Assume every vertex $v \in S_i$ for each $S_i \in \mathcal{V}$, the corresponding vertex $v' \in \overline{V}$ holds a value $f(v)$. Then computing $\min_{v \in S_i}\{f(v)\}$ at node ℓ_i for each $S_i \in \mathcal{V}$ needs $O(\rho\overline{n}^{1/2}\log\overline{n} + D)$ rounds if the tree T_i with root ℓ_i is known.*

Proof. The definition of distributed N-node cluster graphs implies that $\mathcal{G}$ 1-minor distributes over G. Lemma 4.6 then gives that $\mathcal{G}$ ρ-minor distributes over $\overline{G}$, and this distributed mapping can be obtained using $O(\rho\overline{n}^{1/2}\log\overline{n}+D)$ rounds of computations. The broadcast, and the aggregation of minimums then follow from Lemma 4.3. □

This in turn implies that the SplitGraph algorithm in [GKK+15] can be simulated on a graph that's ρ-minor distributed into $\overline{G}$ in $O(n^{o(1)}(\rho\overline{n}^{1/2}\log\overline{n}+D))$ rounds. Putting it together gives our variant of the AKPW low stretch spanning tree algorithm, with the main difference being that it's ran on a ρ-minor distributed over our overall communication network.

Lemma 6.3. *Let $G=(V,E)$ be a graph with n vertices and m edges that ρ-embeds into the communication network $\overline{G}=(\overline{V},\overline{E})$, and $\mathcal{G}=(\mathcal{V},\mathcal{E},\mathcal{L},\mathcal{T},psi)$ be a distributed cluster graph for G.*

It takes $O(n^{o(1)}(\rho\overline{n}^{1/2}\log\overline{n}+D))$ rounds to construct a spanning tree T of G, along with stretch upper bounds that sum to

$$m\cdot 2^{O(\sqrt{\log n\log\log n})}.$$

These upper bounds are sufficient for sampling the edges by stretch. The following was shown in [KMP10], or Theorem 2.2.4 in [Pen13].

Lemma 6.4. *Given a graph G, a tree T, upper bounds on stretches of edges of T w.r.t. G that sum to α, along with a parameter k, there is an independent sampling / rescaling distribution computable locally from the stretch upper bounds that gives a graph H such that with high probability*

1. $\boldsymbol{L}(G)\preceq\boldsymbol{L}(H)\preceq k\boldsymbol{L}(H)$
2. *H contains (rescaled) T, plus $O(\alpha\log n/k)$ edges.*

We then need to contract the tree so that its size becomes similar to the number of off-tree edges.

Lemma 6.5. *Let $H=(V,E)$ be a graph with n vertices and m edges that ρ-embed into the communication network $\overline{G}=(\overline{V},\overline{E})$.*

Let T be a spanning tree of H and $W=E-T$ be the set of off-tree edges of H with respect to T. There is an algorithm to compute a graph $\widehat{G}$ that's 1-embeddable into H satisfying the following conditions in $O(\rho\overline{n}^{1/2}\log\overline{n}+D)$ rounds:

1. *$\widehat{G}$ contains $O(|W|)$ vertices and edges.*
2. *There are operators $\boldsymbol{Z}_1$ and $\boldsymbol{Z}_2$ that can be evaluated in $O(\rho\overline{n}^{1/2}\log\overline{n}+D)$ rounds with*

$$\boldsymbol{L}(H)^{\dagger}=\boldsymbol{Z}_1^{\top}\begin{bmatrix}\boldsymbol{Z}_2 & 0\\ 0 & \boldsymbol{L}\left(\widehat{G}\right)^{\dagger}\end{bmatrix}\boldsymbol{Z}_1$$

Proof. We use the parallel elimination procedure from Section 6.3 of [BGK+14], specifically Lemma 26. At a high level, it eliminates degree 1 and 2 vertices by random sampling a subset of vertices which have degree 1 or 2, computing an independent set, and eliminating them. The algorithm requires $O(\log n)$ rounds in PRAM, and therefore can be implemented in $O(\log n(\rho\sqrt{\overline{n}}\log\overline{n}+D))$ rounds in the CONGEST model by Lemma 4.3. The operators $\boldsymbol{Z}_1,\boldsymbol{Z}_2$ are computed as in Lemma 26 of [BGK+14]. □

We can combine these pieces to prove the main ultrasparsification claim.

Proof of Lemma 4.9. The algorithm to prove Lemma 4.9 is as follows.

1. Compute a low-stretch tree using Lemma 6.3.
2. Sample edges using Lemma 6.4 with $\alpha = m \cdot 2^{O(\sqrt{\log n \log\log n})}$.
3. Compute the operators Z_1, Z_2 using Lemma 6.5.

We verify the conditions of Lemma 4.9. The approximation guarantees and number of off-tree edges are given by Lemma 6.4

After eliminating degree 1 and degree 2 vertices, the resulting graph has size $O(\alpha \log n)$ by Lemma 6.5 Part 1, and Z_1 and Z_2 are computed by Lemma 6.5.

The round complexity in the CONGEST model follows by summing the round complexities in Lemmas 6.3, 6.4, 6.5. □

6.2 Elimination / Sparsified Cholesky

The main goal of this section is to prove Lemma 4.10, which allows the elimination of large subsets of vertices under small error.

Lemma 4.10. *There is a routine* Eliminate(G, d, ϵ) *in the* CONGEST *model that given a graph G that ρ-minor distributes into a communication network $\overline{G}$, along with step count d and error ϵ, produces in*

$$O((\epsilon^{-6}\log^{14} n)^d(\rho\sqrt{\overline{n}}\log\overline{n} + D))$$

rounds a subset $\mathcal{T}$ and access to operators Z_1 and Z_2 such that

1. $|\mathcal{T}| \leq (\frac{49}{50})^d |V(G)|$.
2. *The cost of applying Z_1, $Z_1^\top$ and Z_2 to vectors is $O((\epsilon^{-6}\log^{14}\overline{n})^d(\rho\sqrt{\overline{n}}\log\overline{n} + D))$ rounds of communication on $\overline{G}$.*
3. $\boldsymbol{L}(G)^\dagger$ *is $(1\pm\epsilon)^d$-approximated by a composed operator built from Z_1, Z_2, and the inverse of the of the Schur complement of $\boldsymbol{L}(G)$ onto C, $\mathbf{SC}(\boldsymbol{L}(G), C)$:*

$$(1-\epsilon)^d \boldsymbol{L}(G)^\dagger \preceq Z_1^\top \begin{bmatrix} Z_2 & 0 \\ 0 & \mathbf{SC}(\boldsymbol{L}(G), C)^\dagger \end{bmatrix} Z_1 \preceq (1+\epsilon)^d \boldsymbol{L}(G)^\dagger$$

To prove the above the above lemma, we present a distributed implementation of the *Schur Complement Chain* (SCC) construction due to Kyng, Lee, Peng, Sachdeva, and Spielman [KLP+16]. The key components to this construction are (i) an algorithm that finds a large near-independent set F, and approximates the inverse of the matrix restricted to entries in F and (ii) a procedure for spectrally approximating the Schur complement with respect to $C = V \setminus F$. We next discuss how to implement these components in the CONGEST model.

Finding large α-DD sets. When doing Gaussian elimination, the goal is to find a large subset of vertices F such that we can approximate the inverse of $\boldsymbol{L}_{[F,F]}$ by an operator $\boldsymbol{Z}$ that can be constructed efficiently. Ideally, F forms an independent set. Unfortunately, we are not able to find a large independent set but we can instead find a large, almost-independent set, as made precise in the following definition.

Definition 6.6 (α-DD)**.** A matrix $\boldsymbol{M}$ is α-diagonally dominant (α-DD) if

$$\forall i, \quad \boldsymbol{M}_{i,i} \geq (1+\alpha) \sum_{j:j\neq i} \boldsymbol{M}_{i,j}.$$

An index set F is α-DD if $\boldsymbol{M}_{[F,F]}$ is α-DD.

The algorithm due to [KLP+16] for finding α-DD sets in a Laplacian proceeds as follows: (i) pick a random subsets of vertices and (ii) and discard all those that do not satisfy the condition in Definition 6.6. The pseudocode for computing such sets is given in Algorithm 6. In the CONGEST model, the way the set is "stored" is that each vertex remembers whether it is in the set.

Algorithm 6: Find an α-DD subset F of $\boldsymbol{L}$

1 **procedure** DDSubset($\boldsymbol{L}, \alpha$)
2 Sample each index of $\{1,\ldots,n\}$ independently with probability $\frac{1}{4(1+\alpha)}$ and let F' be the resulting set of sampled indices.
3 Set

$$F = \left\{ i \in F' : |\boldsymbol{L}_{i,i}| \geq (1+\alpha) \sum_{j\in F', j\neq i} |\boldsymbol{L}_{i,j}| \right\}.$$

4 **if** $|F| < \frac{n}{8(1+\alpha)}$ **then**
5 Goto Step 1.
6 **return** F.

We have the following lemma.

Lemma 6.7. *Let $G = (V,E)$ be a graph that ρ-minor distributes into the communication network $\overline{G} = (\overline{V},\overline{E})$. Let $\boldsymbol{L}$ be the Laplacian matrix associated with G and let $\alpha \geq 0$ be a parameter. Then* DDSubset($\boldsymbol{L},\alpha$) *computes an α-DD subset F of $\boldsymbol{L}$ of size $n/(8(1+\alpha))$ in $O(\rho\sqrt{\overline{n}}\log\overline{n} + D)$ rounds.*

Proof. In [LPS15, Lemma 5.2] (and more generally in [KLP+16]), it is shown that Algorithm 6 computes an α-DD subset F of size $n/(8(1+\alpha))$. To bound the round complexity of the algorithm, consider the following distributed implementation:

1. Include each index of $\{1,\ldots,n\}$ in F' with probability $\frac{1}{4(1+\alpha)}$.
2. Each node corresponding to $i \in F'$ sums up the values $|\boldsymbol{L}_{i,j}|$ of the indices j corresponding to its neighbors in G, and then decides whether $|\boldsymbol{L}_{i,i}| \geq (1+\alpha)\sum_{j\in F', j\neq i} |\boldsymbol{L}_{i,j}|$ and if so declares itself as belonging to F.
3. The size of F is computed by an (arbitrarily decided) leader vertex, which aggregates the sum of the following values over all vertices v in G: 1 if v is in F and 0 otherwise.

4. The leader checks whether $|F| < n/(8(1+\alpha))$. If the latter holds, then the leaders informs all the vertices in G to repeat the previous steps. Otherwise, the algorithm terminates.

In the CONGEST model, Step 1 requires no communication between the nodes: each root vertex of supervertices does the sampling independently. In Step 2, each node computes an aggregate of values stored by its neighbors in G, which by Lemma 4.3 takes $O(\rho\sqrt{\overline{n}}\log\overline{n} + D)$ rounds. It is well-known that Steps 3 and 4 can be carried out in $O(D)$ rounds by routing the messages via a BFS tree rooted at the leader. Together with the fact that Algorithm 6 terminates in at most 2 iterations in expectation (see [LPS15, Lemma 5.2]), it follows that the distributed implementation takes $O(\rho\sqrt{\overline{n}}\log\overline{n} + D)$ rounds in expectation. □

Jacobi Iteration on α-DD matrices. Using an α-DD set F, we will construct an operator $\boldsymbol{Z}$ that approximates $\boldsymbol{L}_{[F,F]}^{-1}$ and can be applied efficiently to any vector. An important observation is that we can write $\boldsymbol{L}_{[F,F]} = \boldsymbol{X}_{[F,F]} + \boldsymbol{Y}_{[F,F]}$, where $\boldsymbol{X}_{[F,F]}$ is a diagonal matrix and $\boldsymbol{Y}_{[F,F]}$ is a Laplacian matrix. We have the following lemma.

Lemma 6.8. *Let $G = (V, E)$ be a graph that ρ-minor distributes into the communication network $\overline{G} = (\overline{V}, \overline{E})$. Let $\boldsymbol{L}$ be the Laplacian matrix associated with G and let F be a subset of of V such that $\boldsymbol{L}_{[F,F]}$ is α-DD for some $\alpha \geq 4$. Then* Jacobi$(\boldsymbol{L}_{[F,F]}, \cdot, \epsilon)$ *gives a linear operator $\boldsymbol{Z}$ that over vectors given on the root vertices of the supervertices such that for any vector $\vec{b}$ given by storing $\vec{b}_{v^G}$ on $V_{map}^{G\to\overline{G}}(\cdot)$, returns in $O((\rho\sqrt{\overline{n}} + D)\log(1/\epsilon))$ rounds $\boldsymbol{Z}\vec{b}$ stored on the same vertices, for some matrix $\boldsymbol{Z}$ such that*

$$\boldsymbol{L}_{[F,F]} \preceq \boldsymbol{Z}^{(-1)} \preceq \boldsymbol{L}_{[F,F]} + \epsilon \cdot \mathrm{SC}(\boldsymbol{L}, F).$$

Note that the matrix $\boldsymbol{Z}$ is only used in the analysis, and is never explicitly constructed by the algorithm. We first give the pseudocode of this algorithm in the centralized setting, and then show its distributed implementation.

Algorithm 7: Solve $\boldsymbol{L}_{[F,F]} \cdot \vec{x}_F = \vec{b}_F$ up to ϵ accuracy

```
procedure Jacobi(L_[F,F], b_F, ε)
    Set L_[F,F] = X_[F,F] + Y_[F,F] such that X_[F,F] is diagonal and Y_[F,F] is a Laplacian.
    Set k to be an odd integer that is greater than log(3/ε).
    Set x_F^(0) = X_[F,F]^{-1} b_F.
    for i = 1, ..., k do
        Set x_F^(i) = -X_[F,F]^{-1} Y_[F,F] x_F^(i-1) + X_[F,F]^{-1} b_F.
    return x_F^(k).
```

To measure the quality of the operator produced by Jacobi procedure, we observe that k iterations produce the operator

$$\boldsymbol{Z}^{(k)} := \sum_{i=0}^{k} \boldsymbol{X}_{[F,F]}^{-1} \left(-\boldsymbol{Y}_{[F,F]}\boldsymbol{X}_{[F,F]}^{-1}\right)^{i} \tag{66}$$

by induction. Concretely, suppose we have

$$\vec{x}_F^{(k-1)} = \sum_{i=0}^{k-1} \boldsymbol{X}_{[F,F]}^{-1} \left(-\boldsymbol{Y}_{[F,F]} \boldsymbol{X}_{[F,F]}^{-1}\right)^i \vec{b}_F,$$

then substituting this into the step in Line 6 gives

$$\begin{aligned}
\vec{x}_F^{(k)} &= -\boldsymbol{X}_{[F,F]}^{-1} \boldsymbol{Y}_{[F,F]} \vec{x}_F^{(k-1)} + \boldsymbol{X}_{[F,F]}^{-1} \vec{b}_F \\
&= \boldsymbol{X}_{[F,F]}^{-1} \vec{b}_F + \left(-\boldsymbol{X}_{[F,F]}^{-1} \boldsymbol{Y}_{[F,F]}\right) \sum_{i=0}^{k-1} \boldsymbol{X}_{[F,F]}^{-1} \left(-\boldsymbol{Y}_{[F,F]} \boldsymbol{X}_{[F,F]}^{-1}\right)^i \vec{b}_F \\
&= \boldsymbol{X}_{[F,F]}^{-1} \vec{b}_F + \sum_{i=1}^{k} \boldsymbol{X}_{[F,F]}^{-1} \left(-\boldsymbol{Y}_{[F,F]} \boldsymbol{X}_{[F,F]}^{-1}\right)^i \vec{b}_F = \sum_{i=0}^{k} \boldsymbol{X}_{[F,F]}^{-1} \left(-\boldsymbol{Y}_{[F,F]} \boldsymbol{X}_{[F,F]}^{-1}\right)^i \vec{b}_F.
\end{aligned}$$

We next review two lemmas from [KLP+16] that help us prove the approximation accuracy of the Jacobi iteration on $\boldsymbol{L}_{[F,F]}$. The first shows that α-DD matrices admit good diagonal preconditioners. The second gives a way to bound the error produced by Jacobi.

Lemma 6.9 ([KLP+16], Lemma 3.6.)**.** *Let $\boldsymbol{L}_{[F,F]}$ be an α-DD matrix which can be written in the form $\boldsymbol{X}_{[F,F]} + \boldsymbol{Y}_{[F,F]}$ where $\boldsymbol{X}_{[F,F]}$ is diagonal and $\boldsymbol{Y}_{[F,F]}$ is a Laplacian. Then $\frac{\alpha}{2}\boldsymbol{Y} \preceq \boldsymbol{X}$.*

Lemma 6.10 ([KLP+16], Lemma E.1.)**.** *Let $\boldsymbol{L}_{[F,F]}$ be an α-DD matrix with $\boldsymbol{L}_{[F,F]} = \boldsymbol{X}_{[F,F]} + \boldsymbol{Y}_{[F,F]}$ where $0 \preceq \boldsymbol{Y}_{[F,F]} \preceq \beta \boldsymbol{X}$ for some $0 < \beta < 1$. Then, for any odd k and $\boldsymbol{Z}^{(k)}$ as defined in Eq. (66), we have*

$$\boldsymbol{X}_{[F,F]} + \boldsymbol{Y}_{[F,F]} \preceq (\boldsymbol{Z}^{(k)})^{-1} \preceq \boldsymbol{X}_{[F,F]} + (1+\delta)\boldsymbol{Y}_{[F,F]},$$

where

$$\delta = \beta^k \frac{1+\beta}{1-\beta^{k+1}}.$$

Proof of Lemma 6.8. Let $\boldsymbol{Y}_{[F,F]}$ be the matrix generated when calling Jacobi with $\boldsymbol{L}_{[F,F]}$. Since $\boldsymbol{L}_{[F,F]}$ is an α-DD matrix, by extending $\boldsymbol{Y}_{[F,F]}$ with zero entries, we have $\boldsymbol{Y} \preceq \boldsymbol{L}$. This in turn implies that $\boldsymbol{Y}_{[F,F]} = \mathrm{SC}(\boldsymbol{Y}, F) \preceq \mathrm{SC}(\boldsymbol{L}, F)$.

Lemma 6.9 gives that $\frac{\alpha}{2}\boldsymbol{Y} \preceq \boldsymbol{X}$. As $\alpha \geq 4$, we can invoke Lemma 6.10 with $\beta = 1/2$, which in gives that $(1+\beta)/(1-\beta^{k+1}) \leq 3$. Therefore, our choice of $k = \log(3/\epsilon)$ gives the desired error guarantee. To bound the round complexity of the algorithm, consider the following distributed implementation of Jacobi:

1. Store the values $\vec{b}_F(u)$, $\boldsymbol{X}_{[F,F]}(u,u)$, $\boldsymbol{Y}_{[F,F]}(u,u)$ at $V_{map}^{G \to \overline{G}}(u)$. Store the off-diagonal entries of $\boldsymbol{Y}_{[F,F]}$, together with their weights, in the endpoints of mapped edges $E_{map}^{G \to \overline{G}}(e)$: this is possible because $\boldsymbol{Y}_{[F,F]}$ is a Laplacian.

2. Set $k = \log(1/\epsilon)$ and $\vec{x}_F^{(0)}(u) = \boldsymbol{X}_{[F,F]}^{-1}(u,u) \cdot \vec{b}_F(u)$ for each $u \in F$.

3. For $i = 1, \ldots, k$ do

(a) For each $u \in F$ set

$$\vec{x}_F^{(i)}(u) \leftarrow \boldsymbol{X}_{[F,F]}^{-1}(u,u) \cdot \vec{b}_F(u) - \boldsymbol{X}_{[F,F]}^{-1}(u,u) \sum_{v \in F : \boldsymbol{Y}_{[F,F]}(u,v) \neq 0} \boldsymbol{Y}_{[F,F]}(v,u) \vec{x}^{(i-1)}(v)$$

using the matrix-vector multiplication primitive from Corollary 4.4, with results stored on all root vertices, $V_{map}^{G \to \overline{G}}(u)$.

4. Every vertex $u \in F$ returns $\vec{x}_F^{(k)}(u)$.

The complexity of the algorithm is dominated by the number of rounds to implement Step 3, which is given by Corollary 4.4. As there are $k = \log(1/\epsilon)$ iterations, we have that Step 3 requires $O(\log(1/\epsilon))$ rounds in G.

By Lemma 4.3 and using the fact that weights of the network, and hence the solution vector's magnitudes, are polynomially bounded, we can simulate the algorithm in the original communication network $\overline{G}$ in $O((\rho\sqrt{\overline{n}}\log\overline{n} + D)\log(3/\epsilon))$ rounds. □

Approximating Schur complements using low congestion random walks. We next show that α-DD sets are useful when approximating Schur complements. A key ingredient to our construction is the following combinatorial view of Schur complements.

It is well known that $\mathbf{SC}(\boldsymbol{L}, \mathcal{T})$ is a Laplacian matrix of a graph on vertices in $\mathcal{T} = V \setminus F$. For our purposes, it will be useful interpret $\mathbf{SC}(\boldsymbol{L}, \mathcal{T})$ in terms of random walks. To this end, given a walk $W = u_0, \ldots, u_l$ of length ℓ in G with a subset of vertices $\mathcal{T}$, we say that W is a *terminal-free* walk if $u_0, u_\ell \in \mathcal{T}$ and $u_1, \ldots, u_{\ell-1} \notin \mathcal{T}$.

Lemma 6.11. *For any undirected, weighted graph G and any subset of vertices $\mathcal{T}$, the Schur Complement $\mathbf{SC}(G, \mathcal{T})$ is given as a union over all multi-edges corresponding to terminal-free walks $u_0, \ldots, u_\ell$ with weight*

$$\frac{\prod_{0 \le i < k} \vec{w}_{u_i u_{i+1}}}{\prod_{0 \le i \le k} \sum_{u_i v \in E(G)} \vec{w}_{u_i v}}$$

The theorem below allows us to efficiently sample from this distribution of walks while paying a small cost in the approximation quality.

Lemma 6.12 (Theorem 3.1 in [DGGP19])**.** *Let $G = (V, E)$ be an undirected, weighted graph with a subset of vertices $\mathcal{T}$. Let $\epsilon \in (0, 1)$ be an error parameter and $\mu = \Theta(\epsilon^{-2} \log n)$ be some parameter related to the concentration of sampling. Let H be an initially empty graph, and for every edge $e = (u, v) \in G$, repeat μ times the following procedure, where a random step from a vertex is taken proportional to the edge weights of its adjacent edges.*

1. *Simulate a random walk starting from u until it hits $\mathcal{T}$ at vertex t_1.*
2. *Simulate a random walk starting from v until it hits $\mathcal{T}$ at vertex t_2.*
3. *Let ℓ be the total length of this combined walk (including edge e). Add the edge (t_1, t_2) to H with weight*

$$\frac{1}{\mu \sum_{i=0}^{\ell-1} (1/\vec{w}_{u_i, u_{i+1}})}.$$

The resulting graph H satisfies $\boldsymbol{L}(H) \approx_\epsilon \mathbf{SC}(\boldsymbol{L}(G), \mathcal{T})$ with high probability.

The main idea to make use of the above theorem is to compute an α-DD set F using Lemma 6.7 as this ensures that the random walks in the graph are short in expectation. However, since we are dealing with weighted graphs, there might be scenarios where the expected congestion of an edge is prohibitively large, which makes it difficult to recursively repeat the algorithm. To alleviate this, we add new vertices to the terminals, whenever they have too much congestion. Note that because G is distributed over $\overline{G}$ as a minor, we can only accommodate small vertex congestion due to the need for each root node to inform the entire supervertex. As a result, our resulting congestion depends on the average degree, and we resolve this via calling sparsification (Corollary 4.5) at each step.

Let W be the family of walks generated in Lemma 6.12. For $e \in E$, let $\text{cong}_W(e)$ denote the number of walks from W that use the edge e. Algorith 8 below computes W. Note that it can also be run implicitly to generate the congestion on every edge without exceeding the communication limit on any edges: we simply pass around the congestion on every edge.

Algorithm 8: Generate random walks from each edge until they hit terminals

```
1 procedure RandomWalk(G, 𝒯, μ)
2     Set W ← ∅
3     for e = (u, v) ∈ E(G) in parallel do
4         for i = 1, ..., μ in parallel do
5             Generate a random walk W(u, i) from u until it hits 𝒯 at vertex t_1.
6             Generate a random walk W(v, i) from v until it hits 𝒯 at vertex t_2.
7             Set W(e, i) = W(u, i) ∪ (u, v) ∪ W(v, i) and W = W ∪ {W(e, i)}.
8     return W.
```

Implementation of random walks. We implement the random walks in lines 5 and 6 of RandomWalk as in Algorithm 8 as follows. When a non-terminal vertex gets a random walk edge into it, first we do a sum aggregation so that the leader of the corresponding cluster / super-vertex knows how many out-edges to compute. Below, we will ensure that all non-terminal vertices have low congestion as intermediate vertices of random walks, so we will focus on sampling a single out-edge. To do this, the vertex aggregates the sum of weights of out-edges, which we will call W_{total}. Now, the leader samples a random real number $r \in [0, W_{total}]$. Finally, the leader vertex does a binary search on the label of the out-edge and aggregates sums to figure out which out-edge corresponds to the sample r. All these steps can be done by Lemma 4.3.

We now move on to giving the distributed random walk based algorithm that approximates Schur complements. Psuedocode of this routine is in Algorithm 9.

Here, we discuss subtleties in the distributed implementation of Algorithm RandomWalk and RandomWalkSchur (Algorithms 8 and 9) in the CONGEST model.

Lemma 6.13. *Let $G = (V, E)$ be a graph that ρ-minor distributes into the communication network $\overline{G} = (\overline{V}, \overline{E})$. Let F be an α-DD set, $\mathcal{T} = V \setminus F$, $\epsilon \in (0, 1)$ be an error parameter and $\gamma \geq 1$ be a congestion parameter. Then the procedure* RandWalkSchur$(G, \mathcal{T}, \epsilon, \gamma, \alpha)$ *outputs in $O(\alpha\gamma\epsilon^{-2} \log n(\rho\sqrt{\overline{n}} \log \overline{n} + D))$*

Algorithm 9: Distributed Approximate Schur complements using random walks

procedure RandWalkSchur($G, \mathcal{T}, \epsilon, \gamma, \alpha$)
  Initialize $\widehat{\mathcal{T}} \leftarrow \mathcal{T}$
  Set $H \leftarrow \emptyset$ and $\mu \leftarrow O(\epsilon^{-2} \log n)$
  Implicitly compute the expected congestion of $W = \text{RandomWalk}(G, \mathcal{T}, \mu)$ by propagating the expected congestion on vertices and edges evenly to neighbors for $O(\alpha \log n)$ steps.
  for *all vertices u with $\mathbb{E}[\text{cong}_W(u)] > \gamma$, in parallel* **do**
    Add u to $\widehat{\mathcal{T}}$, $\widehat{\mathcal{T}} \leftarrow \widehat{\mathcal{T}} \cup \{u\}$.
  Set $W \leftarrow \text{RandomWalk}(G, \widehat{\mathcal{T}}, \mu)$ (with walks explicitly generated).
  Initialize the minor distribution of H into $\overline{G}$ by associating each terminal t with all vertices involved in all random walks that ended at t, and building $T^{H \to \overline{G}}(t)$ to be a spanning tree of all edges used, plus $T^{G \to \overline{G}}(u)$ of all vertices on these walks.
  for $W(e, i) \in W$ **do**
    Let (t_1, t_2) be the endpoints of the walk $W(e, i)$.
    Let ℓ be the length of $W(e, i)$.
    Set $H = H \cup (t_1, t_2, w(t_1, t_2))$ with $\vec{w}(t_1, t_2) := 1/(\mu \sum_{i=0}^{\ell-1} (1/\vec{w}_{u_i, u_{i+1}}))$.
  return $H, \widehat{\mathcal{T}}$.

rounds a graph H along with its $\alpha\gamma \log n$-minor distribution into $\overline{G}$ such that with high probability,

$$\boldsymbol{L}(H) \approx_\epsilon \mathbf{SC}(\boldsymbol{L}(G), \widehat{\mathcal{T}})$$

for some (slightly larger) superset $\widehat{\mathcal{T}} \supseteq \mathcal{T}$ of size at most $n - |F| + O(\alpha m \epsilon^{-2} \log n/\gamma)$.

Proof. The spectral guarantee $\boldsymbol{L}(H) \approx_\epsilon \mathbf{SC}(\boldsymbol{L}(G), \widehat{\mathcal{T}})$ follows directly from Lemma 6.12. To bound the size of $\mathcal{T}$, first note that by definition $\mathcal{T} = V \setminus F$ and thus $|\mathcal{T}| = n - |F|$. Next, as F is an α-DD set, the expected length of a random walk that starts at an endpoint of any edge in G and hits a vertex in $\mathcal{T}$ is $O(\alpha)$. Our algorithm simulates $O(m\epsilon^{-2} \log n)$ random walks for each edge, thus the total congestion generated by these walks is $O(\alpha m \epsilon^{-2} \log n)$.

The latter gives that there can be at most $O(\alpha m \epsilon^{-2} \log n/\gamma)$ vertices whose expected congestion is larger than γ, and RandWalkSchur adds these vertices to the set $\mathcal{T}$. It follows that $|\widehat{\mathcal{T}}| \leq n - |F| + O(\alpha m \epsilon^{-2} \log n/\gamma)$. For each vertex u, the congestion incurred by other edges are independent random variables bounded by the length of the walks, which is $O(\log n)$. So by a Chernoff bound, the congestion of all edges with expected congestion less than $\gamma > O(\log^2 n)$ is at most $O(\gamma)$ with high probability. So after line 4 of Algorithm 9 adds all vertices with high expected congestion into the terminals (to form $\widehat{\mathcal{T}}$), all subsequent vertices in $V \setminus \widehat{\mathcal{T}}$ have vertex congestion at most $O(\gamma)$ in the second random walk in line 7 with high probability.

We next study the round complexity. To this end, recall that the expected length of each walk in W is $O(\alpha \log n)$ with high probability.

When we are only passing the congestion of a vertex to neighbors, that is, running the walks implicitly, each round can be executed in one round of message passing as described in Lemma 4.3. As we execute $\mu = O(\epsilon^{-2} \log n)$ rounds for each edge, we have that the round complexity of the congestion estimation part of RandomWalk is $O(\alpha\rho(\sqrt{n} + D)\epsilon^{-2} \log^2 n)$.

For the explicit generation, we aggregate, for all non-terminal vertices of G, the walks that reach them, to the root node of the corresponding super vertex in $\overline{G}$. Then we broadcast these walks outward, we simulate the choice of a random edge by total weights of edges in subtrees (which we compute via Lemma 4.2). As the node congestions are at most 2γ, Lemma 4.3 lets us perform these propagations in $\overline{G}$ in $O(\alpha\gamma\epsilon^{-2}\log n(\rho\sqrt{\overline{n}}\log\overline{n}+D))$ rounds.

Finally, to create the minor distribution of H, the graph with the new random walk edges, into $\overline{G}$, we extend the terminals to include the supervertices of all intermediate (non-terminal) vertices. As the non-terminals have at most $O(\gamma)$ walks through them, the resulting mapping is still a $\rho\gamma$-minor distribution into $\overline{G}$. □

We remark that in this scheme, the end points of the new edges in H cannot actually know these edges in a centralized manner (e.g. aggregate them at root vertices). Instead, such walks are only passed to the vertices in $\overline{G}$ that correspond to the first and last edges of the corresponding walk. This is because we can only guarantee low node congestion of intermediate vertices. Note that that in turn necessitates us sparsifying the graph at every intermediate step as well.

Vertex Sparsifier Chain. Bringing together the above algorithmic components leads to an algorithm for computing a vertex sparsifier chain, whose pseudocode is given in Algorithm 10 below.

Algorithm 10: Eliminate a large subset of vertices for d rounds

procedure ELIMINATE(G, d, ϵ)
  Set $\boldsymbol{L}^{(0)} \leftarrow \boldsymbol{L}$ and $\widehat{\mathcal{T}}_0 = V$.
  Compute a spectral sparsifier $\boldsymbol{M}^{(0)} \approx_\epsilon \boldsymbol{L}^{(0)}$ (Corollary 4.5).
  for $0 < i \le d$ *iteratively* **do**
    Let F_i be an α-DD set of $\boldsymbol{M}^{(i-1)}$ (Lemma 6.7).
    Construct an operator $(\boldsymbol{Z}^{(i)})^{-1}$ that approximates $\boldsymbol{M}^{(i-1)}_{[F,F]}$ (Lemma 6.8).
    Compute $\widetilde{\boldsymbol{M}}^{(i+1)} \approx_\epsilon \mathrm{SC}(\boldsymbol{M}^{i-1}, \widehat{\mathcal{T}}_i)$ (Lemma 6.13) with $\widehat{\mathcal{T}}_i = \widehat{\mathcal{T}}_{i-1} - F_i + U_i$, where U_i is the set of extra vertices added to ensure low congestion.
    Compute an ϵ-spectral sparsifier $\boldsymbol{M}^{(i+1)}$ of $\widetilde{\boldsymbol{M}}^{(i+1)}$. (Corollary 4.5)
  Let $\boldsymbol{Z}_1, \boldsymbol{Z}_2$ be stored implicitly as the product of matrices using the Cholesky factorization (Lemma 2.2).
  return $\boldsymbol{M}^{(d)}, \boldsymbol{Z}_1, \boldsymbol{Z}_2$.

Proof of Lemma 4.10. We start by analyzing the round complexity of the algorithm. By Corollary 4.5, there is a distributed algorithm for computing a sparsifier with $O(n\log^5 n\epsilon^{-2})$ edges in $O(\epsilon^{-2}\log^7 n)$ rounds. We call RANDWALKSCHUR with $\gamma = 1000 \cdot c \cdot \alpha\epsilon^{-2}\log^6 n$, where c is a large enough constant. By Lemma 6.7, we find a 4-DD set of size $n/(8(1+4)) = n/40$. These together imply that the number of vertices in $G^{(i)}$ after i steps in our algorithm is

$$n_i \le \left(1 - \frac{1}{40} + \frac{1}{1000}\right) n_{i-1} \le \left(1 - \frac{1}{50}\right) n_{i-1}.$$

By induction, $n_d \le (\frac{49}{50})^d n$.

In an iteration of the algorithm, the dominating cost is (1) approximating the Schur complement and (2) computing the spectral sparsifier. By our choice of α and γ and Lemma 6.13, the number of rounds required to implement the first one is $O(\epsilon^{-4}\log^7 n(\rho\sqrt{\overline{n}}\log\overline{n}+D))$. The second one introduces a $O(\epsilon^{-2}\log^7 n)$ overhead, which then gives a round complexity of $O(\epsilon^{-6}\log^{14} n(\rho\sqrt{\overline{n}}\log\overline{n}+D))$ per one step in Eliminate. Thus after d steps, the round complexity is:

$$O((\epsilon^{-6}\log^{14} n)^d(\rho\sqrt{\overline{n}}\log\overline{n}+D)).$$

The error is $(1\pm\epsilon)^d$ because we do d rounds of elimination, and each round accumulates $(1\pm\epsilon)$-multiplicative error by using Lemma 6.8 to bound the quality of the inverse of $\boldsymbol{Z}^{(i)}$, Lemma 6.13 to bound the quality of the Schur complement, and Corollary 4.5 to spectrally sparsify the Schur complement, each of which accumulates error $\epsilon/4$. □

7 Implications in Graph Algorithms

In this section we use Theorem 1 to give improved algorithms for maximum flow, min-cost flow, and shortest paths with negative weights in the CONGEST model (Theorems 7, 8, and 9). Our goal is to show that our distributed Laplacian solver can be used to achieve improved complexities for these three problems, so we provide pseudocode in Algorithms 12, 13, 14 (full details of these algorithms are given in Appendix C), and analyze the runtimes in the distributed setting. Our runtimes come from using the Laplacian system solver in Theorem 1 to implement an interior point method until the graph has a low amount of residual flow remaining, which we then route with augmenting path [GU15] or shortest path with positive weights [CM20], both taking $\widetilde{O}(n^{1/2}D^{1/4}+D)$ rounds per iteration. In the remainder of this section, we formalize this reasoning. There are several additional technical pieces, as the algorithms of [Mad16, CMSV17] require changing the graph by adding edges, etc. Throughout, we assume that our Laplacian system solvers are exact – it is justified in the papers [Mad16, CMSV17] that solving to accuracy $1/\mathrm{poly}(m,U)$, where U is the maximum weight / capacity suffices to implement the interior point methods.

In Section 7.3, we simulate Cohen's flow rounding algorithm [Coh95] in the CONGEST model. Given an $s-t$ flow $\vec{f}$, it returns an integral $s-t$ flow $\vec{f}'$ in $O(\sqrt{\overline{m}}\log\overline{m}\cdot D\cdot\log(1/\Delta))$ rounds such that the flow value of $\vec{f}'$ is at least $\vec{f}$'s flow value. In Section 7.4, we implement maximum flow [Mad16] in the CONGEST model, which takes $\widetilde{O}\left(\overline{m}^{3/7}U^{1/7}(\overline{n}^{o(1)}(\overline{n}^{1/2}+D)+\overline{n}^{1/2}D^{1/4})+\overline{m}^{1/2}D\right)$ rounds. In Section 7.5, it takes $\widetilde{O}\left(\overline{m}^{3/7}\overline{n}^{1/2}(n^{o(1)}+D^{1/4})+\overline{m}^{1/2}D\right)$ rounds to execute the min-cost flow [CMSV17] that consists of Laplacian solver, flow rounding and single-source shortest path [CM20] in the CONGEST model. In addition, shortest paths with negative weights can be implemented in the same rounds since it utilizes min-cost flow to make edges non-negative and then compute the shortest paths [CM20].

7.1 Flow Preconditioned Minor

In this subsection, we define flow preconditioned minor. Compared with the definition of ρ congestion, this definition does not bound the size of the pre-image of vertex mapping function $V_{map}^{G\to H}$. Instead, we only allow a bounded number of vertices of G that maps to more than one vertex of H.

Definition 7.1. Let $G=(V,E)$ be a minor of $H=(V_H,E_H)$ (as Definition 4.1). We say this minor is (ρ,α)-flow-preconditioned if:

1. Each edge of H appears as the image of the edge map $E_{map}^{G\to H}(\cdot)$, or in one of the trees connecting supervertices, $T^{G\to H}(v^G)$ for some v^G, at most ρ times.

2. There is a set $U \subset V$ with $|U| = \alpha$ such that the following two conditions hold:

   (a) $S^{G\to H}(v^G) = V_H$ and $T^{G\to H}(v^G)$ is a spanning tree of H with depth at most the diameter of H for each $v^G \in U$.

   (b) $|S^{G\to H}(v^G)| = 1$ for each $v^G \in V \setminus U$.

We say a (ρ, α)-flow-preconditioned minor mapping is stored distributedly, or that G is a (ρ, α)-flow-preconditioned minor distributed over H if it's stored by having all the images of the maps recording their sources (the same as Definition 4.1).

Lemma 7.2. *Let $G = (V, E)$ be a graph with n vertices and m edges that (ρ, α)-flow-preconditioned minor distributes into a communication network $\overline{G} = (\overline{V}, \overline{E})$ with $\overline{n}$ vertices, $\overline{m}$ edges, and diameter D. In the* CONGEST *model, the following operations can be performed using $O(t\alpha D)$ rounds of communication on $\overline{G}$:*

1. *Each $V_{map}^{G\to\overline{G}}(v^G)$ sends $O(t \log n)$ bits of information to all vertices in $S^{G\to\overline{G}}(v^G)$.*

2. *Simultaneously aggregate the sum/minimum of $O(t \log n)$ bits, from all vertices in $S^{G\to\overline{G}}(v^G)$ to $V_{map}^{G\to\overline{G}}(v^G)$ for all $v^G \in V(G)$.*

Proof. Both operations can be achieved by running a BFS or a reverse BFS on $T^{G\to\overline{G}}(v^G)$ for each $v^G \in V$. □

We say a vector $\vec{x} \in \mathbb{R}^V$ on G is distributed on $\overline{G}$ if for each v^G, all the vertices of $S_{map}^{G\to\overline{G}}(v^G)$ records $\vec{x}_{v^G}$.

We say a vector $\vec{f} \in \mathbb{R}^E$ defined on edges of G is distributed to $\overline{G}$ if for each $e \in E$, the two endpoints of $E_{map}^{G\to\overline{G}}(e)$ records $\vec{f}_e$. Sometimes, we treat a vector $\vec{f} \in \mathbb{R}^E$ defined on edges of G as a matrix, denoted as $\boldsymbol{M}_{\vec{f}}$, such that $\boldsymbol{M}_{\vec{f},uv} = \vec{f}_{uv}$ if (u, v) is an edge in E, otherwise $\boldsymbol{M}_{\vec{f},uv} = 0$.

Lemma 7.3. *Let $\overline{G} = (\overline{V}, \overline{E})$ be a communication network $\overline{n}$ vertices, $\overline{m}$ edges, and diameter D. Let $\overline{\mathcal{P}}$ be a collection of paths/cycles of $\overline{G}$ such that every edge of $\overline{G}$ is used for at most ρ times for some $\rho = O(\mathrm{poly}(\overline{m}))$, and for every two consecutive edges $(u, v), (v, w)$ of some path in $\overline{\mathcal{P}}$, v knows that $(u, v), (v, w)$ are two consecutive edges of some path/cycle of $\overline{\mathcal{P}}$. Then the following operations can be performed using $O(\rho\overline{m}^{1/2} \log \overline{m} + D)$ rounds of communication on $\overline{G}$:*

1. *Every path/cycle of $\mathcal{P}$ is associated with a unique ID such that for each edge $e \in E$ in the path/cycle, the two endpoints of $E_{map}^{G\to\overline{G}}(e)$ know the ID.*

2. *Let $\vec{f}$ be an edge vector of G. Compute the sum of $\vec{f}$ for each path/cycle of $\mathcal{P}$ and let the two endpoints of $E_{map}^{G\to\overline{G}}(e)$ know the result for each edge e in the path/cycle.*

Proof. We view $\overline{\mathcal{P}}$ as a graph such that every vertex and edge appears once by treating each appearance of the same vertex/edge as a new vertex/edge. The resulted graph, denoted as G, is a graph of $O(\rho\overline{m})$ vertices and edges that corresponds to a set of edge disjoint paths.

We sample each vertex of G with probability $\log \overline{m}/\overline{m}^{1/2}$. At most $O(\rho \overline{m}^{1/2} \log \overline{m})$ vertices are sampled with high probability, and for each simple path of length $\overline{m}^{1/2}$ that corresponds to an induced subgraph of G, at least one vertex of the simple path is sampled. Each sampled vertices performs a BFS on G until it reaches another sampled vertex or an endpoint of some path in G. We aggregate the result of the BFS (the visit of sampled vertices or endpoints of some path in G) to an arbitrary vertex of $\overline{G}$ in $O(\rho \overline{m}^{1/2} \log \overline{m} + D)$ rounds, and the IDs for paths/cycles of length at least $\overline{m}^{1/2}$ can be assigned and broadcast to each vertex in these paths/cycles in $O(\rho \overline{m}^{1/2} \log \overline{m} + D)$ rounds.

Then, all the paths/cycles that do not contain any sampled vertex initiate a BFS from each vertex in the paths/cycles such that if two BFS collide, only the one initiated by small vertex ID is kept. Since every path/cycle that does not contain any sampled vertex is of length at most $O(\overline{m}^{1/2})$, this step can be done in $O(\overline{m}^{1/2})$ rounds, and afterwards the IDs for these paths/cycles can be computed and broadcast to each vertex on these paths/cycles in $O(\overline{m}^{1/2})$ rounds.

Hence, the first operation can be done in $O(\rho \overline{m}^{1/2} \log \overline{m} + D)$ rounds. The second operation can also be done in $O(\rho \overline{m}^{1/2} \log \overline{m} + D)$ rounds using sampled vertices in a way similar as the first operation. □

7.2 Basic Operations on Flow Preconditioned Minor

In the rest of this section, we will use the following basic operations on flow preconditioned minor in our algorithms: (Let $\overline{G} = (\overline{V}, \overline{E})$ be a communication network, and $G = (V, E)$ be a graph that is a flow-preconditioned minor distributed to $\overline{G}$.)

1. Local Edge Vector Operation: Let $\vec{f}^{(1)}, \vec{f}^{(2)}, \ldots, \vec{f}^{(t)}$ be t edge vectors of G that are distributed on $\overline{G}$. Compute $\vec{f}$ that is an edge vector of G distributed on $\overline{G}$ such that $\vec{f}_e$ is a function of $\vec{f}_e^{(1)}, \vec{f}_e^{(2)}, \ldots, \vec{f}_e^{(t)}$ for each edge $e \in E$.

2. Local Vertex Vector Operation: Let $\vec{x}^{(1)}, \vec{x}^{(2)}, \ldots, \vec{x}^{(t)}$ be t vertex vectors of G that are distributed on $\overline{G}$. Compute $\vec{x}$ that is a vertex vector of G distributed on $\overline{G}$ such that $\vec{x}_{v^G}$ is a function of $\vec{x}_{v^G}^{(1)}, \vec{x}_{v^G}^{(2)}, \ldots, \vec{x}_{v^G}^{(t)}$ for each vertex $v^G \in V$.

3. Norm Operation: For a vertex vector $\vec{x} \in \mathbb{R}^V$ or an edge vector $\vec{f} \in \mathbb{R}^E$ on G distributed to $\overline{G}$ and a $p > 1$, compute p-norm of $\vec{x}$ or $\vec{f}$ and broadcast the result to each vertex of $\overline{G}$.

4. Coordinate Selection Operation: For an edge vector $\vec{f} \in \mathbb{R}^E$ of G distributed on $\overline{G}$ and an integer k, identify top k coordinates of $\vec{f}$ with largest absolute value.

5. Matrix Vector Multiplication Operation: For an edge vector $\vec{f} \in \mathbb{R}^E$ of G and a vertex vector $\vec{x} \in \mathbb{R}^V$ on G both distributed to $\overline{G}$, compute $\boldsymbol{M}_{\vec{f}}\vec{x}$ that is distributed to $\overline{G}$.

We prove the following lemma to bound the number of rounds to perform each basic operation.

Lemma 7.4. *Let $\overline{G} = (\overline{V}, \overline{E})$ be a communication network with $\overline{n}$ vertices and $\overline{m}$ edges, and $G = (V, E)$ be a graph that is (ρ, α)-flow-preconditioned minor distributed to $\overline{G}$. Then, we have*

1. *an algorithm to perform a local vertex vector operation or a local edge vector operation in $O(1)$ rounds;*

2. *an algorithm to perform a norm operation in $O(D)$ rounds;*

3. *an algorithm to perform a coordinate selection operation for $\vec{f}$ in $O(D \cdot \mathrm{poly}(\log \beta))$ rounds, where $\beta = \frac{\max_{e \in E} |\vec{f}_e|}{\min_{e,e' \in E: \vec{f}_e \neq \vec{f}_{e'}} |\vec{f}_e - \vec{f}_{e'}|}$;*

4. *an algorithm to perform a matrix vector multiplication operation in $O(\rho + \alpha D)$ rounds.*

Proof. The local vertex vector operation or local edge vector operation can be computed locally by each vertex of $\overline{G}$.

The norm operation can be computed by a BFS on $\overline{G}$ such that every G by aggregating the sum of $\vec{x}_{v^G}$ for each $v^G \in V$.

The coordinate selection operation can be computed by binary searching k-th largest absolute value of $\vec{f}_e$ among all the $e \in E$ and counting the number of coordinates with absolute value greater than or equal to the value binary searched.

The matrix vector multiplication operation can be implemented by computing $\vec{f}_{uv}\vec{x}_v$ at one endpoint of $E_{map}^{G \to \overline{G}}(uv)$ in $S_{map}^{G \to \overline{G}}(u)$ and taking the sum of $\sum_{v:uv \in E} \vec{f}_{uv}\vec{x}_v$ for each u by $T_{map}^{G \to \overline{G}}(u)$. The number of rounds required is by Lemma 7.2. □

7.3 Flow Rounding

We simulate the flow rounding algorithm by Cohen [Coh95] in the CONGEST model as a subroutine for maximum flow and min-cost flow. The algorithm by Cohen [Coh95] is summarized as Algorith 11.

Lemma 7.5 (Proposition 5.3 of [Coh95])**.** *Let $G = (V, E)$ be a graph with n vertices and m edges, $f : E \to \mathbb{R}^{\geq 0}$ be a s-t flow function, and Δ be a real value such that $1/\Delta$ is a power of 2 and $f(e)$ is an integral multiplication of Δ for every $e \in E$. Then*

1. *Algorithm* FlowRounding *rounds f on edge $e \in E$ to $\lfloor f(e) \rfloor$ or $\lceil f(e) \rceil$ such that the resulted flow has the total flow value not less than f.*

2. *If the total flow value of f is integral and there is an integral cost function $c : E \to \mathbb{Z}^{\geq 0}$, then Algorithm* FlowRounding *rounds f on edge $e \in E$ to $\lfloor f(e) \rfloor$ or $\lceil f(e) \rceil$ such that the resulted flow has the total flow value not less than f, and the total cost not more than f.*

In this section, we present two distributed simulations of Cohen's algorithm, one for maximum flow rounding, and another one for min-cost flow rounding. The underlying reason of two algorithms is that the strategy of choosing path directions (Line 8-11 of Algorithm 11) are different: the former one always choose direction that does not decrease the flow value, and later one always chooses the direction that does not increase total cost.

Lemma 7.6. *Let $\overline{G} = (\overline{V}, \overline{E})$ be a communication network with $\overline{n}$ vertices, $\overline{m}$ edges and diameter D, $G = (V, E)$ be a flow network containing two vertices s and t such that G is a (ρ, α)-flow-preconditioned minor distributed to $\overline{G}$, Δ be a real value such that $1/\Delta$ is an integer that is a power of 2, and (for two vertices $s, t \in \overline{V}$) $\vec{f}$: be a s-t flow function on G such that $f(e)$ is an integral multiplication of Δ for every $e \in \overline{E}$, and $\vec{f}$ is distributed to $\overline{G}$. Then*

1. *There is a distributed algorithm which runs in $O((\rho\sqrt{\overline{m}}(\log \overline{m})^2 + D) \cdot \log(1/\Delta))$ rounds to compute an integer s-t flow function $f' : E \to \mathbb{Z}^+$ such that the flow value of f' at least that of f and $f'(e) \in \{\lfloor f(e) \rfloor, \lceil f(e) \rceil\}$ for every $e \in \overline{E}$.*

Algorithm 11: Flow Rounding

```
1  procedure FlowRounding(G, s, t, f, c, Δ)
2      if the total flow of f is not integral then
3          Add an edge from t to s with flow value the same as total flow.
4      while Δ < 1 do
5          E' ← {(u, v) ∈ E : f(u, v)/Δ is odd}
6          Find an Eulerian partition of E' (ignoring the directions of the edges)
7          for every cycle of the Eulerian partition of E' do
8              if cycle contains the edge (t, s) then
9                  Traverse the cycle such that edge (t, s) is a forward edge.
10             else if cost function c exists then
11                 Traverse the cycle such that the sum of costs on forward edges is no more than
                    the sum of costs on backward edges.
12             else
13                 Traverse the cycle arbitrarily.
14         for every edge (u, v) E' do
15             if (u, v) is a forward edge w.r.t the traversal of the path containing (u, v) then
16                 f(u, v) ← f(u, v) + Δ
17             else
18                 f(u, v) ← f(u, v) − Δ
19         Δ ← 2Δ
20     return f.
```

2. *If the total flow value of f is integral and there is an integral cost function c, then there is a distributed algorithm $O((\rho\sqrt{\overline{m}}(\log\overline{m})^2+D)\cdot\log(1/\Delta))$ rounds to compute an integer s-t flow function $f' : E \to \mathbb{Z}^+$ such that the resulted flow has the total flow value not less than f, and the total cost not more than f.*

Proof. We first extend G such that $S^{G\to\overline{G}}(s) = S^{G\to\overline{G}}(t) = \overline{V}$. The resulted G is a $(\rho+2, \alpha+2)$-flow-preconditioned minor distributed on $\overline{G}$. Then in $O(1)$ rounds, we can add edge (t, s) to G by letting $E^{G\to\overline{G}}_{map}(t, s) = \overline{v}\overline{v}$ for an arbitrary vertex $\overline{v} \in \overline{V}$. The resulted G is still a $(\rho+2, \rho+2)$-flow-preconditioned minor distributed on $\overline{G}$.

Throughout the algorithm, we view the flow function as a vector on E, denoted as $\vec{f}$, that is distributed to $\overline{G}$.

For a $\vec{f}$ and a fixed Δ, E' which is an edge set of G can be identified in $O(1)$ rounds. Now we show that an Eulerian partition of E' can be determined in $O((\alpha+2)D)$ rounds. By the first and second condition of the lemma, for any vertex $v^G \in V$, the number of edges in E' incident to v^G is always even. Hence, to construct an Eulerian partition of E', we only need to pair all the incident edges in E' for each vertex of V. This pairing process can be done in $O(1)$ rounds for all the vertices v^G such that $|V^{G\to\overline{G}}(v^G)| = 1$. For each vertex $v^G \in V$ such that $V^{G\to\overline{G}}(v^G) = \overline{V}$, we simulate the following algorithm in $O(D)$ rounds

such that the pairing process are simulated:

- Run a reverse BFS on $T_{map}^{G\to\overline{G}}(v^G)$ such that for each $v^{\overline{G}} \in \overline{V}$: Let $E'_{v^{\overline{G}}}$ be the union of the edges sent to $v^{\overline{G}}$ from all the children of $v^{\overline{G}}$ in $T^{G\to\overline{G}}(v^G)$ and all the edges in E' incident to v^G whose images by $E_{map}^{G\to\overline{G}}$ are edges incident to $v^{\overline{G}}$. If $|E'_{v^{\overline{G}}}|$ is even, then pair all the edges in $E'_{v^{\overline{G}}}$ arbitrarily, otherwise, send one edge of $E'_{v^{\overline{G}}}$ to the parent of $v^{\overline{G}}$ in $T^{G\to\overline{G}}(v^G)$, and pair the remaining edges of $E'_{v^{\overline{G}}}$ arbitrarily.

In addition, since for every vertex $v^{\overline{G}}$ of $T^{G\to\overline{G}}(v^G)$, every child of $v^{\overline{G}}$ with respect to $T^{G\to\overline{G}}(v^G)$ sends the information of at most one edge to $v^{\overline{G}}$, the Eulerian partition of E' corresponds to a union of cycles of $\overline{G}$ such that every edge of $\overline{G}$ is used at most $\rho + 2$ times.

Lemma 7.3 gives that for each cycle in the Eulerian partition, aggregating the required information takes $\widetilde{O}(\overline{m}^{1/2}\log\overline{m} + D)$ rounds. And since all the cycles are edge disjoint for G, the traversal of each cycle of the Eulerian partition can be determined in $O(\rho\sqrt{\overline{m}}\log\overline{m} + D)$ rounds, and the direction of the traverse of each cycle can be broadcasted to each edge of the cycle in $O(\rho\sqrt{\overline{m}}\log\overline{m} + D)$ rounds.

Hence, simulating one iteration of the while loop on Line 4 of Algorithm FlowRounding takes $O(\rho\sqrt{\overline{m}}\log\overline{m} + D)$ rounds. So the overall number of rounds needed for Algorithm FlowRounding is $O((\rho\sqrt{\overline{m}}\log\overline{m} + D)\cdot\log(1/\Delta))$. □

7.4 Maximum Flow

In this subsection, we present a distributed exact maximum flow algorithm for flow network with integral capacity in $\widetilde{O}\left(\overline{m}^{3/7}U^{1/7}\overline{n}^{o(1)}(\overline{n}^{1/2}D^{1/4} + D) + \overline{m}^{1/2}\right)$ rounds in the CONGEST model, where U is upper bound of the capacities among all the edges. Based on the distributed Laplacian solver, our algorithm simulate the sequential exact maximum flow algorithm by Madry [Mad16]. Madry's sequential algorithm is briefly summarized in Algorithm 12, and the details are given in Section C.1.

Theorem 6 ([GU15, CM20]). *Let G be a (undirected or directed) graph with n vertices, m edges and undirected diameter D, s be a vertex of G, and $\vec{w}$ be an edge vector such that for any edge (u,v) of G, vertices u and v know $w_{u,v}$. Assume in every round, two vertices can send $O(\log n)$ bit information to each other if there is an edge between them in G no matter the direction of the edge. Then there is a distributed SSSP algorithm that in $\widetilde{O}(n^{1/2}D^{1/4} + D)$ rounds computes the distances with respect to $\vec{w}$ from s to all of its reachable vertices as well as an implicit shortest path tree rooted at s such that every vertex knows its parent in the shortest path tree.*

In our distributed maximum flow and min-cost flow algorithm, the graph which we run single source shortest path (SSSP) algorithm on is different to the communication network, because the algorithms we want to simulate [Mad16, CMSV17] add additional vertices and edges to the graph. Hence, we show that this SSSP algorithm can be simulated efficiently if the graph is a flow preconditioned minor distributed to the communication network.

Corollary 7.7. *Let $\overline{G} = (\overline{V}, \overline{E})$ be a communication network with $\overline{n}$ vertices and $\overline{m}$ edges, and $G = (V, E)$ be a (undirected or directed) graph with n vertices, m edges and diameter D that is (ρ,α)-flow-preconditioned minor distributed to $\overline{G}$. Then there is a distributed SSSP algorithm that, for any given source vertex $s^G \in V$, performs $\widetilde{O}(\rho(\overline{n}^{1/2}D^{1/4} + D)\cdot\alpha^2)$ rounds.*

Algorithm 12: MaxFlow (G_0, s, t, U, F)

Input: directed graph $G_0 = (V, E_0, \vec{u})$ with each $e \in E_0$ having two non-negative integer capacities u_e^- and u_e^+; $|V| = n$ and $|E_0| = m$; source s and sink t; the largest integer capacity U; target flow value $F \geq 0$;

Add m undirected edges (t, s) with forward and backward capacities $2U$ to G_0;
for *each* $e = (u, v) \in E_0$ **do**
    Replace e by three undirected edges (u, v), (s, v) and (u, t) whose capacities are u_e;
Let the new graph be $G = (V, E)$;
Initialize the flow vector $\vec{f} \leftarrow \vec{0}$ and dual vector $\vec{y} \leftarrow \vec{0}$;
Update $\vec{f}$ and $\vec{y}$ by solving two Laplacian linear systems on G and a constant number of local vertex/edge vector operations;
Compute the congestion vector $\vec{\rho}$ by a constant number of local edge vector operations;
repeat
    if $\|\vec{\rho}\|_3$ *is at most one computed threshold* **then**
        Update $\vec{f}$ and $\vec{y}$ by solving two Laplacian linear systems on G and a constant number of local vertex/edge vector operations;
        Update $\vec{\rho}$ by a constant number of local vertex/edge vector operation;
    else
        Determine the set S^* that contains the $m^{4\eta}$ edges with the largest $|\rho_e|$ by a coordinate selection operation;
        Update graph G via replacing each edge in S^* by a path and setting some quantities;
until $\widetilde{O}(m^{3/7}U^{1/7})$ *times*;
while *there is an augmenting path from* s *to* t *w.r.t.* $\vec{f}$ *for* G **do**
    Augment an augmenting path for $\vec{f}$ using the shortest path from s to t in the residual graph;

Proof. We assume that excluding the α vertices of G that are mapped to all the vertices of $\overline{G}$, at most one vertex is mapped to any vertex of $\overline{V}$. This is without loss of generality, because if multiple vertices are mapped to the same vertex of $\overline{G}$, then any communication between these vertices are free. In the following, we will call vertices of G that are mapped to all the vertices of $\overline{G}$ *simulated* vertices.

We first explain how the SSSP algorithm of [CM20] can be modified to work as an s-t shortest path algorithm in a setting where only the source (start) vertex s and the sink (target) vertex t (and their incident edges) are simulated vertices. In particular, we argue that the algorithm can be simulated in $\widetilde{O}(\rho(n^{1/2}D^{1/4} + D))$ rounds (where the multiplicative ρ factor simply comes from the fact that every edge of $\overline{G}$ is used at most ρ times for edges in G).

The algorithm of [CM20] consists of eight steps. In Steps 1 and 2, vertices sample themselves with certain probabilities. These steps require no communication and therefore can also be carried out in our setting in which source and sink are just simulated. We slightly modify Step 1 to ensure that the vertex t is never sampled. This does not affect the correctness of the algorithm if we are only interested in computing the shortest path from s to t as shortest paths are simple and thus t will never be an inner vertex on this shortest path.

In Steps 4 and 7, certain auxiliary graphs are created implicitly in the sense that each vertex only knows its incident edges in the auxiliary graph and their respective edge weights. Therefore these steps also require no communication and therefore can also be carried out in our setting in which source and sink are just simulated.

To implement the rest of the algorithm we will rely on the following observation: whenever a step of the algorithm is performed solely by broadcasting or aggregating values via a global BFS tree of the network, then this step immediately can be carried out in our setting with the two simulated vertices as well. This is the case in Steps 5 and 6 of the algorithm.

In Step 8, a certain number of iterations of the Bellman-Ford algorithm is performed on a graph that in addition to the edges of the input graph contains edges from s to certain other vertices. Similar to [CM20], we carry out the first iteration of the Bellman-Ford algorithm – in which the neighbors of s set their tentative distance to the weight of the edge from s – by a global broadcast in $O(D)$ rounds. In [CM20], the remaining iterations of Bellman-Ford are carried out in the standard way where vertices directly communicate with their neighbors. For our modification of [CM20] we do the same, but ignore the vertex t for these iterations. In the end, we explicitly need to ensure that the simulated vertex t also gets to know its distance from s. We achieve this by additionally performing one iteration of the Bellman-Ford iteration in which only the incoming edges of t (and the corresponding neighbors of t) are considered. This can be carried out in $O(D)$ rounds by broadcasting. This works because the incoming neighbors of t (which are part of the communication network and are not just simulated by it) already know their distance from s at this stage due to the previous iterations of Bellman-Ford.

This leaves only Step 3 of the algorithm. In Step 3, Lemma 2.4 of [FN18] is applied to compute approximate distances from each vertex of a set S (where S includes the vertex s, but not the vertex t.). Essentially this Lemma amounts to running a "weighted" version of the breadth-first-search algorithm for each vertex of S (which is repeated $O(\log(nW))$ times with a certain weight rounding applied to the edges in each iteration). The start times of these BFS algorithms are chosen with random delay to guarantee that the congestion at each vertex is low. For the BFS starting at vertex s, the first iteration can be carried out by broadcasting the random delay of s. The neighbors of s (knowing the weight of the edge from s) then know when the "weighted" BFS of s reaches them and can continue with it at the respective time. This gives an additional additive term of $\tilde{O}(D)$ in the running time, which does not affect the asymptotic bounds stated in Lemma 2.4 of [FN18]. This concludes our discussion of the s-t

shortest path algorithm.

Now observe that with the same approach we can obtain an SSSP algorithm in a setting where the source vertex s is the only vertex simulated by the network: we simply need to remove the special handling we had for vertex t in our approach above.

Note that these two algorithmic primitives are sufficient to compute SSSP in a setting where there are α simulated vertices: Let S denote the set of vertices consisting of s^G and the simulated vertices. First, perform an SSSP computation from each vertex $s \in S$ ignoring the other vertices of S (i.e., perform the SSSP computation in the graph $G \setminus S \cup \{s\}$). Then, perform an s-t shortest path computation for each pair of vertices $s, t \in S$, ignoring the other vertices of S (i.e., perform the s-t shortest path computation in the graph $G \setminus S \cup \{s, t\}$). Now each vertex v can reconstruct its distance from s^G by the information it stored so far as the shortest path from s^G to v can be subdivided into subpaths between vertices of S containing no other vertices of S. Overall, we perform $O(\alpha)$ SSSP computations with at most one simulated vertex and $O(\alpha^2)$ s-t shortest path computations with at most two simulated vertices. Finally, note that as soon as each vertex v knows its distance from s^G an implicit shortest path tree (in which each vertex knows its parent in the tree) can be reconstructed by performing a single iteration of the Bellman-Ford algorithm. For the α simulated vertices, we perform this final step in $O(\alpha D)$ rounds by broadcasting via a global BFS tree. □

Theorem 7. *Let $\overline{G} = (\overline{V}, \overline{E})$ be a communication network with $\overline{n}$ vertices, $\overline{m}$ edges, and diameter D, G_0 be a graph and c be an integral capacity function for each edge of G_0 with maximum capacity U satisfying one of the following two conditions:*

1. *G_0 is the same as $\overline{G}$, and for each edge $(u, v) \in \overline{E}$, u and v know the capacity of edge (u, v).*
2. *G_0 is a directed graph obtained by associating each edge of $\overline{G}$ a direction such that for each edge $(u, v) \in \overline{E}$, u and v know the direction of edge (u, v) and its capacity.*

Then there is a distributed algorithm to compute exact s-t maximum flow for two vertices s and t of G_0 in

$$\widetilde{O}\left(\overline{m}^{3/7}U^{1/7}\overline{n}^{o(1)}(\overline{n}^{1/2}D^{1/4} + D) + \overline{m}^{1/2}\right)$$

rounds in the CONGEST model.

Proof. We simulate Algorithm 12. By [Mad16], the accuracy required throughout the algorithm is $1/\mathrm{poly}(\overline{m})$. Without loss of generality, we assume all the values throughout multiplied by 2^γ are integers for some $\gamma = O(\log \overline{m})$. Throughout the algorithm, we set $S^{G_0 \to \overline{G}}(s) = S^{G_0 \to \overline{G}}(t) = \overline{V}$, and for each vertex $v \in \overline{V} \setminus \{s, t\}$, $S^{G_0 \to \overline{G}}(v) = v$.

In line 1, we need to add m parallel (t, s) edges each with capacity U. This step can be simulated in $O(1)$ rounds by specifying an arbitrary vertex in $v^{\overline{G}} \in \overline{V}$ such that m parallel (t, s) edges are mapped to selfloops of $v^{\overline{G}}$.

In line 3, every edge (u, v) with capacity α of G_0 is replace by three edges (u, v), (s, u) and (v, t) with capacity α. Let G denote the graph after line 3. We always make sure that $E_{map}^{G \to \overline{G}}(s, u) = (u, u)$ and $E_{map}^{G \to \overline{G}}(v, t) = (v, v)$. Hence, G is a $(3, 2)$-flow-preconditioned minor distributed to $\overline{G}$.

In line 14, if we replace an edge (u, v) of G by a path $(u, v_1, v_2, \ldots, v_\ell, v)$, then we consider the following two cases:

1. If $E_{map}^{G\to\overline{G}}(u,v)$ is a selfloop of some vertex x in $\overline{G}$, then add all the new vertices $v_1, v_2, \ldots, v_\ell$ such that $V_{map}^{G\to\overline{G}}(v_i) = x$, and all the new edges are also selfloops on x.

2. If $E_{map}^{G\to\overline{G}}(u,v)$ corresponds to an edge of $\overline{G}$, then add all the new vertices $v_1, v_2, \ldots, v_\ell$ such that $S^{G\to\overline{G}}(v_i) = \{V_{map}^{G\to\overline{G}}(u)\}$, and add edges such that $E_{map}^{G\to\overline{G}}(v_\ell, v) = E_{map}^{G\to\overline{G}}(u,v)$ and the remaining edges to be selfloops on $V_{map}^{G\to\overline{G}}(u)$.

Hence, we maintain the invariant that G is $(3,2)$-flow-preconditioned minor distributed to $\overline{G}$.

Note that the execution of each line takes a constant number of basic vector operations in Lemma 7.4 or solves a constant number of Laplacian systems on graph G (excepting replacing an edge by a path in line 14, which can be done in $O(1)$ rounds). By Lemma 7.4, each basic vector operation can be simulated in $O(D\log\overline{m})$ rounds.

To solve a Laplacian system, we first eliminate all the vertices that are added in line 14. Since all these vertices are of degree 2, this elimination can be done locally in each vertex of $\overline{G}$, and the resulted graph is $O(1)$-minor distributed to $\overline{G}$. By Theorem 1, the Laplacian system can be solved in $\overline{n}^{o(1)}(\overline{n}^{1/2}+D)$ rounds. Then we obtain the solution of Laplacian system with respect to G by adding the eliminated vertices back locally.

Since the repeat part takes $O(\overline{m}^{3/7}U^{1/7})$ iterations, the total number of rounds required to simulate all progress steps is $\widetilde{O}(\overline{m}^{3/7+o(1)}U^{1/7}(\overline{n}^{1/2}+D))$.

By Lemma 7.6, the flow rounding takes $O(\log m\cdot(\overline{m}^{1/2}\log^2\overline{m}+D))$ rounds. After the flow rounding, the difference between the flow value and maximum flow value is at most $O(\overline{m}^{3/7}U^{1/7})$. By Corollary 7.7, each iteration of finding an augmenting path takes $\widetilde{O}(D+\overline{n}^{1/2}D^{1/4})$ rounds. Hence, the additional augmenting step takes $\widetilde{O}(\overline{m}^{3/7}U^{1/7}(\overline{n}^{1/2}D^{1/4}+D))$ rounds. □

7.5 Unit Capacity Minimum Cost Flow

In this subsection, we present a distributed minimum cost unit capacity flow algorithm with integral cost in $\widetilde{O}\left(\overline{m}^{3/7+o(1)}(\overline{n}^{1/2}D^{1/4}+D)\right)$ rounds in the CONGEST model. Based on the distributed Laplacian solver, our algorithm simulate the algorithm by Cohen et al. [CMSV17]. The sequential algorithm is briefly summarized in Algorithm 13, and the details of the algorithm are given in Section C.2.

Theorem 8. *Let $\overline{G} = (\overline{V}, \overline{E})$ be a communication network with $\overline{n}$ vertices and $\overline{m}$ edges, G_0 be a directed unweighted graph also defined on $\overline{V}$ such that each edge (u,v) of G_0 is an edge of $\overline{E}$ if the direction is ignored, and u and v in the communication network know the direction of edge (u,v) and its cost. Then, given G_0 and a demand vector $\vec{\sigma}$, there is a distributed algorithm to compute the minimum cost flow for graph G_0 with respect to $\vec{\sigma}$ in*

$$\widetilde{O}(\overline{m}^{3/7+o(1)}(\overline{n}^{1/2}D^{1/4}+D)\mathrm{poly}(\log W))$$

rounds in the CONGEST model.

Proof. We simulate Algorithm 13 and the accuracy required throughout the algorithm is $1/\mathrm{poly}(\overline{m})$. Without loss of generality, we assume all the values throughout the algorithm multiplied by 2^γ are integers for some $\gamma = O(\log\overline{m})$.

To build graph G_1 in line 1, we add vertex v_{aux} to the graph such that $S^{G_1\to\overline{G}}(v_{aux}) = \overline{V}$, and for all the vertices $v\in\overline{V}$, $S^{G_1\to\overline{G}}(v) = \{v\}$. For each edge (u,v) of G_1 with $u,v\in\overline{V}$, we let $E_{map}^{G_1\to\overline{G}}(u,v) = (u,v)$.

Algorithm 13: MinCostFlow $(G, \vec{\sigma}, W)$

Input: directed graph $G_0 = (V_0, E_0, \vec{c}_0)$ with each edge having unit capacity and cost $\vec{c}_0$; $|V_0| = n$ and $|E_0| = m$; integral demand vector $\vec{\sigma}$; the absolute maximum cost W;

Create a new vertex v_{aux} with $\sigma(v_{aux}) = 0$ and add parallel edges (v, v_{aux}) or (v_{aux}, v) according to $\rho(v)$ and degree of v in G_0, and denote the resulted graph as $G_1 = (V_1, E_1, \vec{c}_1)$;

Initialize the bipartite graph $G = (P \cup Q, E, \vec{c})$ with $P \leftarrow V_1$ and $Q \leftarrow \{e_{uv} \mid (u, v) \in E_1\}$ where e_{uv} is a vertex corresponding to edge $(u, v) \in E_1$;

Add a new vertex v_0 and undirected edges (v_0, v) for every $v \in P$ to G;

Initialize the resistance vector $\vec{r}$, flow vector $\vec{f}$, dual vector $\vec{s}$, measure vector $\vec{v}$ and congestion vector $\vec{\rho}$ by a constant number of local vector operations;

for $i = 1$ **to** $1200\sqrt{3}m^{2/7}\log^{4/3} W$ **do**

  Reset the resistances of the auxiliary edges (v_0, v) for each $v \in P$ by a constant number of local vector operations;

  for $j = 1$ **to** $m^{1/7}$ **do**

    while $\|\vec{\rho}\|_{\vec{r},3} > 400\sqrt{3}m^{3/7}\log^{1/3} W$ **do**

      Increase the energy via resetting the resistance r_e and measure ν_e for $e \in E$ by a constant number of local vector operations;

    Update the flow vector $\vec{f}$ and dual vector $\vec{s}$ by solving two Laplacian linear systems in $\boldsymbol{L}(G)$ and a constant number of local vector operation;

Round the solution to be integral by calling FlowRounding and obtain $\vec{M}$;

repeat

  Construct the directed graph $\overrightarrow{G}_M$ of G w.r.t. $\vec{M}$ by a constant number of local edge vector operations;

  Compute the shortest path π in $\overrightarrow{G}_M$;

  Augment $\vec{M}$ using the augmenting path π;

until $\widetilde{O}(m^{3/7})$ *times*;

return $\vec{M}$;

For edge (v, v_{aux}) or (v_{aux}, v), we let $E_{map}^{G_1 \to \overline{G}}(v, v_{aux})$ to be a selfloop on v. The construction of G_1 can be done locally, and the resulted graph is a $(2, 1)$-flow-preconditioned minor distributed to $\overline{G}$.

To construct G, for each vertex $e_{uv} \in Q$, we map e_{uv} to be one of u and v of $\overline{G}$ arbirarily. For vertices of P, the mapping is the same as that of G_1. For each edge (u, e_{uv}) of G, if $V_{map}^{G \to \overline{G}}(u) = V_{map}^{G \to \overline{G}}(e_{uv})$, then we map edge (u, e_{uv}) to a selfloop on $V_{map}^{G \to \overline{G}}(u)$, otherwise, $(V_{map}^{G \to \overline{G}}(u), V_{map}^{G \to \overline{G}}(e_{uv}))$ is an edge of $\overline{G}$, and we set $E_{map}^{G \to \overline{G}}(u, e_{uv})$ to be the edge $(V_{map}^{G \to \overline{G}}(u), V_{map}^{G \to \overline{G}}(e_{uv}))$. Hence, the first two lines can be simulated in $O(1)$ rounds, and the resulting graph G is a $(2, 1)$-flow-preconditioned minor distributed to $\overline{G}$.

To simulate Line 3, we add an additional vertex v_0 to G such that $S^{G \to \overline{G}}(v_0) = \overline{V}$, and add edges (v_0, v) for each $v \in P$ by setting $E_{map}^{G \to \overline{G}}(v_0, v)$ be a selfloop on v if $v \in \overline{V}$, and $E_{map}^{G \to \overline{G}}(v_0, v)$ be a selfloop on an arbitrary vertex of $\overline{V}$ if $v = v_{aux}$. The resulting graph G is a $(3, 2)$-flow-preconditioned minor distributed to $\overline{G}$.

For lines 5-10 of Algorithm 13, the execution of each line takes a constant number of basic vector operations in Lemma 7.4 or solves a Laplacian system on graph G. To solve a Laplacian system, since all the vertices of Q have two incident edges, we eliminate vertices of Q in $O(1)$ rounds, and the resulting graph is $O(1)$-minor distributed to $\overline{G}$. By Theorem 1, every Laplacian system can be solved in $\overline{n}^{o(1)}(\overline{n}^{1/2}+D)$ rounds. Based on the parameter setting, lines 5-10 can be simulated in $\widetilde{O}(\overline{m}^{3/7+o(1)}(\overline{n}^{1/2}+D)\mathrm{poly}(\log W))$ rounds.

Before the flow rounding, we add s and t such that $S^{G\to\overline{G}}(s) = S^{G\to\overline{G}}(t) = \overline{V}$, and add edges by adding selfloops on vertices of $\overline{V}$. The resulting graph is a $(5,4)$-flow-preconditioned minor distributed to $\overline{G}$. By Lemma 7.6, line 11 can be simulated in $\widetilde{O}(\overline{m}^{1/2}+D)$ rounds.

To simulate lines 12-16, all the operations except finding shortest path can be simulated in a way similar to that of lines 5-10. To find the shortest path from $P\cap F_M$ to $Q\cap F_M$, we add an additional vertex v', and edges (v',v) for each $v\in P\cap F_m$ with weight zero to G. The resulted graph is a $(4,3)$-flow-preconditioned minor distributed to $\overline{G}$. By Corollary 7.7, the shortest path can be computed in $\widetilde{O}(D+\overline{n}^{1/2}D^{1/4})$ rounds. The total number of rounds for lines 12-16 is $\widetilde{O}(m^{3/7}(D+\overline{n}^{1/2}D^{1/4}))$ rounds.

Hence, the total number of rounds required is

$$\begin{aligned}&\widetilde{O}(\overline{m}^{3/7+o(1)}(\overline{n}^{1/2}+D)\mathrm{poly}(\log W))+\widetilde{O}(\overline{m}^{1/2}+D)+\widetilde{O}(\overline{m}^{3/7}(D+\overline{n}^{1/2}D^{1/4}))\\ =&\widetilde{O}(\overline{m}^{3/7+o(1)}(\overline{n}^{1/2}D^{1/4}+D)\mathrm{poly}(\log W)).\end{aligned}$$

□

7.6 Negative shortest path

We now give a distributed algorithm for computing single source shortest path with negative weights. It is a direct use of the reduction from shortest paths with negative weights to min-cost flow by Cohen et al. [CMSV17]. Pseudocode of this algorithm is in Algorithm 14.

Algorithm 14: ShortestPaths $(\overline{G}, s, W)$

Input: directed graph $G=(V,E,w)$ with $|V|=n$ and $|E|=m$; source s; the absolute maximum weight W;

`/* Reduction to a weighted perfect` $\vec{1}$`-matching problem */`

1 Let the bipartite graph be $G_{12}=(V_1\cup V_2, E_{12}, w_{12})$ with $V_1=\{v_1 \mid v\in V\}$, $V_2=\{V_2 \mid v\in V\}$, $E_{12}=\{u_1v_2 \mid uv\in E\}\cup\{v_1v_2 \mid v\in V\}$ and $w_{12}(u_1v_2)=\begin{cases}-w_{uv} & uv\in E\\ 0 & u=v\end{cases}$;

2 $(\vec{f},\vec{y})\leftarrow \textsc{MinCostFlow}(G_{12},\vec{1},W)$;

3 **for** *each edge* $(u,v)\in\overline{E}$ **do**

4 $\quad w'_{uv}\leftarrow w_{uv}+y_u-y_v$;

5 Compute the shortest paths with source s on $\overline{G}=(\overline{V},\overline{E},w')$;

Theorem 9. *Let $\overline{G}=(\overline{V},\overline{E})$ be a communication network with $\overline{n}$ vertices and $\overline{m}$ edges, and $w:\overline{E}\to\{-W,-W+1,\ldots,-1,0,1,\ldots,W\}$ be an integral weight function. For a vertex $s\in\overline{V}$, there is a distributed algorithm to compute the shortest path from s to all the other vertices that has a shortest path from s in*

$$\widetilde{O}(\overline{m}^{3/7+o(1)}(\overline{n}^{1/2}D^{1/4}+D)\mathrm{poly}(\log W))$$

rounds in the CONGEST model.

Proof. The strategy of Algorithm 14 is that transferring the given G to a bipartite graph G_{12} and calling the Algorithm MinCostFlow on G_{12} to obtain the dual solution $\vec{y}$, which is utilized to transform the original edge weights in G to be non-negative, and then using the single source shortest path algorithm with non-negative weights on the new instance $G' = (V, E, w)$ to compute the shortest paths.

In a constant number of rounds, we can construct G_{12} based on $\overline{G}$ that is a $(1, 0)$-flow-preconditioned minor distributed to $\overline{G}$. By Theorem 8 and Corollary 7.7, Algorithm 14 can be simulated in the desired number of rounds. □

Acknowledgements

Sebastian Forster is supported by the Austrian Science Fund (FWF): P 32863-N. Yang P. Liu was supported by the Department of Defense (DoD) through the National Defense Science and Engineering Graduate Fellowship (NDSEG) Program. Richard Peng is supported by the National Science Foundation (NSF) under Grant No. 1846218. Xiaorui Sun is supported by start-up funds from University of Illinois at Chicago.

References

[AK20] Mohamad Ahmadi and Fabian Kuhn. Distributed maximum matching verification in CONGEST. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 37:1–37:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[AKM+20] AmirMahdi Ahmadinejad, Jonathan A. Kelner, Jack Murtagh, John Peebles, Aaron Sidford, and Salil P. Vadhan. High-precision estimation of random walks in small space. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1295–1306. IEEE, 2020.

[AKO18] Mohamad Ahmadi, Fabian Kuhn, and Rotem Oshman. Distributed approximate maximum matching in the CONGEST model. In Ulrich Schmid and Josef Widder, editors, *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018*, volume 121 of *LIPIcs*, pages 6:1–6:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[ALGV20] Nima Anari, Kuikui Liu, Shayan Oveis Gharan, and Cynthia Vinzant. Log-concave polynomials IV: exchange properties, tight mixing times, and faster sampling of spanning trees. *CoRR*, abs/2004.07220, 2020.

[AMV20] Kyriakos Axiotis, Aleksander Madry, and Adrian Vladu. Circulation control for faster minimum cost flow in unit-capacity graphs. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 93–104. IEEE, 2020.

[AR19] Udit Agarwal and Vijaya Ramachandran. Distributed weighted all pairs shortest paths through pipelining. In *2019 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2019, Rio de Janeiro, Brazil, May 20-24, 2019*, pages 23–32. IEEE, 2019.

[AR20] Udit Agarwal and Vijaya Ramachandran. Faster deterministic all pairs shortest paths in congest model. In Christian Scheideler and Michael Spear, editors, *SPAA '20: 32nd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, July 15-17, 2020*, pages 11–21. ACM, 2020.

[ARKP18] Udit Agarwal, Vijaya Ramachandran, Valerie King, and Matteo Pontecorvi. A deterministic distributed algorithm for exact weighted all-pairs shortest paths in $\widetilde{O}(n^{3/2})$ rounds. In Calvin Newport and Idit Keidar, editors, *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, pages 199–205. ACM, 2018.

[ASZ20] Alexandr Andoni, Clifford Stein, and Peilin Zhong. Parallel approximate undirected shortest paths via low hop emulators. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 322–335. ACM, 2020.

[BBV04] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[BGK+14] Guy E. Blelloch, Anupam Gupta, Ioannis Koutis, Gary L. Miller, Richard Peng, and Kanat Tangwongsan. Nearly-linear work parallel SDD solvers, low-diameter decomposition, and low-stretch subgraphs. *Theory Comput. Syst.*, 55(3):521–554, 2014.

[BKKL17] Ruben Becker, Andreas Karrenbauer, Sebastian Krinninger, and Christoph Lenzen. Near-optimal approximate shortest paths and transshipment in distributed and streaming models. In Andréa W. Richa, editor, *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, volume 91 of *LIPIcs*, pages 7:1–7:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[BN19] Aaron Bernstein and Danupon Nanongkai. Distributed exact weighted all-pairs shortest paths in near-linear time. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 334–342. ACM, 2019.

[BS07] Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Struct. Algorithms*, 30(4):532–563, 2007. Announced at ICALP '03.

[CDK+21] Parinya Chalermsook, Syamantak Das, Yunbum Kook, Bundit Laekhanukit, Yang P. Liu, Richard Peng, Mark Sellke, and Daniel Vaz. Vertex sparsification for edge connectivity. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1206–1225. SIAM, 2021.

[CGH16] Yun Kuen Cheung, Gramoz Goranci, and Monika Henzinger. Graph minors for preserving terminal distances approximately - lower and upper bounds. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 131:1–131:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[CGP+18] Timothy Chu, Yu Gao, Richard Peng, Sushant Sachdeva, Saurabh Sawlani, and Junxing Wang. Graph sparsification, spectral sketches, and faster resistance computation, via short cycle decompositions. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 361–372. IEEE Computer Society, 2018.

[CKM+11] Paul Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. Electrical flows, Laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 273–282, 2011.

[CKM+14] Michael B. Cohen, Rasmus Kyng, Gary L. Miller, Jakub W. Pachocki, Richard Peng, Anup Rao, and Shen Chen Xu. Solving SDD linear systems in nearly $m \log^{1/2} n$ time. In *STOC*, pages 343–352, 2014.

[CLLM10] Moses Charikar, Tom Leighton, Shi Li, and Ankur Moitra. Vertex sparsifiers and abstract rounding algorithms. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 265–274, 2010.

[CLM+15] Michael B. Cohen, Yin Tat Lee, Cameron Musco, Christopher Musco, Richard Peng, and Aaron Sidford. Uniform sampling for matrix approximation. In Tim Roughgarden, editor, *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 181–190. ACM, 2015.

[CM20] Shiri Chechik and Doron Mukhtar. Single-source shortest paths in the CONGEST model with improved bound. In Yuval Emek and Christian Cachin, editors, *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*, pages 464–473. ACM, 2020.

[CMM17] Michael B. Cohen, Cameron Musco, and Christopher Musco. Input sparsity time low-rank approximation via ridge leverage score sampling. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1758–1777. SIAM, 2017.

[CMSV17] Michael B. Cohen, Aleksander Madry, Piotr Sankowski, and Adrian Vladu. Negative-weight shortest paths and unit capacity minimum cost flow in $\widetilde{O}(m^{10/7} \log W)$ time (extended abstract). In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 752–771. SIAM, 2017.

[Coh95] Edith Cohen. Approximate max-flow on small depth networks. *SIAM Journal on Computing*, 24(3):579–597, 1995.

[CP15] Michael B. Cohen and Richard Peng. ℓ_p row sampling by Lewis weights. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, STOC '15, pages 183–192, New York, NY, USA, 2015. ACM.

[CPZ19] Yi-Jun Chang, Seth Pettie, and Hengjie Zhang. Distributed triangle detection via expander decomposition. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 821–840. SIAM, 2019.

[CS19] Yi-Jun Chang and Thatchaphol Saranurak. Improved distributed expander decomposition and nearly optimal triangle enumeration. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 66–73, 2019.

[CS20] Yi-Jun Chang and Thatchaphol Saranurak. Deterministic distributed expander decomposition and routing with applications in distributed derandomization. *arXiv preprint arXiv:2007.14898*, 2020.

[DEMN21] Michal Dory, Yuval Efron, Sagnik Mukhopadhyay, and Danupon Nanongkai. Distributed weighted min-cut in nearly-optimal time. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1144–1153. ACM, 2021.

[DGGP19] David Durfee, Yu Gao, Gramoz Goranci, and Richard Peng. Fully dynamic spectral vertex sparsifiers and applications. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 914–925. ACM, 2019.

[DGT17] Dean Doron, François Le Gall, and Amnon Ta-Shma. Probabilistic logarithmic-space algorithms for laplacian solvers. In Klaus Jansen, José D. P. Rolim, David Williamson, and Santosh S. Vempala, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, volume 81 of *LIPIcs*, pages 41:1–41:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[DHNS19] Mohit Daga, Monika Henzinger, Danupon Nanongkai, and Thatchaphol Saranurak. Distributed edge connectivity in sublinear time. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 343–354. ACM, 2019.

[DKP+17] David Durfee, Rasmus Kyng, John Peebles, Anup B Rao, and Sushant Sachdeva. Sampling random spanning trees faster than matrix multiplication. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 730–742, 2017.

[DPPR20] David Durfee, John Peebles, Richard Peng, and Anup B. Rao. Determinant-preserving sparsification of SDDM matrices. *SIAM J. Comput.*, 49(4), 2020.

[DSHK+12] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness

of distributed approximation. *SIAM Journal on Computing*, 41(5):1235–1265, 2012. Announced at STOC '11.

[DSMPU15] Atish Das Sarma, Anisur Rahaman Molla, Gopal Pandurangan, and Eli Upfal. Fast distributed pagerank computation. *Theor. Comput. Sci.*, 561:113–121, 2015.

[DSNPT13] Atish Das Sarma, Danupon Nanongkai, Gopal Pandurangan, and Prasad Tetali. Distributed random walks. *Journal of the ACM (JACM)*, 60(1):1–31, 2013.

[DST17] Dean Doron, Amir Sarid, and Amnon Ta-Shma. On approximating the eigenvalues of stochastic matrices in probabilistic logspace. *Comput. Complex.*, 26(2):393–420, 2017.

[EGK+14] Matthias Englert, Anupam Gupta, Robert Krauthgamer, Harald Räcke, Inbal Talgam-Cohen, and Kunal Talwar. Vertex sparsifiers: New results from old techniques. *SIAM J. Comput.*, 43(4):1239–1262, 2014. Announced at APPROX-RANDOM '10.

[Elk06] Michael Elkin. An unconditional lower bound on the time-approximation trade-off for the distributed minimum spanning tree problem. *SIAM J. Comput.*, 36(2):433–456, 2006. Announced at STOC '04.

[Elk20] Michael Elkin. Distributed exact shortest paths in sublinear time. *J. ACM*, 67(3):15:1–15:36, 2020. Announced at STOC '17.

[EN18] Michael Elkin and Ofer Neiman. On efficient distributed construction of near optimal routing schemes. *Distributed Comput.*, 31(2):119–137, 2018. Announced at PODC '16.

[EN19a] Michael Elkin and Ofer Neiman. Hopsets with constant hopbound, and applications to approximate shortest paths. *SIAM J. Comput.*, 48(4):1436–1480, 2019. Announced at FOCS '16.

[EN19b] Michael Elkin and Ofer Neiman. Linear-size hopsets with small hopbound, and constant-hopbound hopsets in RNC. In Christian Scheideler and Petra Berenbrink, editors, *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 333–341. ACM, 2019.

[FF56] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[FN18] Sebastian Forster and Danupon Nanongkai. A faster distributed single-source shortest paths algorithm. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 686–697. IEEE Computer Society, 2018.

[GB20] Iqra Altaf Gillani and Amitabha Bagchi. A queueing network-based distributed laplacian solver. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '20, page 535–537, New York, NY, USA, 2020. Association for Computing Machinery.

[GHS83] Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983.

[GK13] Mohsen Ghaffari and Fabian Kuhn. Distributed minimum cut approximation. In Yehuda Afek, editor, *Distributed Computing - 27th International Symposium, DISC 2013, Jerusalem, Israel, October 14-18, 2013. Proceedings*, volume 8205 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2013.

[GKK+15] Mohsen Ghaffari, Andreas Karrenbauer, Fabian Kuhn, Christoph Lenzen, and Boaz Patt-Shamir. Near-optimal distributed maximum flow: Extended abstract. In Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 81–90. ACM, 2015.

[GKP98] Juan A. Garay, Shay Kutten, and David Peleg. A sublinear time distributed algorithm for minimum-weight spanning trees. *SIAM J. Comput.*, 27(1):302–316, 1998. Announced at FOCS '93.

[GL18] Mohsen Ghaffari and Jason Li. Improved distributed algorithms for exact shortest paths. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 431–444. ACM, 2018.

[GNT20] Mohsen Ghaffari, Krzysztof Nowicki, and Mikkel Thorup. Faster algorithms for edge connectivity via random 2-out contractions. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1260–1279. SIAM, 2020.

[Gor19] Gramoz Goranci. Dynamic graph algorithms and graph sparsification: New techniques and connections. *CoRR*, abs/1909.06413, 2019.

[Gre96] Keith D Gremban. *Combinatorial preconditioners for sparse, symmetric, diagonally dominant linear systems.* PhD thesis, Carnegie Mellon University, 1996.

[GU15] Mohsen Ghaffari and Rajan Udwani. Brief announcement: Distributed single-source reachability. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 163–165, 2015.

[HK73] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

[HKN16] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A deterministic almost-tight distributed algorithm for approximating single-source shortest paths. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 489–498. ACM, 2016.

[HNS17] Chien-Chung Huang, Danupon Nanongkai, and Thatchaphol Saranurak. Distributed exact weighted all-pairs shortest paths in $\widetilde{O}(n^{5/4})$ rounds. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 168–179. IEEE Computer Society, 2017.

[HW12] Stephan Holzer and Roger Wattenhofer. Optimal distributed all pairs shortest paths and applications. In Darek Kowalski and Alessandro Panconesi, editors, *ACM Symposium on Principles of Distributed Computing, PODC '12, Funchal, Madeira, Portugal, July 16-18, 2012*, pages 355–364. ACM, 2012.

[IG17] Taisuke Izumi and François Le Gall. Triangle finding and listing in CONGEST networks. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 381–389. ACM, 2017.

[Ind06] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *Journal of the ACM (JACM)*, 53(3):307–323, 2006.

[JLS19] Arun Jambulapati, Yang P. Liu, and Aaron Sidford. Parallel reachability in almost linear work and square root depth. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1664–1686. IEEE Computer Society, 2019.

[KLOS14] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 217–226, 2014.

[KLP+16] Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A. Spielman. Sparsified cholesky and multigrid solvers for connection laplacians. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 842–850. ACM, 2016.

[KMP10] Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving SDD linear systems. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 235–244, 2010.

[KMP11] Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-m log n time solver for SDD linear systems. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 590–598, 2011.

[KNZ14] Robert Krauthgamer, Huy L. Nguyen, and Tamar Zondiner. Preserving terminal distances using minors. *SIAM J. Discret. Math.*, 28(1):127–141, 2014.

[KOSA13] Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving SDD systems in nearly-linear time. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 911–920, 2013.

[KP98] Shay Kutten and David Peleg. Fast distributed construction of small k-dominating sets and applications. *J. Algorithms*, 28(1):40–66, 1998.

[KR13] Robert Krauthgamer and Inbal Rika. Mimicking networks and succinct representations of terminal cuts. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1789–1799. SIAM, 2013.

[KS16] Rasmus Kyng and Sushant Sachdeva. Approximate gaussian elimination for laplacians - fast, sparse, and simple. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 573–582. IEEE Computer Society, 2016.

[KX16] Ioannis Koutis and Shen Chen Xu. Simple parallel and distributed algorithms for spectral graph sparsification. *ACM Trans. Parallel Comput.*, 3(2):14:1–14:14, 2016.

[Kyn17] Rasmus Kyng. *Approximate Gaussian Elimination.* PhD thesis, Yale University, 2017. Available at: http://rasmuskyng.com/rjkyng-dissertation.pdf.

[Li20] Jason Li. Faster parallel algorithm for approximate shortest path. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 308–321. ACM, 2020.

[LM10] F Thomson Leighton and Ankur Moitra. Extensions and limits to vertex sparsification. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 47–56. ACM, 2010.

[LP16] Russell Lyons and Yuval Peres. *Probability on Trees and Networks.* Cambridge University Press, 2016.

[LPP19] Christoph Lenzen, Boaz Patt-Shamir, and David Peleg. Distributed distance computation and routing with small messages. *Distributed Comput.*, 32(2):133–157, 2019.

[LPS15] Yin Tat Lee, Richard Peng, and Daniel A. Spielman. Sparsified Cholesky solvers for SDD linear systems. *CoRR*, abs/1506.08204, 2015.

[LS18] Huan Li and Aaron Schild. Spectral subspace sparsification. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 385–396. IEEE, 2018.

[LS20a] Yang P. Liu and Aaron Sidford. Faster divergence maximization for faster maximum flow. *CoRR*, abs/2003.08929, 2020.

[LS20b] Yang P. Liu and Aaron Sidford. Faster energy maximization for faster maximum flow. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 803–814. ACM, 2020.

[Mad16] Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 593–602, 2016. Available at: https://arxiv.org/abs/1608.06016.

[Moi09] Ankur Moitra. Approximation algorithms for multicommodity-type problems with guarantees independent of the graph size. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 3–12. IEEE, 2009.

[MP13] Gary L. Miller and Richard Peng. Approximate maximum flow on separable undirected graphs. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1151–1170. Society for Industrial and Applied Mathematics, 2013. Available at http://arxiv.org/abs/1210.5227.

[MR89] Gary L. Miller and John H. Reif. Parallel tree contraction part 1: Fundamentals. *Adv. Comput. Res.*, 5:47–72, 1989.

[MRSV17] Jack Murtagh, Omer Reingold, Aaron Sidford, and Salil P. Vadhan. Derandomization beyond connectivity: Undirected laplacian systems in nearly logarithmic space. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 801–812. IEEE Computer Society, 2017.

[MRSV19] Jack Murtagh, Omer Reingold, Aaron Sidford, and Salil P. Vadhan. Deterministic approximation of random walks in small space. In Dimitris Achlioptas and László A. Végh, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2019, September 20-22, 2019, Massachusetts Institute of Technology, Cambridge, MA, USA*, volume 145 of *LIPIcs*, pages 42:1–42:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[Nan14] Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 565–573. ACM, 2014.

[NS14] Danupon Nanongkai and Hsin-Hao Su. Almost-tight distributed minimum cut algorithms. In Fabian Kuhn, editor, *Distributed Computing - 28th International Symposium, DISC 2014, Austin, TX, USA, October 12-15, 2014. Proceedings*, volume 8784 of *Lecture Notes in Computer Science*, pages 439–453. Springer, 2014.

[Pel00] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, Philadelphia, PA, 2000.

[Pen13] Richard Peng. *Algorithm Design Using Spectral Graph Theory*. PhD thesis, Carnegie Mellon University, Pittsburgh, August 2013. Available at: http://reports-archive.adm.cs.cmu.edu/anon/2013/CMU-CS-13-121.pdf.

[Pen16] Richard Peng. Approximate undirected maximum flows in $O(mpolylog(n))$ time. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1862–1867. SIAM, 2016. Available at http://arxiv.org/abs/1411.7631.

[PR00] David Peleg and Vitaly Rubinovich. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM J. Comput.*, 30(5):1427–1442, 2000. Announced at FOCS '93.

[PS14] Richard Peng and Daniel A. Spielman. An efficient parallel solver for SDD linear systems. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 333–342. ACM, 2014.

[Räc02] Harald Räcke. Minimizing congestion in general networks. In *43rd Symposium on Foundations of Computer Science (FOCS 2002), 16-19 November 2002, Vancouver, BC, Canada, Proceedings*, pages 43–52. IEEE Computer Society, 2002.

[RST14] Harald Räcke, Chintan Shah, and Hanjo Täubig. Computing cut-based hierarchical decompositions in almost linear time. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 227–238. SIAM, 2014.

[She13] Jonah Sherman. Nearly maximum flows in nearly linear time. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 263–269, 2013.

[SRS18] Aaron Schild, Satish Rao, and Nikhil Srivastava. Localization of electrical flows. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1577–1584. SIAM, 2018.

[SS11] Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM J. Comput.*, 40(6):1913–1926, 2011.

[ST14] D. Spielman and S. Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM Journal on Matrix Analysis and Applications*, 35(3):835–885, 2014. Available at http://arxiv.org/abs/cs/0607105.

[Tro11] Joel Tropp. Freedman's inequality for matrix martingales. *Electronic Communications in Probability*, 16:262–270, 2011.

[vdBLN+20] Jan van den Brand, Yin Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 919–930. IEEE, 2020.

[Vis12] Nisheeth K. Vishnoi. *$Lx = b$ Laplacian Solvers and Their Algorithmic Applications.* now publishers, 2012.

A Lower Bound

Theorem 2. *In the* CONGEST *model of computation, solving Laplacian systems to accuracy $\epsilon \leq \frac{1}{2}$ requires at least $\widetilde{\Omega}(n^{1/2} + D)$ rounds of communication.*

Proof. Note first that any low-accuracy solver with accuracy $\epsilon \leq \frac{1}{2}$ can be boosted to a solver with precision $\tilde{\epsilon}$ at the cost of increasing the running time by a factor of $O(\log \tilde{\epsilon}^{-1})$ (see Lemma 1.6.8. in [Pen13]). This is done by running $O(\log \tilde{\epsilon}^{-1})$ iterations of the iterative refinement method, which in each iteration performs one matrix-vector product, one vector substraction, one vector addition, and one call to the

low-accuracy solver. Excluding the call to the low-accuracy solver, all of these operations can be performed in a constant number of rounds. The running time of the iterative refinement method is therefore dominated by the $O(\log \tilde{\epsilon}^{-1})$ calls to the low-accuracy solver. We thus assume in the following that we are given a high-accuracy solver with accuracy $\epsilon = 1/\mathrm{poly}(n)$, which as just argued requires only an overhead of $O(\log n)$ in the number of rounds compared to a low-accuracy solver.

To prove the lower bound we use the framework of Das Sarma et. al. [DSHK+12] who established an $\Omega(\sqrt{\overline{n}}/(\log \overline{n}) + D)$ lower bound for the following verification problem: Given a subgraph H of the communication network $\overline{G}$ (which has $\overline{n}$ nodes and diameter D) and two nodes s and t, the network needs to decide whether s and t are connected (i.e., lie in the same connected component). In their lower bound construction, the distance between s and t in $\overline{G}$ is $O(\log n)$. We show that any algorithm for solving Laplacian Systems up to small enough error $\epsilon = 1/\mathrm{poly}(n)$ can be used to give s and t the information whether they are connected in H in additional $\mathrm{dist}_{\overline{G}}(s,t) = O(\log n)$ rounds. In particular, we exploit that such a solver can be used to compute an approximation to the effective s-t resistance.

Define the weighted graph H' (which we view as a resistor network) as having the same nodes and edges as $\overline{G}$ and resistances $r_e = 1$ for every edge $e \in E(H)$ and $r_e = n$ for every edge $e \notin E(H)$. Let $\chi_{s,t}$ be the n-dimensional vector is 1 at the coordinate corresponding to s, -1 at the coordinate corresponding to t, and 0 otherwise. It is well known (see, e.g., [CKM+11, Vis12]) that $\mathrm{res}_{H'}(s,t) = \phi(s) - \phi(t)$ for any vector ϕ satisfying $\boldsymbol{L}(H')\phi = \chi_{s,t}$, where $\phi(s)$ and $\phi(t)$ are the values of the coordinates corresponding to s and t, respectively. Let $\vec{\phi}'$ be an approximate solution with error ϵ to the linear system $\boldsymbol{L}(H')\vec{\phi} = \vec{\chi}_{s,t}$, i.e.,

$$\left\| \vec{\phi}' - \boldsymbol{L}\left(H'\right)^{\dagger} \vec{\chi}_{s,t} \right\|_{\boldsymbol{L}(H')} \leq \epsilon \cdot \left\| \vec{\chi}_{s,t} \right\|_{\boldsymbol{L}(H')^{\dagger}} . \tag{67}$$

Henceforth let $\phi = \boldsymbol{L}\left(H'\right)^{\dagger} \vec{\chi}_{s,t}$. Thus, (67) is equivalent to the statement

$$\left\| \vec{\phi}' - \phi \right\|_{\boldsymbol{L}(H')} \leq \epsilon \cdot \sqrt{\mathrm{res}_{H'}(s,t)} \, . \tag{68}$$

It is well-known that for any Laplacian matrix $\boldsymbol{L}$ with integer resistances from 1 to $R = \mathrm{poly}(n)$ this first eigenvalue is $\lambda_1(\boldsymbol{L}) = 0$ the second eigenvalue is bounded by $\lambda_2(\boldsymbol{L}) \geq \frac{1}{\mathrm{poly}(n)}$ and thus for any vector $\vec{x}$ the following bounds relating the matrix norm to the infinity norm hold:

$$\|\vec{x}\|_{\infty} \leq \|\vec{x}\|_2 \leq \frac{1}{\sqrt{\left\|\boldsymbol{L}^{\dagger}\right\|_2}} \|\vec{x}\|_L = \frac{1}{\lambda_2(\boldsymbol{L})} \|\vec{x}\|_L \leq \mathrm{poly}(n) \|\vec{x}\|_L \tag{69}$$

The combination of (68) and (69) together with the estimate $\mathrm{res}_{H'}(s,t) \leq nR$ gives

$$\left\| \vec{\phi}' - \phi \right\|_{\infty} \leq \epsilon \cdot \mathrm{poly}(n) \, . \tag{70}$$

By setting ϵ to a small enough value inversely polynomial in n, (70) implies

$$\mathrm{res}_{H'}(s,t) - 0.25 \leq \phi'(s) - \phi'(t) \leq \mathrm{res}_{H'}(s,t) + 0.25 \, .$$

In the rest of the proof, we argue that knowing the value $\phi'(s) - \phi'(t)$ (which can be made known to both s and t in $\mathrm{dist}_{\overline{G}}$ rounds) suffices to decide whether s and t are connected in H. If s and t are connected in H, then – since effective resistances obey the triangle inequality – the effective s-t resistance in H' is upper-bounded by the length of the shortest path between s and t in H, i.e., $\mathrm{res}_{H'}(s,t) \leq n - 1$. If s and t

are not connected in H, then let S be the connected component containing S and let $e_1, \ldots, e_k$ (for some $k \le m \le n^2$) be the edges leaving S in H. By the Nash-Williams inequality (see, e.g., [LP16], chapter 2.5), the effective s-t resistance is at least

$$\mathrm{res}_{H'}(s,t) \ge \frac{1}{\sum_{i=1}^{k} \frac{1}{r(e_i)}} = \frac{1}{\sum_{i=1}^{k} \frac{1}{(1+\epsilon)^2 n^3}} \ge (1+\epsilon)^2 n\,.$$

Thus, s and t are connected in H if and only if $\phi'(s) - \phi'(t) \le n - 0.5$. □

B Building Blocks for Distributed Minors

Here we give the deferred proofs from Section 4.1. All of our algorithms are based on placing about $\sqrt{\overline{n}}$ special vertices, picked randomly so that any vertex is within a distance of about $\sqrt{\overline{n}}$ from these, and then aggregating information at these special vertices globally via a DFS tree in about $\sqrt{\overline{n}} + D$ rounds.

For this, it is useful to define a tree decomposition scheme for the set of overlapping trees used to connect the supervertices.

Lemma B.1. *There is an algorithm* SpecialVertices *that takes an input a forest F specified with mappings of vertices and edges into a graph $\overline{G}$ such that each vertex of $\overline{G}$ appears in at most ρ trees of F, and each edge of $\overline{G}$ is used in at most ρ trees of F, and returns after $O(\rho\sqrt{\overline{n}})$ rounds of communication a collection of $O(\rho\sqrt{\overline{n}}\log n)$ special vertices of F, labeled at their mapped vertices in $\overline{G}$, such that with high probability, for each T in F, we have:*

1. *either the diameter of T is at most $\sqrt{\overline{n}}$,*
2. *or for any vertex of T,*
   (a) *it can reach at most 2 special vertices, without going through more special vertices.*
   (b) *its distance in T to closest special vertex is at most $O(\sqrt{\overline{n}})$.*

Algorithm 15: Partition all trees of a forest into low diameter pieces via special vertices

```
1 procedure SpecialVertices (F, Ḡ)
2     Sample K by including each vertex in each tree of F with probability log n̄ / n̄^{1/2}.
3     for O(√n̄) rounds do
4         Each vertex propagate to all its neighbors whether taking that edge towards it leads to a
          vertex in K.
5     Add all vertices that can reach special vertices in three or more directions to K.
```

Proof. Consider SpecialVertices in Algorithm 15.

The congestion bound gives that the total number of vertices among the trees is at most $O(\rho\overline{n})$. This means picking $O(\rho\sqrt{\overline{n}}\log\overline{n})$ random vertices from

$$\bigcup_{v^G \in V(G)} S^{G\to\overline{G}}(v^G),$$

ensures that with high probability, the maximum distance to a special vertex along any tree path is $O(\sqrt{\overline{n}})$ with high probability.

This means that in $O(\sqrt{\overline{n}})$ rounds of propagation, we can find, for each edge in each tree, whether there is a special vertex in either direction. Formally, the local operation at each vertex is to check whether there are 2 or more directions from it that lead to special vertices: if there are, then all edges entering this vertex are part of paths that reach special vertices. Otherwise, all except that one direction that the path from a special vertex came from can be continued, so we push 'possible' along all except that direction.

By declaring all vertices with three or more edges leaving it that lead to special vertices as special themselves. As this only adds in the lowest common ancestors of the previous special vertices, it only increases the number of special vertices by a constant factor. This step is also completely local, so the total cost is dominated by the propagation steps. □

This partition routing allows us to root all of the trees.

Proof. (of Lemma 4.2) We first run the partition scheme SpecialVertices on the forest that's the unions of the spanning trees of the supernodes of G. Lemma B.1 gives that each resulting piece consisting of edges reachable to each other without going through special vertices have diameter at most $O(\sqrt{\overline{n}})$. So $O(\sqrt{\overline{n}})$ rounds of propagation lets the root vertex inform all nodes in its piece. In this number of rounds, we also propagate the ID of these special vertices, as well as distances to them, to all vertices in each piece using another $O(\sqrt{\overline{n}})$ rounds of communication.

Note this in particular allows the two special nodes on each piece to know their distance to each other. The amount of information aggregated at each special vertex may be large. Aggregating these information centrally along the BFS tree of $\overline{G}$ then allows us to find the distance from the root vertex to all the special vertices in their piece in a centralized manner. Once this information is propagated back, each edge can just be oriented towards the direction of the special vertex closer to the root, giving the desired orientations. □

This direction to the root is necessary for propagating information that cannot be duplicated, such as the sum of values. Using it, we can prove our main communication tool, Lemma 4.3.

Proof. (of Lemma 4.3) Once again we run the partition scheme SpecialVertices. Lemma B.1 ensures that all paths hit one such vertex after at most $O(\sqrt{\overline{n}})$ steps. After that, we root the tree using Lemma 4.2.

For the push case, we repeatedly push information from vertices to their neighbors. Each such step costs $O(\rho)$ due to the vertex congestion bound. By the bound above, after $O(\sqrt{\overline{n}})$ such steps (which costs a total of $O(\rho\sqrt{\overline{n}})$ rounds, this information either reached all nodes in the corresponding supernode, or some special vertex.

Getting all info on special nodes to a center node (along with the ID of v^G that they originated from) over a BFS tree takes $O(\rho\sqrt{\overline{n}}\log\overline{n} + D)$ rounds, after which they can also be re-distributed to all special nodes. Then by the distance bound from Lemma B.1 another $O(\rho\sqrt{\overline{n}})$ rounds of propagations to neighbors passes the information to everyone.

The aggregation of sum or minimum follows similarly. We repeat $O(\rho\sqrt{\overline{n}})$ rounds of all non-special vertices propagating their sum, or min, up to their parents. In case of sum, once a value 'floats' to its parent, it's set to 0 at the current vertex so we do not over count. Finally, all the information at the special vertices are aggregated via the global BFS tree, and passed to the corresponding root vertices. □

Proof. (of Lemma 4.6) Let

$$S^{G_1 \to G_2} : V(G_1) \to V(G_2)^*$$
$$V_{map}^{G_1 \to G_2} : V(G_1) \to V(G_2)$$
$$T^{G_1 \to G_2} : V(G_1) \to E(G_2)^*$$
$$E_{map}^{G_1 \to G_2} : E(G_1) \to E(G_2)$$

be the maps from G_1 to G_2, and similarly, let the mapping from G_2 to $\overline{G}$ be:

$$S^{G_2 \to \overline{G}} : V(G_2) \to V\left(\overline{G}\right)^*$$
$$V_{map}^{G_2 \to \overline{G}} : V(G_2) \to V\left(\overline{G}\right)$$
$$T^{G_2 \to \overline{G}} : V(G_2) \to E\left(\overline{G}\right)^*$$
$$E_{map}^{G_2 \to \overline{G}} : E(G_2) \to E\left(\overline{G}\right)$$

We will construct the mapping from G to $\overline{G}$ from these. The edge and vertex maps are directly by composition:

$$V_{map}^{G_1 \to \overline{G}}\left(v^{G_1}\right) = V_{map}^{G_2 \to \overline{G}}\left(V_{map}^{G_1 \to G_2}\left(v^{G_1}\right)\right)$$
$$E_{map}^{G_1 \to \overline{G}}\left(e^{G_1}\right) = E_{map}^{G_2 \to \overline{G}}\left(E_{map}^{G_1 \to G_2}\left(e^{G_1}\right)\right)$$

The edge mapping is a direct transfer of the pre-images, locally per edge. While the vertex label propagation is via one round of communication along new supervertex, via Lemma 4.3.

So we can focus on the construction of new supervertices and their spanning trees. For $v^{G_1} \in V(G_1)$, we let

$$S^{G_1 \to \overline{G}}\left(v^{G_1}\right) = \bigcup_{v^{G_2} \in S^{G_1 \to G_2}\left(v^{G_1}\right)} S_{G_2 \to \overline{G}}\left(v^{G_2}\right)$$

with corresponding spanning tree a subset of the edges

$$\bigcup_{v^{G_2} \in S^{G_1 \to G_2}\left(v^{G_1}\right)} T^{G_2 \to \overline{G}}\left(v^{G_2}\right).$$

To compute this union we have each root vertex of each supervertex corresponding to some v^{G_2} inform the entire supervertex of the new ID in G_1. As each vertex of G_2 corresponds to the image of at most ρ_1 v^{G_1}s, this mapping takes $O(\rho_1 \rho_2(\sqrt{n}\log n + D))$ iterations.

Then the edges of $\overline{G}$ with the new labels (of vertex ID from G_1) gives the corresponding supervertices. That is, $S^{G_1 \to \overline{G}}(v^{G_1})$ is simply the set of vertices that received the label v^{G_1} after we propagated from the root vertices of G_2 in $\overline{G}$.

We then need to find spanning forests among these unions of trees. We do so with a variant of Brouvka's algorithm, combined with parallel tree contraction. Specifically, we iterate the following $O(\log n)$ times:

1. Each remaining vertex (in each of the super vertices) pick a random priority.
2. Each vertex identify highest priority neighbor, computed on the minor with some edges already contracted using Lemma 4.3.
3. Contract either:
   (a) all the leaf vertices (degree 1 nodes),
   (b) a vertex disjoint subset of edges with both end points having degree 2.

In the second case of contracting edges with both endpoints having degree 2, we can find an independent set of such edges whose total size is at least a constant factor of all such edges by simply picking a random subset with probability 1/5, and dropping the ones where another end point is picked.

This procedure reduces the number of vertices in each component a constant factor in expectation. This is because the first step ensures the edges found form a tree with edge count at least half the number of vertices. Then parallel tree contraction [MR89] ensures that either the number of leaves, or the number of edges with both end points degree 2 is at least a constant factor of the tree size. Contracting the larger set of these then gives the desired constant factor progress, so it terminates in $O(\log n)$ rounds.

Note that in subsequent rounds, the edges already identified to be part of $T^{G_1 \to \overline{G}}(v^G)$ form a spanning forest, and we're working on the minor with this forest contracted. This means we need to invoke Lemma 4.3 repeatedly to do the neighborhood aggregations on this contracted graph. □

Proof. (of Corollary 4.7) We want to simplify F into a sequence of subgraphs that have simple 1-minor distributions into G. After that, we can invoke Lemma 4.6 to make progress.

We repeat the same tree contraction procedure used for finding spanning forests in Lemma 4.6 above. It gives that at each step, we're computing G/F for a set of vertex disjoint stars F.

Given such a F, we can then construct a 1-minor distribution of G/F into G by:

1. Having the center vertex generate the new vertex ID.
2. Propagate this ID to vertices in the corresponding supervertex in $\overline{G}$.
3. Have edges that declared themselves part of F pass this info from one end point to the other.
4. All leaf supervertices then pull this new vertex ID into their root vertex.

after which invoking Lemma 4.6 gives the minor. This halves the number of □

C Max flow and Minimum Cost Flow Algorithm

In this section, we give the missing details of the max flow algorithm and minimum cost flow algorithm as Section 7.

C.1 Max Flow Algorithm

In this section, we give the missing subroutines of Algorithm 16. The subroutines AUGMENTATION, FIXING and BOOSTING are shown in Algorithm 17, 18 and 19 respectively.

Algorithm 16: MAXFLOW (G_0, s, t, U, F)

Input: directed graph $G_0 = (V, E_0, \vec{u})$ with each $e \in E_0$ having two non-negative integer capacities u_e^- and u_e^+; $|V| = n$ and $|E_0| = m$; source s and sink t; the largest integer capacity U; target flow value $F \geq 0$;

`/* Preconditioning Edges */`

Add m undirected edges (t, s) with forward and backward capacities $2U$ to G_0;

`/* Initialization */`

for *each* $e = (u, v) \in E_0$ **do**

  replace e by three undirected edges (u, v), (s, v) and (u, t) whose capacities are u_e;

Let the new graph be $G = (V, E)$;

Initialize $\vec{f} \leftarrow \vec{0}$ and $\vec{y} \leftarrow \vec{0}$;

`/* Progress Step */`

$\vec{\widetilde{f}}, \vec{\widehat{f}}, \vec{\widehat{y}} \leftarrow \text{AUGMENTATION}(G, s, t, F)$;

Compute $\vec{\rho}$ by letting $\rho_e \leftarrow \frac{\widetilde{f}_e}{\min\{u_e^+ - f_e, u_e^- + f_e\}}$;

$\vec{f}, \vec{y} \leftarrow \text{FIXING}\left(G, \vec{\widehat{f}}, \vec{\widehat{y}}\right)$;

$\eta \leftarrow \frac{1}{14} - \frac{1}{7}\log_m U - O(\log\log(mU))$, $\widehat{\delta} \leftarrow \frac{1}{m^{\frac{1}{2}-\eta}}$;

for $t = 1$ **to** $100 \cdot \frac{1}{\widehat{\delta}} \cdot \log U$ **do**

  if $\|\vec{\rho}\|_3 \leq \frac{m^{\frac{1}{2}-\eta}}{33(1-\alpha)}$ **then**

    $\delta \leftarrow \frac{1}{33(1-\alpha)\|\vec{\rho}\|_3}$;

    $\vec{\widetilde{f}}, \vec{\widehat{f}}, \vec{\widehat{y}} \leftarrow \text{AUGMENTATION}(G, s, t, F)$;

    Compute $\vec{\rho}$ by letting $\rho_e \leftarrow \frac{\widetilde{f}_e}{\min\{u_e^+ - f_e, u_e^- + f_e\}}$;

    $\vec{f}, \vec{y} \leftarrow \text{FIXING}\left(G, \vec{\widehat{f}}, \vec{\widehat{y}}\right)$;

  else

    let S^* be the edge set that contains the $m^{4\eta}$ edges with the largest $|\rho_e|$;

    $G \leftarrow \text{BOOSTING}\left(G, S^*, U, \vec{f}, \vec{y}\right)$;

$\vec{f} \leftarrow \text{FLOWROUNDING}(G, \vec{f}, s, t)$;

while *there is an augmenting path from* s *to* t *with respect to* $\vec{f}$ *for* G **do**

  augment an augmenting path for $\vec{f}$;

Algorithm 17: AUGMENTATION (G, s, t, F)

For each $e \in E$, let $r_e \leftarrow \frac{1}{(u_e^+ - f_e)^2} + \frac{1}{(u_e^- + f_e)^2}$ and $w_e \leftarrow \frac{1}{r_e}$;
Solve Laplacian linear system $\boldsymbol{L}(G)\vec{\widetilde{\phi}} = F \cdot \vec{\chi}_{s,t}$ where $\vec{\chi}_{s,t}$ is the vector whose entry is -1 (resp. 1) at vertex s (resp. t) and 0 otherwise;
For each $e = (u, v) \in E$, let $\widetilde{f_e} \leftarrow \frac{\widetilde{\phi}_v - \widetilde{\phi}_u}{r_e}$ and $\widehat{f_e} \leftarrow f_e + \delta \widetilde{f_e}$;
For each $v \in V$, let $\widehat{y}_v \leftarrow y_v + \delta \widetilde{\phi}_v$;
return $\vec{\widetilde{f}}, \vec{\widehat{f}}, \vec{\widehat{y}}$;

Algorithm 18: FIXING $\left(G, \vec{\widehat{f}}, \vec{\widehat{y}}\right)$

For each $e = (u, v) \in E$, let $r_e \leftarrow \frac{1}{\left(u_e^+ - \widehat{f_e}\right)^2} + \frac{1}{\left(u_e^- + \widehat{f_e}\right)^2}$, $w_e \leftarrow \frac{1}{r_e}$ and $\theta_e \leftarrow w_e \left[(\widehat{y}_v - \widehat{y}_u) - \left(\frac{1}{u_e^+ - \widehat{f_e}} - \frac{1}{u_e^- - \widehat{f_e}} \right) \right]$;
$\vec{f}' \leftarrow \vec{\widehat{f}} + \vec{\theta}$;
Let $\vec{\widehat{\delta}}$ be $\vec{\theta}$'s residue vector;
For each $e \in E$, let $r_e \leftarrow \frac{1}{(u_e^+ - f_e')^2} + \frac{1}{(u_e^- + f_e')^2}$ and $w_e \leftarrow \frac{1}{r_e}$;
Solve Laplacian linear system $\boldsymbol{L}(G)\vec{\phi}' = -\vec{\widehat{\delta}}$;
For each $e = (u, v) \in E$, let $\theta_e' \leftarrow \frac{\phi_v' - \phi_u'}{r_e}$;
For each $e \in E$, $f_e \leftarrow f_e' + \theta_e'$;
For each vertex $v \in V$, $y_v \leftarrow \widehat{y}_v + \phi_v'$;
return $\vec{f}, \vec{y}$;

Algorithm 19: BOOSTING $\left(G, S^*, U, \vec{f}, \vec{y}\right)$

for *each edge* $e = (u, v) \in S^*$ **do**
    $\beta(e) \leftarrow 2 + \left\lceil \frac{2U}{\min\{u_e^+ - f_e, u_e^- + f_e\}} \right\rceil$;
    replace e with path $u \rightsquigarrow v$ that consists of $\beta(e)$ edges $e_1, \cdots, e_{\beta(e)}$ oriented towards v and $\beta(e) + 1$ vertices $v_0 = u, v_1, \cdots, v_{\beta(e)-1}, v_{\beta(e)} = v$;
    $e_1, e_2 \leftarrow e$;
    for $3 \le i \le \beta(e)$, let $u_{e_i}^+ \leftarrow +\infty$ and $u_{e_i}^- \leftarrow \left(\frac{1}{u_e^+ - f_e} - \frac{1}{u_e^- + f_e} \right)^{-1} (\beta(e) - 2) - f_e$;
    for each $1 \le i \le \beta(e)$, let $f_{e_i} \leftarrow f_e$;
    $y_{v_0} \leftarrow y_u$;
    $y_{v_{\beta(e)}} \leftarrow y_v$;
    $y_{v_1} \leftarrow y_v$;
    $y_{v_2} \leftarrow y_v + \frac{1}{u_e^+ - f_e} - \frac{1}{u_e^- + f_e}$;
    for $3 \le i \le \beta(e)$, set $y_{v_3}, \cdots, y_{v_{\beta(e)-1}}$ such that $y_{v_i} - y_{v_{i-1}} = -\frac{1}{\beta(e)-2} \left(\frac{1}{u_e^+ - f_e} - \frac{1}{u_e^- + f_e} \right)$;
Update G;

C.2 Unit Capacity Minimum Cost Flow Algorithm

We give the detailed unit capacity minimum cost flow algorithm (see Algorithm 20) proposed by Cohen et al. [CMSV17] in this subsection. The subroutines Initialization, Perturbation, Progress and Repairing are shown in Algorithm 21, 22, 23 and 24 respectively.

Algorithm 20: MinCostFlow $(G, \vec{\sigma}, W)$

Input: directed graph $G_0 = (V_0, E_0, \vec{c}_0)$ with each edge having unit capacity and cost $\vec{c}_0$; $|V_0| = n$ and $|E_0| = m$; integral demand vector $\vec{\sigma}$; the absolute maximum cost W;

$G = (P \cup Q, E), \vec{b}, \vec{f}, \vec{y}, \vec{s}, \vec{v}, \widehat{\mu}, c_\rho, c_T, \eta \leftarrow$ Initialization$(G_0, \vec{\sigma})$;

Add a new vertex v_0 and undirected edges (v_0, v) for every $v \in P$ to G;

for $i = 1$ **to** $c_T \cdot m^{1/2-3\eta}$ **do**

  for *each* $v \in P$ **do**

    set resistance of edge (v_0, v) for each $v \in P$ to be $r_{v_0 v} \leftarrow \frac{m^{1+2\eta}}{a(v)}$, where $a(v) \leftarrow \sum_{u \in Q, e=(v,u) \in E} \nu_e + \nu_{\overleftarrow{e}}$; ▷ $\overleftarrow{e} = (\overline{v}, u)$ is e's partner edge that is the unique edge sharing one common vertex from Q.

  for $j = 1$ **to** $m^{2\eta}$ **do**

    while $\|\vec{\rho}\|_{\vec{v},3} > c_\rho \cdot m^{1/2-\eta}$ **do**

      $\vec{\rho}, \vec{y}, \vec{s}, \vec{v} \leftarrow$ Perturbation$(G, \vec{\rho}, \vec{f}, \vec{y}, \vec{s}, \vec{v})$;

    $\vec{f}, \vec{s}, \vec{\rho}, \widehat{\mu} \leftarrow$ Progress$(G, \vec{\sigma}, \vec{f}, \vec{v})$;

Repairing $(G, \vec{f}, \vec{y})$;

Algorithm 21: INITIALIZATION $(G, \vec{\sigma})$

Create a new vertex v_{aux} with $\sigma(v_{aux}) = 0$;
for *each* $v \in V_0$ **do**
  $t(v) \leftarrow \sigma(v) + \frac{1}{2}\deg^{G_0}_{in}(v) - \frac{1}{2}\deg^{G_0}_{out}(v)$;
  if $t(v) > 0$ **then** construct $2t(v)$ parallel edges (v, v_{aux}) with costs $\|\vec{c}_0\|_1$;
  else if $t(v) < 0$ **then**
    construct $|2t(v)|$ parallel edges (v_{aux}, v) with costs $\|\vec{c}_0\|_1$;
Let the new graph be $G_1 = (V_1, E_1, \vec{c}_1)$;
Initialize the bipartite graph $G = (P \cup Q, E, \vec{c})$ with $E \leftarrow \emptyset$, $P \leftarrow V_1$ and $Q \leftarrow \{e_{uv} \mid (u, v) \in E_1\}$ where e_{uv} is a vertex corresponding to edge $(u, v) \in E_1$;
for *each* $(u, v) \in E_1$ **do**
  let $E \leftarrow E \cup \{(u, e_{uv}), (v, e_{uv})\}$ with $c(u, e_{uv}) = c_1(u, v)$ and $c(v, e_{uv}) = 0$, and set $b(u) \leftarrow \sigma(u) + \deg^{G_1}_{in}(u)$, $b(v) \leftarrow \sigma(v) + \deg^{G_1}_{in}(v)$ and $b(e_{uv}) \leftarrow 1$;
For each $v \in P$, set $y_v \leftarrow \|\vec{c}\|_\infty$, and for each $v \notin P$, set $y_v \leftarrow 0$;
For each $e = (u, v) \in E$, set $f_e \leftarrow \frac{1}{2}$, $s_e \leftarrow c_e + y_u - y_v$ and $\nu_e \leftarrow \frac{s_e}{2\|\vec{c}\|_\infty}$;
Set $\widehat{\mu} \leftarrow \|\vec{c}\|_\infty$, $c_\rho \leftarrow 400\sqrt{3} \cdot \log^{1/3} W$, $c_T \leftarrow 3c_\rho \log W$ and $\eta \leftarrow \frac{1}{14}$;
return $G, \vec{b}, \vec{f}, \vec{y}, \vec{s}, \vec{\nu}, \widehat{\mu}, c_\rho, c_T$ and η;

Algorithm 22: PERTURBATION $(G, \vec{\rho}, \vec{f}, \vec{y}, \vec{s}, \vec{\nu})$

for *each* $v \in Q$ **do**
  let $e = (u, v)$ and $\overline{e} = (\overline{u}, v)$;
  $y_v \leftarrow y_v - s_e$;
  $\nu_e \leftarrow 2\nu_e$;
  $\nu_{\overline{e}} \leftarrow \nu_{\overline{e}} + \frac{\nu_e f_{\overline{e}}}{f_e}$;

Algorithm 23: PROGRESS $(G, \vec{\sigma}, \vec{f}, \vec{\nu})$

For each $e \in E$, let $r_e \leftarrow \frac{\nu_e}{f_e^2}$;
Solve Laplacian linear system $\boldsymbol{L}(G)\vec{\widehat{\phi}} = \vec{\sigma}$;
For each $e = (u, v) \in E$, let $\widehat{f_e} \leftarrow \frac{\widehat{\phi}_v - \widehat{\phi}_u}{r_e}$ and $\rho_e \leftarrow \frac{|\widehat{f_e}|}{f_e}$;
$\delta \leftarrow \min\left\{\frac{1}{8\|\vec{\rho}\|_{\vec{\nu},4}}, \frac{1}{8}\right\}$;
Update $f'_e \leftarrow (1 - \delta)f_e + \delta\widehat{f_e}$ and $s'_e \leftarrow s_e - \frac{\delta}{1-\delta}(\widehat{\phi}_v - \widehat{\phi}_u)$;
For each $e \in E$, let $f^{\#}_e \leftarrow \frac{(1-\delta)f_e s_e}{s'_e}$;
Obtain the flow vector $\vec{\sigma}'$ corresponding to the residue $\vec{f}' - \vec{f}^{\#}$;
For each $e \in E$, let $r_e \leftarrow \frac{s'^2_e}{(1-\delta)f_e s_e}$;
Solve Laplacian linear system $\boldsymbol{L}(G)\vec{\widetilde{\phi}} = \vec{\sigma}'$;
For each $e = (u, v) \in E$, let $\widetilde{f_e} \leftarrow \frac{\widetilde{\phi}_v - \widetilde{\phi}_u}{r_e}$;
Update $f_e \leftarrow f^{\#}_e + \widetilde{f_e}$ and $s_e \leftarrow s'_e - \frac{s'_e \widetilde{f_e}}{f^{\#}_e}$;

Algorithm 24: REPAIRING $(G, \vec{f}, \vec{y})$

Let $\vec{b}^+$ be the demand vector corresponding to the current flow $\vec{f}$;
For each $v \in P \cup Q$, set $b_v^{\leq} \leftarrow \min(b_v, b_v^+)$;
For each $v \in P \cup Q$, if $f(E(v)) > b_v^{\leq}$, set $\vec{f}$ on $E(v)$ such that $f(E(v)) = b_v^{\leq}$, and let the resulting vector be $\vec{f}^{\leq}$;
Add source s and sink t to G, and connect s to each $v \in P$ with $f_{sv}^{\leq} \leftarrow f^{\leq}(E(v))$, and connect each $v \in Q$ to t with $f_{vt}^{\leq} \leftarrow f^{\leq}(E(v))$ in G;
$\vec{M} \leftarrow$ FLOWROUNDING$(G, \vec{f}^{\leq}, s, t)$;
Remove s, t and related coordinates on $\vec{f}^{\leq}$ and $\vec{M}$ from G;
for $i = 1$ *to* $\widetilde{O}(m^{3/7})$ **do**
    construct graph $\overrightarrow{G}_M = (P \cup Q, E_M, \widetilde{c}_M)$ using G, $\vec{M}$ and $\vec{\widetilde{c}}$ such that for each $e = (u, v) \in E$, $\widetilde{c}_e = c_e - y_u - y_v$ and $E_M = \{(u, v) \in E \mid u \in P, v \in Q\} \cup \{(v, u) \mid u \in Q, v \in P, M_{uv} \neq 0\}$, $\widetilde{c}_M(u, v) = \begin{cases} \widetilde{c}_{uv}, & u \in P, v \in Q \\ -\widetilde{c}_{uv}, & u \in Q, v \in P \end{cases}$;
    set $F_M \leftarrow \{v \in P \cup Q \mid M(v) < b_v\}$;
    compute a shortest path π in $\overrightarrow{G}_M$ from $P \cap F_M$ to $Q \cap F_M$;
    /* $\mathcal{D}_{\overrightarrow{G}_M}(P, u)$ is the distance from P to u in $\overrightarrow{G}_M$ */
    /* Edges that are reachable in $\overrightarrow{G}_M$ from $P \cap F_M$ have non-negative weights $\widetilde{c}_e$ */
    for $u \in P \cup Q$ **do**
        if *u can be reached from P in $\overrightarrow{G}_M$* **then**
            if $u \in P$ **then**
                $y_u \leftarrow y_u - \mathcal{D}_{\overrightarrow{G}_M}(P, u)$;
            else
                $y_u \leftarrow y_u + \mathcal{D}_{\overrightarrow{G}_M}(P, u)$;
    augment $\vec{M}$ using the augmenting path π;
return $\vec{M}$;