Hardness Results for Laplacians of Simplicial Complexes via Sparse-Linear Equation Complete Gadgets

Ming Ding
ETH Zürich, Switzerland

Rasmus Kyng
ETH Zürich, Switzerland

Maximilian Probst Gutenberg
ETH Zürich, Switzerland

Peng Zhang
Rutgers University, Piscataway, NJ, USA

Abstract

We study linear equations in combinatorial Laplacians of k-dimensional simplicial complexes (k-complexes), a natural generalization of graph Laplacians. Combinatorial Laplacians play a crucial role in homology and are a central tool in topology. Beyond this, they have various applications in data analysis and physical modeling problems. It is known that nearly-linear time solvers exist for graph Laplacians. However, nearly-linear time solvers for combinatorial Laplacians are only known for restricted classes of complexes.

This paper shows that linear equations in combinatorial Laplacians of 2-complexes are as hard to solve as general linear equations. More precisely, for any constant $c \geq 1$, if we can solve linear equations in combinatorial Laplacians of 2-complexes up to high accuracy in time $\tilde{O}((\#\text{ of nonzero coefficients})^c)$, then we can solve general linear equations with polynomially bounded integer coefficients and condition numbers up to high accuracy in time $\tilde{O}((\#\text{ of nonzero coefficients})^c)$. We prove this by a nearly-linear time reduction from general linear equations to combinatorial Laplacians of 2-complexes. Our reduction preserves the sparsity of the problem instances up to poly-logarithmic factors.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Mathematics of computing → Computations on matrices; Mathematics of computing → Algebraic topology

Keywords and phrases Simplicial Complexes, Combinatorial Laplacians, Linear Equations, Fine-Grained Complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.53

Category Track A: Algorithms, Complexity and Games

Related Version *Extended Version*: https://arxiv.org/abs/2202.05011 [14]

Funding *Rasmus Kyng*: The research leading to these results has received funding from grant no. 200021 204787 of the Swiss National Science Foundation.
Maximilian Probst Gutenberg: The research leading to these results has received funding from grant no. 200021 204787 of the Swiss National Science Foundation.

1 Introduction

1.1 Simplicial Complexes, Homology, and Combinatorial Laplacians

We study linear equations whose coefficient matrices are combinatorial Laplacians of k-dimensional abstract simplicial complexes (k-complexes), which generalize the well-studied graph Laplacians. An abstract simplicial complex $\mathcal{K}$ is a family of sets, known as simplices, closed under taking subsets, i.e., every subset of a set in $\mathcal{K}$ is also in $\mathcal{K}$. The dimension of $\mathcal{K}$ is the maximum dimension among all simplices in $\mathcal{K}$. A geometric notion of abstract simplicial complexes is simplicial complexes, under which a k-simplex is the convex hull of $k+1$ vertices (for example, 0,1,2-simplices are vertices, edges, and triangles, respectively). In particular, complexes in 1 dimension are graphs; combinatorial Laplacians in 1-complexes are graph Laplacians.

Nearly-linear time solvers exist for linear equations in graph Laplacians [42, 26, 27, 39, 11, 31, 24], and some natural generalizations such as connection Laplacians [29] and directed Laplacians [10, 9]. However, nearly-linear time solvers for linear equations in combinatorial Laplacians are only known for very restricted classes of 2-complexes [8, 3]. We ask whether one can extend these nearly-linear solvers to general combinatorial Laplacians.

Combinatorial Laplacians are defined via boundary operators of the chain spaces of an oriented complex. Given an oriented simplicial complex $\mathcal{K}$, a k-chain is a (signed) weighted sum of the k-simplices in $\mathcal{K}$. The boundary operator ∂_k is a linear map from the k-chain space to the $(k-1)$-chain space; in particular, it maps a k-simplex to a signed sum of its boundary $(k-1)$-simplices, where the signs are determined by the orientations of the k-simplex and its boundary $(k-1)$-simplices. For example, ∂_1 is the oriented vertex-edge incidence matrix. The combinatorial Laplacian $\mathcal{L}_k$ is defined to be $\partial_{k+1}\partial_{k+1}^\top + \partial_k^\top \partial_k$. In particular, $\mathcal{L}_0 = \partial_1\partial_1^\top$ is the graph Laplacian.

Combinatorial Laplacians play an important role in both pure mathematics and applied areas. These matrices originate in the study of discrete Hodge decomposition [18]: The kernel of $\mathcal{L}_k$ is isomorphic to the kth homology space of $\mathcal{K}$. The properties of combinatorial Laplacians have been studied in many works [20, 15, 16, 17, 36]. A central problem in homology theory is evaluating the Betti number of the kth homology space, which equals the rank of $\mathcal{L}_k$. In the case of homology over the reals, computing the rank of $\mathcal{L}_k$ can be reduced to solving a poly-logarithmic number of linear equations in $\mathcal{L}_k$ [2]. Computation of Betti numbers over the reals is a key step in numerous problems in applied topology, computational topology, and topological data analysis [47, 21, 5, 19, 6]. In addition, combinatorial Laplacians have applications in statistical ranking [25, 45], graphics and image processing [35, 44], electromagnetism and fluid mechanics [13], data representations [7], cryo-electron microscopy [46], biology [41]. We refer to the reader to [34] for an accessible survey.

The reader may be puzzled that despite a vast literature on combinatorial Laplacians and their central role in topology, little is known about solving linear equations in these matrices except in very restricted cases [8, 3]. In this paper, we show that approximately solving linear equations in general combinatorial Laplacians is as hard as approximately solving general linear equations over the reals, which explains the lack of special-purpose solvers for this class of equations. More precisely, if one can solve linear equations in combinatorial Laplacians of general 2-complexes to high accuracy in time $\tilde{O}((\#\text{ of nonzero coefficients})^c)$[1] for some constant

[1] $\tilde{O}$ hides poly-logarithmic factors in following parameters of the input: ratio of maximum and minimum non-zero singular values, the maximum ratio of non-zero entries (in absolute value), and the inverse of the accuracy parameter.

$c \geq 1$, then one can solve general linear equations with polynomially bounded integer coefficients and condition numbers up to high accuracy in time $\tilde{O}((\#\text{ of nonzero coefficients})^c)$. A recent breakthrough shows that general linear equations can be solved up to high accuracy in time $\tilde{O}((\#\text{ of nonzero coefficients})^{2.27159})$ [40, 38], which for sparse linear equations is asymptotically faster than the long-standing runtime barrier of fast matrix multiplication [43], which currently achieves a running time of $\tilde{O}(n^{2.3728596})$ [1]. Understanding the optimal value of c is a major open problem in numerical linear algebra. Our result, viewed positively, shows that one can reduce the problem of designing fast solvers for general linear equations to that for combinatorial Laplacians.

1.2 Hardness Results Based on Linear Equations

Kyng and Zhang [33] initiated the study of hardness results for solving structured linear equations. They showed that solving linear equations in a slight generalization of graph Laplacians such as 2-commodity Laplacians, 2-dimensional truss stiffness matrices, and 2-total-variation matrices is as hard as solving general linear equations.

Suppose given an invertible matrix $\boldsymbol{A}$ and a vector $\boldsymbol{b}$ over the reals, we want to approximately solve $\boldsymbol{Ax} = \boldsymbol{b}$, i.e., find $\tilde{\boldsymbol{x}}$ such that $\|\boldsymbol{A}\tilde{\boldsymbol{x}} - \boldsymbol{b}\|_2 \leq \epsilon \|\boldsymbol{b}\|_2$ for some ϵ.

▶ **Definition** ((Informal) Sparse-linear-equation completeness of matrix family $\mathcal{B}$). *Consider a family of matrices $\mathcal{B}$, and suppose that for any instance $(\boldsymbol{A}, \boldsymbol{b}, \epsilon)$ we can produce matrix $\boldsymbol{B} \in \mathcal{B}$, vector $\boldsymbol{c}$, and accuracy parameter δ, such that if we can solve $\boldsymbol{By} = \boldsymbol{c}$ up to error δ, then we can produce $\tilde{\boldsymbol{x}}$ that solves $\boldsymbol{Ax} = \boldsymbol{b}$ to the desired accuracy.*

If, given $(\boldsymbol{A}, \boldsymbol{b}, \epsilon)$, we can compute $(\boldsymbol{B}, \boldsymbol{c}, \delta)$ in $\tilde{O}(\mathrm{nnz}(\boldsymbol{A}))$ time with $\mathrm{nnz}(\boldsymbol{B}) = \tilde{O}(\mathrm{nnz}(\boldsymbol{A}))$ then we say that the class $\mathcal{B}$ is sparse-linear-equation complete*.*

In our preliminaries in Section 2, we state a formal definition of sparse-linear equation completeness that also extends to non-invertible matrices.

The reason of using the term "completeness" is that if a solver with runtime $\tilde{O}((\#\text{ of nonzero coefficients})^c)$ is known for the class $\mathcal{B}$, then a solver with runtime $\tilde{O}((\#\text{ of nonzero coefficients})^c)$ exists for general matrices. Such solvers are known for the classes of Laplacian Matrices, Directed Laplacian Matrices, Connection Laplacian Matrices, etc., all with $c = 1$. Thus, if any of these classes were sparse-linear-equation complete, we would immediately get nearly-linear time solvers for general linear equations.

In this language, Kyng and Zhang [33] showed that 2-commodity Laplacians, 2-dimensional truss stiffness matrices, and 2-total-variation matrices are all sparse-linear-equation complete. We note that [32] considered a larger family of hardness assumptions based on linear equations, which, among other things, can express weaker hardness statements based on weaker reductions.

1.3 Our Contributions

In the terminology established above, our main result can be stated very succinctly:

▶ **Theorem 1.1** (Informal First Main Theorem). *Linear equations in combinatorial Laplacians of 2-complexes are sparse-linear-equation complete.*

In fact, we show this by showing an even simpler problem is sparse-linear-equation complete, namely linear equations in the boundary operator of a 2-complex.

▶ **Theorem 1.2** (Informal Second Main Theorem). *Linear equations in the boundary operators ∂_2 of 2-complexes are sparse-linear-equation complete.*

This result is formally stated in Theorem 3.1. Below, in Section 1.3.1, we sketch how our first main theorem above follows from our second main theorem. We give a formal proof of this in the full version [14].

Our proof establishes the sparse-linear-equation completeness of boundary operators of a 2-complex via a two-step reduction. In our first reduction step, we show sparse-linear-equation completeness of a very simple class of linear equations which we call *difference-average equations*: these are equations where every row either restricts the difference of two variables: $\boldsymbol{x}(i) - \boldsymbol{x}(j) = b$, or it sets one variable to be the average of two others: $\boldsymbol{x}(i) + \boldsymbol{x}(j) = 2\boldsymbol{x}(k)$. This reduction was implicitly proved in [33] as an intermediate step. In this paper, we make the reduction explicit, which may be of independent interest, as this reduction class is likely to be a good starting point for many other hardness reductions. One can think of this step as analogous to showing that 3-SAT is NP-complete: It gives us a simple starting point for proving the hardness of other problems. The formal theorem statement appears in Theorem 2.9. In our second reduction step, we reduce a given difference-average equation problem to a linear equation in the boundary operator of a 2-complex.

Both the two steps preserve the number of nonzero coefficients in the linear equations up to a logarithmic factor, and only blow up the coefficients and condition numbers polynomially. The reductions are also robust to error in the sense that to solve the original problem to high accuracy, it suffices to solve the reduced problem to accuracy at most polynomially higher. Finally, we can compose the two reductions to show that solving linear equations in 2-complex boundary operators to high accuracy is as hard as solving general linear equations with polynomially bounded integer coefficients and condition numbers to high accuracy.

We give more details on both reductions below, but first we describe why solving linear equations in combinatorial Laplacians $\mathcal{L}_1$ is also sparse-linear-equation complete.

1.3.1 Hardness for Combinatorial Laplacians From Hardness for Boundary Operators

Our main technical result, Theorem 3.1, shows that the class of linear equations in the boundary operators of 2-complexes is sparse-linear-equation complete. But, as the following simple lemma shows, we can reduce the problem of solving in a boundary operator ∂_2 to solving in the corresponding combinatorial Laplacian $\mathcal{L}_1$, and hence the latter problem must be at least as hard. This then immediately implies our first main result, Theorem 1.1. The reduction is captured in the following lemma.

▶ **Lemma 1.3** (Informal reduction from boundary operators to combinatorial Laplacians in 2-complexes)**.** *Suppose we can solve linear equations in combinatorial Laplacians of 2-complexes to high accuracy in nearly-linear time. Then, we can solve linear equations in boundary operators ∂_2 of 2-complexes to high accuracy in nearly-linear time.*

The proof is by standard arguments which we sketch below. In the full version of the paper [14], we will formally state the theorem and provide a rigorous proof. Suppose we have a high-accuracy solver for combinatorial Laplacians of 2-complexes. Using this, we want to obtain a solver for linear equations in the boundary operator ∂_2. Note that when the equation $\partial_2\boldsymbol{f} = \boldsymbol{d}$ is infeasible, we measure the solution quality by $\|\partial_2\boldsymbol{f} - \boldsymbol{\Pi}_{\partial_2}\boldsymbol{d}\|_2$ where $\boldsymbol{\Pi}_{\partial_2}$ denotes the orthogonal projection onto the image $\mathrm{im}(\partial_2)$. The minimum over $\boldsymbol{f}$ of the quantity $\|\partial_2\boldsymbol{f} - \boldsymbol{\Pi}_{\partial_2}\boldsymbol{d}\|_2$ is zero, which is obtained by setting $\boldsymbol{f} = \partial_2^{\dagger}\boldsymbol{d}$ (where $\partial_2^{\dagger}$ is the Moore-Penrose pseudo-inverse of ∂_2). The equation $\partial_2\boldsymbol{f} = \boldsymbol{d}$ is feasible exactly when $\boldsymbol{\Pi}_{\partial_2}\boldsymbol{d} = \boldsymbol{d}$.

A central and basic fact in the study of simplicial homology is that $\mathrm{im}(\partial_1^{\top}) \cap \mathrm{im}(\partial_2) = \{\boldsymbol{0}\}$. This follows from $\partial_1\partial_2 = \boldsymbol{0}$ (the boundary of a boundary is zero, which gives $\mathrm{im}(\partial_2) \subseteq \ker(\partial_1)$) and the general fact in linear algebra that $\ker(\boldsymbol{A})$ is the orthogonal complement of $\mathrm{im}(\boldsymbol{A}^{\top})$ so that $\ker(\partial_1)$ is orthogonal to $\mathrm{im}(\partial_1^\top)$. Intuitively, the fact that the boundary of a boundary is zero generalizes that the boundary of a disc is a circular path, and such a path has no endpoints. This implies that $\boldsymbol{\Pi}_{\partial_2}\partial_1^\top = \mathbf{0}$. Now, suppose that $\tilde{\boldsymbol{x}}$ approximately solves $\mathcal{L}_1\boldsymbol{x} = \boldsymbol{d}$, i.e. $\mathcal{L}_1\tilde{\boldsymbol{x}} \approx \boldsymbol{d}$. We can rewrite this as $\partial_1^\top\partial_1\tilde{\boldsymbol{x}} + \partial_2\partial_2^\top\tilde{\boldsymbol{x}} \approx \boldsymbol{d}$. Now, if we apply $\boldsymbol{\Pi}_{\partial_2}$ on both sides, $\boldsymbol{\Pi}_{\partial_2}\boldsymbol{d} \approx \boldsymbol{\Pi}_{\partial_2}\partial_2\partial_2^\top\tilde{\boldsymbol{x}} = \partial_2\partial_2^\top\tilde{\boldsymbol{x}}$. Thus if we set $\tilde{\boldsymbol{f}} = \partial_2^\top\tilde{\boldsymbol{x}}$, then we have $\boldsymbol{\Pi}_{\partial_2}\boldsymbol{d} \approx \partial_2\tilde{\boldsymbol{f}}$, which matches our notion of $\tilde{\boldsymbol{f}}$ approximately solving the (possibly infeasible) linear equation $\partial_2\boldsymbol{f} = \boldsymbol{d}$. This means that if we can approximately solve linear equations in $\mathcal{L}_1$, we can solve linear equations in ∂_2. This way we can also argue that if we can solve linear equations in $\partial_2\partial_2^\top$, then we can solve linear equations in ∂_2. Finally, one should note that $\mathrm{nnz}(\mathcal{L}_1) = O(\mathrm{nnz}(\partial_2))$ and that using our definition of condition number (see Section 2), both have polynomially related condition number[2]. This also means a high accuracy solve in one can be converted to a high accuracy solve in the other.

1.3.2 Linear Equations in $\partial_2\partial_2^\top$

In addition to the many applications discussed in Section 1.1, the problem of solving linear equations in $\partial_2\partial_2^\top$ also arises when using Interior Point Methods to solve a generalized max-flow problem in higher-dimensional simplicial complexes as defined in [36]. We sketch how this inverse problem arises when using an Interior Point Method in the full version of the paper [14]. By a similar argument as Lemma 1.3, we can show that if we can solve linear equations in $\partial_2\partial_2^\top$ to high accuracy in nearly-linear time, then we can solve linear equations in ∂_2 to high accuracy in nearly-linear time.

1.3.3 Sparse-Linear-Equation Completeness of Difference-Average Equations

Our first reduction transforms general linear equations with polynomially bounded integer entries and condition numbers into difference-average equations. We first transform a general linear equation instance to a linear equation instance such that the coefficient matrix has row sum zero and the sum of positive coefficients in each row is a power of 2, by introducing a constant number of more variables and equations. Then, we transform each single equation to a set of difference-average equations by bit-wise pairing and replacing each pair of variables with a new variable via an average equation.

1.3.4 Sparse-Linear-Equation Completeness of Boundary Operators of Simplicial Complexes

Our second reduction transforms difference-average linear equations into linear equations in the boundary operators of 2-complexes. Solving $\partial_2\boldsymbol{f} = \boldsymbol{d}$ can be interpreted as computing a flow $\boldsymbol{f}$ in the triangle space of a 2-complex subject to pre-specified edge demands $\boldsymbol{f}$.

Our reduction is inspired by a reduction in [36] that proves NP-hardness of computing maximum *integral* flows in 2-complexes via a reduction from 3-coloring problems in graphs. However, the correctness of their reduction heavily relies on that the flow values in the 2-complex are 0-1 integers, which does not apply in our setting. In addition, it is unclear how to encode linear equations as a graph coloring problem even if fractional colors are allowed.

We employ some basic building blocks used in [36] including punctured spheres and tubes. However, we need to carefully arrange and orient the triangles in the 2-complex to encode both the positive and negative coefficients in difference-average equations, and we need to express the averaging relations not covered by the previous work.

[2] This is because ∂_1 has polynomially bounded singular values.

An important aspect of our contribution is that we carefully control the number of nonzeros of the boundary operator matrix that we construct, and we bound the condition number of this matrix and how error propagates from an approximate solution to the boundary operator problem back to the original difference-average equations. In order to do so, we develop explicit triangulation algorithms that specify the precise number of triangles needed to triangulate each building block and allow a detailed error and condition number analysis.

We remark that our constructed 2-complex does not admit an embedding into a sphere in 3 dimensions. Recent work [3] has shown that simplicial complexes with a known embedding into $\mathbb{R}^3$ have non-trivial linear equation solvers, but the full extent to which embeddability can lead to better solvers remains an open question.

We analyze our construction in the Real RAM model. However, it can be transferred to the fixed point arithmetic model with $(\log N)^{O(1)}$ bits per number, where N is the size of the problem instance.

1.4 Related Works

Generalized Flows. One can generalize the notions of flows, demands vectors, circulations, and cuts to higher-dimensional simplicial complexes [17, 36]. Recall that in a graph, flows and circulations are defined on a vector space over edges, while demands and cuts are defined on a vector space over vertices. On a connected graph, a demand vector is a vector orthogonal to the all-ones vector, i.e. in the kernel of the boundary operator ∂_0. A flow that routes demand $\boldsymbol{d}$, is a vector $\boldsymbol{f}$ such that $\partial_1 \boldsymbol{f} = \boldsymbol{d}$.

More generally, on a k-complex, we say a demand vector is a vector $\boldsymbol{d}$ on $(k-1)$-simplices with $\boldsymbol{d} \in \ker(\partial_{k-1})$. We say a flow is a vector $\boldsymbol{f}$ on k-simplices, and that the flows $\boldsymbol{f}$ routes demand $\boldsymbol{d}$ if $\partial_k \boldsymbol{f} = \boldsymbol{d}$. Given a demand vector $\boldsymbol{d} \in \ker(\partial_{k-1})$ and a capacity vector $\boldsymbol{c}$ for the k-simplices, a reasonable generalization of the max-flow problem to k-complexes is to compute a flow $\boldsymbol{f}$ satisfying $\partial_k \boldsymbol{f} = \alpha \boldsymbol{d}$ and $\boldsymbol{0} \leq \boldsymbol{f} \leq \boldsymbol{c}$ to maximize the flow value α.

Solving Linear Equations. Linear equations are ubiquitous in computational mathematics, computer science, engineering, physics, biology, and economics. Currently, the best known algorithm for solving general dense linear equations in dimensions $n \times n$ runs in time $\tilde{O}(n^{\omega})$, where $\omega < 2.3728596$ is the matrix multiplication constant [1]. For sparse linear equations with N nonzero coefficients and condition number κ, the best known approximate algorithms run in time $\widetilde{O}(\min\{N^{2.27159}, N\kappa\})$, where the first runtime is from [40, 38] and the second is by the conjugate gradient[3] [23].

In contrast to general linear equations, linear equations in graph Laplacians and its generalizations can be solved asymptotically faster, as mentioned earlier. In addition, faster solvers are also known for restricted classes of total-variation matrices [28], stiffness matrices from elliptic finite element systems [4], and 2 and 3-dimensional truss stiffness matrices [12, 30]. An interesting open question is to what extent one can generalize these faster solvers to more classes of matrices.

Reduction From Sparse Linear Equations. [32] defines a parameterized family of hypotheses for runtime of solving sparse linear equations. Under these hypotheses, they prove hardness of approximately solving packing and covering linear programs. For example, if one can solve a packing linear program up to ϵ accuracy in time $\widetilde{O}(\#\text{ of nonzero coefficients} \times \epsilon^{-0.165})$, then one can solve linear equations in time asymptotically faster than $\widetilde{O}(\#\text{ of nonzero coefficients} \times$ condition number of matrix), which is the runtime of conjugate gradient.

[3] If the coefficient matrix is symmetric positive semidefinite, the runtime is $\tilde{O}(N\sqrt{\kappa})$.

2 Preliminaries

2.1 Simplicial Homology

We define the basic concepts of simplicial homology. We recommend the reader the books [37] and [22] for a more complete treatment.

Simplicial Complexes. A *k-dimensional simplex* (or k-simplex) $\sigma = \text{conv}\{v_0, \dots, v_k\}$ is the convex hull of $k+1$ affinely independent points $v_0, \dots, v_k$. For example, 0,1,2-simplices are vertices, edges, and triangles, respectively. A *face* of σ is the convex hull of a non-empty subset of $\{v_0, v_1, \dots, v_k\}$. An *orientation* of σ is given by an ordering Π of its vertices, written as $\sigma = [v_{\Pi(0)}, \dots, v_{\Pi(k)}]$, such that two orderings define the same orientation if and only if they differ by an even permutation. If Π is even, then $[v_{\Pi(0)}, \dots, v_{\Pi(k)}] = [v_0, \dots, v_k]$; if Π is odd, then $[v_{\Pi(0)}, \dots, v_{\Pi(k)}] = -[v_0, \dots, v_k]$.

A *simplicial complex* $\mathcal{K}$ is a finite collection of simplices such that (1) for every $\sigma \in \mathcal{K}$ if $\tau \subset \sigma$ then $\tau \in \mathcal{K}$ and (2) for every $\sigma_1, \sigma_2 \in \mathcal{K}$, $\sigma_1 \cap \sigma_2$ is either empty or a face of both σ_1, σ_2. The *dimensions* of $\mathcal{K}$ is the maximum dimension of any simplex in $\mathcal{K}$. We refer to a simplicial complex in k dimensions as a k-complex.

Boundary Operators. A k-chain is a formal sum of the oriented k-simplices in $\mathcal{K}$ with the coefficients over $\mathbb{R}$. Let $C_k(\mathcal{K})$ denote the kth chain space. The *boundary operator* is a linear map $\partial_k : C_k(\mathcal{K}) \to C_{k-1}(\mathcal{K})$ such that for an oriented k-simplex $\sigma = [v_0, v_1, \dots, v_k]$,

$$\partial_k(\sigma) = \sum_{i=0}^{k} (-1)^i [v_0, \dots, \hat{v}_i, \dots, v_k],$$

where $[v_0, \dots, \hat{v}_i, \dots, v_k]$ is the oriented $(k-1)$-simplex obtained by removing v_i from σ, and $(-1)^i$ is its *induced orientation*. The operator ∂_k can be written as a matrix in $n_{k-1} \times n_k$ dimensions, where n_d is the number of d-simplices in $\mathcal{K}$. The (i, j)th entry of ∂_k is ± 1 if the ith $(k-1)$-simplex is a face of the jth k-simplex where the sign is determined by the orientations, and 0 otherwise. See Figure 1 and Eq. (1) for an example.

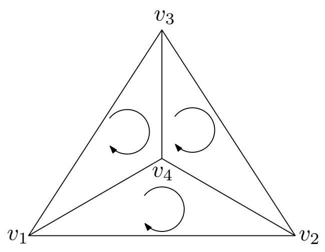

Figure 1 An example of boundary operator and oriented triangulation. We orient 2-simplices clockwise, and orient 1-simplices by the increasing order of vertex indices.

$$\partial_2 = \begin{array}{c} \\ [v_1,v_2] \\ [v_2,v_3] \\ [v_1,v_3] \\ [v_1,v_4] \\ [v_2,v_4] \\ [v_3,v_4] \end{array} \begin{array}{c} \begin{array}{ccc} [v_1,v_4,v_2] & [v_2,v_4,v_3] & [v_1,v_3,v_4] \end{array} \\ \left[\begin{array}{ccc} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \end{array}\right] \end{array}. \qquad (1)$$

An important property of the boundary operator is that applying the boundary operator twice results in the zero operator, i.e.,

$$\partial_{k-1}\partial_k = \mathbf{0}. \tag{2}$$

This implies $\mathrm{im}(\partial_k) \subseteq \ker(\partial_{k-1})$. Thus, we can define the quotient space $H_k = \ker(\partial_k) \setminus \mathrm{im}(\partial_{k+1})$, referred to as the kth homology space of $\mathcal{K}$. The dimension of H_k is the kth Betti number of $\mathcal{K}$, which plays an important role in understanding the homology spaces.

Hodge Theory and Combinatorial Laplacians. Combinatorial Laplacians arise from the discrete Hodge decomposition.

▶ **Theorem 2.1** (Hodge decomposition [34]). *Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{B} \in \mathbb{R}^{n\times p}$ be matrices satisfying $\boldsymbol{AB} = \mathbf{0}$. Then, there is an orthogonal direct sum decomposition*

$$\mathbb{R}^n = \mathrm{im}(\boldsymbol{A}^\top) \oplus \ker(\boldsymbol{A}^\top\boldsymbol{A} + \boldsymbol{BB}^\top) \oplus \mathrm{im}(\boldsymbol{B}).$$

By Eq. (2), it is valid to set $\boldsymbol{A} = \partial_k$ and $\boldsymbol{B} = \partial_{k+1}$. The matrix we get in the middle term is the *combinatorial Laplacian*: $\mathcal{L}_k \overset{\text{def}}{=} \partial_k^\top\partial_k + \partial_{k+1}\partial_{k+1}^\top$. In particular, $\mathcal{L}_0 = \partial_1\partial_1^\top$ is the graph Laplacian. The kth homology space $H_k(\mathcal{K})$ is isomorphic to $\ker(\mathcal{L}_k)$, and thus the kth Betti number of $\mathcal{K}$ equals the dimension of $\ker(\mathcal{L}_k)$.

Triangulation. A *triangulation* of a topological space $\mathcal{X}$ is a simplicial complex $\mathcal{K}$ together with a homeomorphism between $\mathcal{X}$ and $\mathcal{K}$. In this paper, the only topological spaces that we compute triangulations of are 2-dimensional manifolds. A 2-dimensional manifold can be triangulated by a 2-complex, where every edge in the 2-complex is contained in exactly one triangle (boundary edge) or two triangles (interior edge). An *oriented triangulation* of a 2-dimensional manifold is a triangulation together with an orientation for each triangle such that any two neighboring triangles induce opposite signs on their shared interior edge.

Figure 1 is an example of (oriented) triangulation: the topological space is a disk; boundary edges are $[v_1, v_2], [v_2, v_3], [v_1, v_3]$; interior edges are $[v_1, v_4], [v_2, v_4], [v_3, v_4]$; the orientation of each triangle is clockwise.

2.2 Notation for Matrices and Vectors

We use parentheses to denote entries of a matrix or a vector: Let $\boldsymbol{A}(i,j)$ bet the (i,j)th entry of a matrix $\boldsymbol{A}$, and let $\boldsymbol{x}(i)$ bet the ith entry of a vector $\boldsymbol{x}$. We use $\mathbf{1}_n, \mathbf{0}_n$ to denote n-dimensional all-one vector and all-zero vector, respectively. We define $\|\boldsymbol{x}\|_{\max} = \max_{i\in[n]} |\boldsymbol{x}(i)|$, $\|\boldsymbol{x}\|_1 = \sum_{i\in[n]} |\boldsymbol{x}(i)|$. Given a matrix $\boldsymbol{A} \in \mathbb{R}^{d\times n}$, we use $\boldsymbol{A}(i)$ to denote the ith row of $\boldsymbol{A}$ and $\mathrm{nnz}(\boldsymbol{A})$ the number of nonzero entries of $\boldsymbol{A}$. Without loss of generality, we assume that $\mathrm{nnz}(\boldsymbol{A}) \geq \max\{d, n\}$. We let $\|\boldsymbol{A}\|_{\max} = \max_{i,j} |\boldsymbol{A}(i,j)|$. We use $\mathrm{im}(\boldsymbol{A})$ to denote the image (i.e., the column space) of $\boldsymbol{A}$ and $\mathrm{null}(\boldsymbol{A})$ the null space of $\boldsymbol{A}$. We let $\boldsymbol{\Pi}_{\boldsymbol{A}} = \boldsymbol{A}(\boldsymbol{AA}^\top)^\dagger\boldsymbol{A}^\top$ be the orthogonal projection onto $\mathrm{im}(\boldsymbol{A})$, where $\boldsymbol{M}^\dagger$ is the pseudo-inverse of $\boldsymbol{M}$. Let $\lambda_{\max}(\boldsymbol{A})$ be the maximum eigenvalue of $\boldsymbol{A}$ and $\lambda_{\min}(\boldsymbol{A})$ the minimum *nonzero* eigenvalue of $\boldsymbol{A}$. Similarly, let $\sigma_{\max}(\boldsymbol{A})$ be the maximum eigenvalue of $\boldsymbol{A}$ and $\sigma_{\min}(\boldsymbol{A})$ the minimum *nonzero* singular value of $\boldsymbol{A}$. The condition number of $\boldsymbol{A}$, denoted by $\kappa(\boldsymbol{A})$, is the ratio of the maximum to the minimum *nonzero* singular value of $\boldsymbol{A}$.

We define a function U that takes a matrix $\boldsymbol{A}$ and a vector $\boldsymbol{b}$ as arguments and returns the maximum of $\|\cdot\|_{\max}$ of all the arguments, that is, $U(\boldsymbol{A}, \boldsymbol{b}) = \max\{\|\boldsymbol{A}\|_{\max}, \|\boldsymbol{b}\|_{\max}\}$.

2.3 Systems of Linear Equations

We define approximately solving linear equations in a general form, following [33]. For more details, we refer the readers to Section 2.1 of [33].

▶ **Definition 2.2** (Linear Equation Problem (LE)). *Given a matrix $\boldsymbol{A} \in \mathbb{R}^{d\times n}$, a vector $\boldsymbol{b} \in \mathbb{R}^d$, we refer to the linear equation problem for the tuple $(\boldsymbol{A}, \boldsymbol{b})$, denoted by* LE $(\boldsymbol{A}, \boldsymbol{b})$*, as the problem of finding an $\boldsymbol{x} \in \mathbb{R}^n$ such that $\boldsymbol{x} \in \arg\min_{\boldsymbol{x}\in\mathbb{R}^n} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2$.*

▶ **Fact 2.3.** *Let $\boldsymbol{x}^* \in \arg\min_{\boldsymbol{x}\in\mathbb{R}^n} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2$. Then,*

$$\boldsymbol{A}\boldsymbol{x}^* = \boldsymbol{A}(\boldsymbol{A}^\top\boldsymbol{A})^\dagger\boldsymbol{A}^\top\boldsymbol{b} = \boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{b} \quad \text{and} \quad \|\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{b}\|_2^2 = \|(\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{A}})\boldsymbol{b}\|_2^2.$$

By the above fact, solving LE $(\boldsymbol{A}, \boldsymbol{b})$ is equivalent to finding an $\boldsymbol{x}$ such that $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{b}$. This equation is known as the normal equation, and it is always feasible. If $\boldsymbol{b} \in \mathrm{im}(\boldsymbol{A})$, then $\boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{b} = \boldsymbol{b}$.

In practice, we are more interested in *approximately* solving linear equations, since numerical errors are unavoidably in data collection and computation and approximate solvers may run faster.

▶ **Definition 2.4** (Linear Equation Approximation Problem (LEA)). *Given a matrix $\boldsymbol{A} \in \mathbb{R}^{d\times n}$, vectors $\boldsymbol{b} \in \mathbb{R}^d$, and an error parameter $\epsilon \in (0, 1]$, we refer to linear equation approximate problem for the tuple $(\boldsymbol{A}, \boldsymbol{b}, \epsilon)$, denoted by* LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon)$*, as the problem of finding an $\boldsymbol{x} \in \mathbb{R}^n$ such that $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{b}\|_2 \le \epsilon \|\boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{b}\|_2$.*

▶ **Fact 2.5.** *Let $\boldsymbol{x}$ be a solution to* LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon)$*. Then,*

$$\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 \le \|\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{b}\|_2^2 + \epsilon^2 \|\boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{b}\|_2^2.$$

The definition of the approximate error in Definition 2.4 is equivalent to several error notions that are commonly used in solving linear equations. In particular,

$$\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{b}\|_2 = \left\|\boldsymbol{A}^\top\boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^\top\boldsymbol{b}\right\|_{(\boldsymbol{A}^\top\boldsymbol{A})^\dagger} = \|\boldsymbol{x} - \boldsymbol{x}^*\|_{\boldsymbol{A}^\top\boldsymbol{A}}.$$

2.3.1 Matrix Classes

We are interested in linear equations whose coefficient matrices belonging to the following matrix classes.

1. $\mathcal{G}$ refers to the class of *General Matrices* that have integer entries and do not have all-0 rows and all-0 columns. We refer to linear equations whose coefficient matrix is in $\mathcal{G}$ as *general linear equations.*
2. $\mathcal{DA}$ refers to the class of *Difference-Average Matrices* whose rows fall into two categories:
   a. A *difference row* which has exactly two nonzero entries 1 and -1;
   b. An *average row* which has exactly three nonzero entries 1, 1, and -2.

   Multiplying a difference row vector to a column vector $\boldsymbol{x}$ gives $\boldsymbol{x}(i) - \boldsymbol{x}(j)$; multiplying an average row vector to $\boldsymbol{x}$ gives $\boldsymbol{x}(i) + \boldsymbol{x}(j) - 2\boldsymbol{x}(k)$. We refer to linear equations whose coefficient matrix is in $\mathcal{DA}$ as *difference-average linear equations.*
3. $\mathcal{B}_2$ refers to the class of *Boundary Operator Matrices ∂_2 in 2-complexes.* We refer to linear equations whose coefficient matrix is in $\mathcal{B}_2$ as *2-complex boundary linear equations.*

Our definition of "general matrices" specifies the matrix must have integer entries. However, when the input matrix is invertible, using a simple rounding argument, we can convert any linear equation into an linear equation with integer entries $\widetilde{O}(1)$ bits per entry. We caution the reader this relies on our definition of $\widetilde{O}(\cdot)$ as hiding polylogarithmic factors in the input condition number. In general, the condition number can be exponentially large – however, our results are mainly of interest when the condition number is quasipolynomially bounded.

2.3.2 Reduction Between Linear Equations

We will again follow the definition of efficient reductions in [33]. We say LEA over matrix class $\mathcal{M}_1$ is *nearly-linear time reducible* to LEA over matrix class $\mathcal{M}_2$, denoted by $\mathcal{M}_1 \leq_{nlt} \mathcal{M}_2$, if the following holds:

1. There is an algorithm that maps an arbitrary instance LEA $(\boldsymbol{M}_1, \boldsymbol{c}_1, \epsilon_1)$ where $\boldsymbol{M}_1 \in \mathcal{M}_1$ to an instance LEA $(\boldsymbol{M}_2, \boldsymbol{c}_2, \epsilon_2)$ where $\boldsymbol{M}_2 \in \mathcal{M}_2$ such that there is another algorithm that can map a solution to LEA $(\boldsymbol{M}_2, \boldsymbol{c}_2, \epsilon_2)$ to a solution to LEA $(\boldsymbol{M}_1, \boldsymbol{c}_1, \epsilon_1)$.
2. Both the two algorithms run in time $\widetilde{O}(\mathrm{nnz}(\boldsymbol{M}_1))$.
3. In addition, we can guarantee $\mathrm{nnz}(\boldsymbol{M}_2) = \widetilde{O}(\mathrm{nnz}(\boldsymbol{M}_1))$, and $\epsilon_2^{-1}, \kappa(\boldsymbol{M}_2), U(\boldsymbol{M}_2, \boldsymbol{b}_2) = \mathrm{poly}(\mathrm{nnz}(\boldsymbol{M}_1), \epsilon_1^{-1}, \kappa(\boldsymbol{M}_1), U(\boldsymbol{M}_1, \boldsymbol{b}_1))$.

We do not require a nearly-linear time reduction to preserve the number of variables or constraints (dimensions) of linear equations. The dimensions of the new linear equation instance that we construct can be much larger than that of the original instance. On the other hand, a reduction that *only* preserves dimensions may construct a dense linear equation instance even if the original instance is sparse. A nearly-linear time reduction that preserves *both* the number of nonzeros and dimensions would be stronger than what we achieve.

▶ **Fact 2.6.** *If $\mathcal{M}_1 \leq_{nlt} \mathcal{M}_2$ and $\mathcal{M}_2 \leq_{nlt} \mathcal{M}_3$, then $\mathcal{M}_1 \leq_{nlt} \mathcal{M}_3$.*

▶ **Definition 2.7** (Sparse linear equation complete (SLE-complete)). *We say LEA over a matrix class $\mathcal{M}$ is* sparse-linear-equation-complete *if $\mathcal{G} \leq_{nlt} \mathcal{M}$.*

▶ **Fact 2.8.** *Suppose LEA over $\mathcal{M}$ is SLE-complete. If one can solve all instances LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon)$ with $\boldsymbol{A} \in \mathcal{M}$ in time $\tilde{O}(\mathrm{nnz}(\boldsymbol{A})^c)$ where $c \geq 1$, then one can solve all instances LEA $(\boldsymbol{A}', \boldsymbol{b}', \epsilon')$ with $\boldsymbol{A}' \in \mathcal{G}$ in time $\tilde{O}(\mathrm{nnz}(\boldsymbol{A}')^c)$.*

Under the above definitions, [33] implicitly shows the following results. We provide an explicit and simplified proof in the full version of the paper [14].

▶ **Theorem 2.9** (Implicitly stated in [33]). *LEA over $\mathcal{DA}$ is SLE-complete.*

3 Main Results

Our main result is stated in the following theorem.

▶ **Theorem 3.1.** *LEA over $\mathcal{B}_2$ is SLE-complete.*

Theorem 3.1 states that a linear equation approximation problem over a boundary operator of a 2-complex is sparse linear equation complete. See Section 2 for detailed definitions.

Although our main theorem focuses on linear equation approximate problems, we construct nearly-linear time reductions for both linear equation problem LE and its approximate counterpart LEA. We first reduce LE instances $(\boldsymbol{A}, \boldsymbol{b})$ (and LEA instances $(\boldsymbol{A}, \boldsymbol{b}, \epsilon)$) over difference-average matrices to those over 2-complex boundary operator matrices, *under the assumption* $\boldsymbol{b} \in \mathrm{im}(\boldsymbol{A})$ (stated in Theorem 3.2 and 3.3). In this case, the constructed 2-complexes have unit edge weights. We then provide a slightly modified nearly-linear time reduction for LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon)$ over difference-average matrices to LEA over 2-complex boundary operator matrices *without assuming* $\boldsymbol{b} \in \mathrm{im}(\boldsymbol{A})$ (stated in Theorem 3.4). In this case, we introduce polynomially bounded edge weights for the constructed 2-complexes.

▶ **Theorem 3.2.** *Given a linear equation instance LE $(\boldsymbol{A}, \boldsymbol{b})$ where $\boldsymbol{A} \in \mathcal{DA}$ and $\boldsymbol{b} \in \mathrm{im}(\boldsymbol{A})$, we can reduce it to an instance LE (∂_2, γ) where $\partial_2 \in \mathcal{B}_2$, in time $O(\mathrm{nnz}(\boldsymbol{A}))$, such that a solution to LE (∂_2, γ) can be mapped to a solution to LE $(\boldsymbol{A}, \boldsymbol{b})$ in time $O(\mathrm{nnz}(\boldsymbol{A}))$.*

▶ **Theorem 3.3.** *Given a linear equation instance* LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon^{DA})$ *where* $\boldsymbol{A} \in \mathcal{DA}$ *and* $\boldsymbol{b} \in \operatorname{im}(\boldsymbol{A})$*, we can reduce it to an instance* LEA $(\partial_2, \gamma, \epsilon^{B_2})$ *where* $\partial_2 \in \mathcal{B}_2$ *and* $\epsilon^{B_2} \le \frac{\epsilon^{DA}}{42\,\mathrm{nnz}(\boldsymbol{A})}$*, in time* $O(\mathrm{nnz}(\boldsymbol{A}))$*, such that a solution to* LEA $(\partial_2, \gamma, \epsilon^{B_2})$ *can be mapped to a solution to* LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon^{DA})$ *in time* $O(\mathrm{nnz}(\boldsymbol{A}))$*.*

▶ **Theorem 3.4.** *Given an instance* LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon^{DA})$ *where* $\boldsymbol{A} \in \mathcal{DA}$*, we can reduce it to an instance* LEA $(\boldsymbol{W}^{1/2}\partial_2, \boldsymbol{W}^{1/2}\gamma, \epsilon^{B_2})$ *where* $\partial_2 \in \mathcal{B}_2$ *and* $\boldsymbol{W}$ *is a diagonal matrix with nonnegative diagonals, in time* $O(\mathrm{nnz}(\boldsymbol{A}))$*. Let* s, ϵ, K, U *denote* $\mathrm{nnz}(\boldsymbol{A}), \epsilon^{DA}, \kappa(\boldsymbol{A}), U(\boldsymbol{A}, \boldsymbol{b})$*, respectively. Then, we can guarantee that*

$$\mathrm{nnz}(\partial_2) = O(s),\ U(\boldsymbol{W}^{1/2}\partial_2, \boldsymbol{W}^{1/2}\gamma) = O\left(sU\epsilon^{-1}\right),$$
$$\epsilon^{B_2} = \Omega(\epsilon U^{-1} s^{-1}),\ \kappa(\boldsymbol{W}^{1/2}\partial_2) = O\left(s^{15/2}K^2\epsilon^{-2}\right)$$

and a solution to LEA $(\boldsymbol{W}^{1/2}\partial_2, \boldsymbol{W}^{1/2}\gamma, \epsilon^{B_2})$ *can be mapped to a solution to* LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon^{DA})$ *in time* $O(\mathrm{nnz}(\boldsymbol{A}))$*.*

We refer the reader to the full version of the paper [14] for a formal proof of Theorem 3.2, 3.3, and 3.4.

3.1 Overview of Our Proof

Multiplying a 2-complex boundary operator $\partial_2 \in \mathbb{R}^{m\times t}$ to a vector $\boldsymbol{f} \in \mathbb{R}^t$ can be interpreted as transforming flows in the triangle space to demands in the edge space. Given $\gamma \in \mathbb{R}^m$, solving LE (∂_2, γ) can be interpreted as finding flows $\boldsymbol{f}$ in the triangle space subject to edge demands γ. We will encode difference-average linear equations as a 2-complex flow network.

Encoding Linear Operations. In difference-average linear equations, linear operations include subtraction, addition, and multiplication. We observe a simple fact: If we glue two triangles Δ_1, Δ_2 with the same orientation, then the net flow $\partial_2 \boldsymbol{f}$ on the shared edge is $\boldsymbol{f}(\Delta_1) - \boldsymbol{f}(\Delta_2)$ (see Figure 2 (a)); if we glue two triangles Δ_1, Δ_2 with opposite orientations, then the net flow $\partial_2 \boldsymbol{f}$ on the shared edge is $\boldsymbol{f}(\Delta_1) + \boldsymbol{f}(\Delta_2)$ (see Figure 2 (b)). Given an equation $\boldsymbol{a}^\top \boldsymbol{x} = b$ with the nonzero coefficients being ± 1, we can encode it by gluing more triangles as above and setting the demand of the shared edge to be b. To handle the coefficient -2 in an average equation, say $\boldsymbol{x}(i) + \boldsymbol{x}(j) - 2\boldsymbol{x}(k)$, we implicitly interpret it as $\boldsymbol{x}(i) + \boldsymbol{x}(j) - \boldsymbol{x}(k_1) - \boldsymbol{x}(k_2)$ together with an additional difference equation $\boldsymbol{x}(k_1) = \boldsymbol{x}(k_2)$ (see Figure 2 (c)).

Encoding a Variable. We use a sphere to encode a variable involved in many equations. We can obtain an oriented triangulation of the sphere and set all the edge demand to be 0 so that all the triangles on the sphere must have an equal flow value (see Figure 2 (d)).

Putting It All Together. We represent each variable using a triangulated sphere. To add a constraint between variables, in each corresponding sphere we add a "hole" if the variable coefficient is ± 1 and add two "holes" if the coefficient is 2. Then we attach a "tube" to each of these holes. Similarly, we can obtain an oriented triangulation of each tube and set the edge demand properly so that all the triangles on the tube must have the same flow value to the triangles on the sphere that the tube is attached to. We then connect these tubes corresponding to different variables so that they share common edges. Depending on how the tubes connect and on the net flow demand on the shared edges, we can represent different linear constraints on the variables.

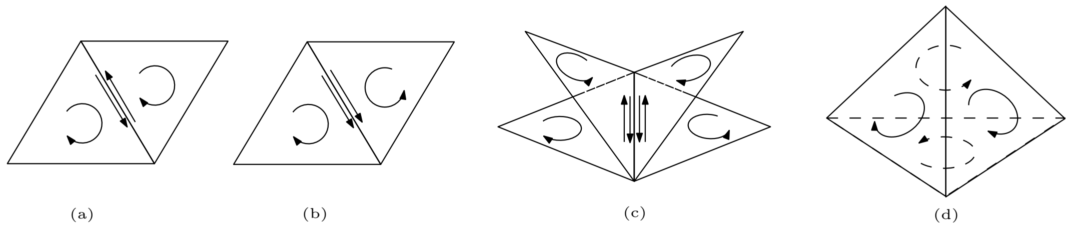

Figure 2 An illustration for encoding linear operations and variables: (a) encodes the "subtraction" with two identically oriented triangles. (b) encodes the "addition" with two oppositely oriented triangles. (c) encodes the "multiplication" by a coefficient 2 in an average equation. (d) encodes a variable with an identically oriented triangulated sphere, and this will later allow us to use several "copies" of this variable, by making holes in this sphere and attaching a tube to each such hole.

Discussion.

1. *Why encode difference/average equations rather than directly encoding general equations with integer coefficients?*

   We can generalize the above encoding method to encode a general equation $\boldsymbol{g}^\top \boldsymbol{y} = c$ with arbitrary integer coefficients into a 2-complex with roughly $\|\boldsymbol{g}\|_1$ tubes. However, the encoding size required to express a general system of linear equations $\boldsymbol{G}\boldsymbol{y} = \boldsymbol{c}$ this way can be as large as $\Omega(\text{nnz}(\boldsymbol{G}) \|\boldsymbol{G}\|_{\max})$. This dependence on $\|\boldsymbol{G}\|_{\max}$ is prohibitive, and makes for a fairly weak result.

   On the other hand, we can first reduce the general linear equations $\boldsymbol{G}\boldsymbol{y} = \boldsymbol{c}$ into difference-average linear equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where $\|\boldsymbol{A}\|_{\max} = 2$ and $\text{nnz}(\boldsymbol{A}) = O(\text{nnz}(\boldsymbol{G}) \log \|\boldsymbol{G}\|_{\max})$ (by Lemma A.1 in [14]). Then we can encode $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ into a 2-complex. The encoding size required to express the the difference-average linear equations as a 2-complex is thus $O(\text{nnz}(\boldsymbol{A}))$ (by Lemma 4.2). Thus, the overall encoding size required to express the original linear equation $\boldsymbol{G}\boldsymbol{y} = \boldsymbol{c}$ is now $\widetilde{O}(\text{nnz}(\boldsymbol{G}))$, exponentially improving the dependence on $\|\boldsymbol{G}\|_{\max}$.

   Therefore, the two-step reduction is a nearly-linear time reduction while the one-step reduction is not.

2. *Why encode into a 2-complex rather than a 1-complex?*

   We do not expect that general linear equations with integer coefficients can be efficiently encoded using a 1-complex. This would immediately imply a nearly-linear time solver for general linear equations, as fast solvers for 1-complex operators exist (using Laplacian linear equation solvers).

4 Reducing Exact Solvers for $\mathcal{DA}$ to $\mathcal{B}_2$ Assuming the Right-Hand Side Vector in the Image of the Coefficient Matrix

In this section, we describe a nearly-linear time reduction from instances LE $(\boldsymbol{A}, \boldsymbol{b})$ over $\mathcal{DA}$ to instances LE over $\mathcal{B}_2$, under the assumption that $\boldsymbol{b} \in \text{im}(\boldsymbol{A})$, and analyze its runtime and the size of the reduced problem. In the full version of the paper [14], we show that the same reduction with a carefully chosen error parameter reduces linear equation approximate problem LEA over $\mathcal{DA}$ to LEA over $\mathcal{B}_2$, assuming $\boldsymbol{b} \in \text{im}(\boldsymbol{A})$. In addition, we slightly modify the reduction algorithm to drop the assumption $\boldsymbol{b} \in \text{im}(\boldsymbol{A})$ for LEA.

4.1 Reduction Algorithm

Recall that an instance LE $(\boldsymbol{A}, \boldsymbol{b})$ over $\mathcal{DA}$ only consists of two types of linear equations: (1) *Difference equation*: $\boldsymbol{x}(i) - \boldsymbol{x}(j) = \boldsymbol{b}(q)$; (2) *Average equation*: $\boldsymbol{x}(i) + \boldsymbol{x}(j) - 2\boldsymbol{x}(k) = 0$. Suppose LE $(\boldsymbol{A}, \boldsymbol{b})$ has d_1 difference equations and d_2 average equations. Without loss of generality, we reorder all the equations so that the first d_1 equations are difference equations and the rest are average equations. The following algorithm ReduceDATo$\mathcal{B}_2$ constructs a 2-complex and a system of linear equations in its boundary operator.

Algorithm Reduce$\mathcal{DA}$To$\mathcal{B}_2$.
Input: an instance LE $(\boldsymbol{A}, \boldsymbol{b})$ where $\boldsymbol{A} \in \mathcal{DA}$ is a $d \times n$ matrix and $\boldsymbol{b} \in \mathbb{R}^d$.
Output: $(\partial_2, \gamma, \boldsymbol{\Delta}^c)$ where $\partial_2 \in \mathcal{B}_2$ is an $m \times t$ matrix, $\gamma \in \mathbb{R}^m$, and $\boldsymbol{\Delta}^c$ is a set of n triangles.

1. For each $i \in [n]$ and variable $\boldsymbol{x}(i)$ in LE $(\boldsymbol{A}, \boldsymbol{b})$, we construct a sphere $\mathcal{S}_i$.
2. For each $q \in [d_1]$, which corresponds to a difference equation $\boldsymbol{x}(i) - \boldsymbol{x}(j) = \boldsymbol{b}(q)$, we add a *loop*[4] α_q with a net flow demand $\boldsymbol{b}(q)$. Then,
   a. we add a *boundary component*[5] $\beta_{q,i}$ on $\mathcal{S}_i$, and a boundary component $\beta_{q,j}$ on $\mathcal{S}_j$;
   b. we construct a tube $\mathcal{T}_{q,i}$ with boundary components $\{-\beta_{q,i}, \alpha_q\}$, and a tube $\mathcal{T}_{q,j}$ with boundary components $\{-\beta_{q,j}, -\alpha_q\}$.

   See Figure 3 for an illustration[6].

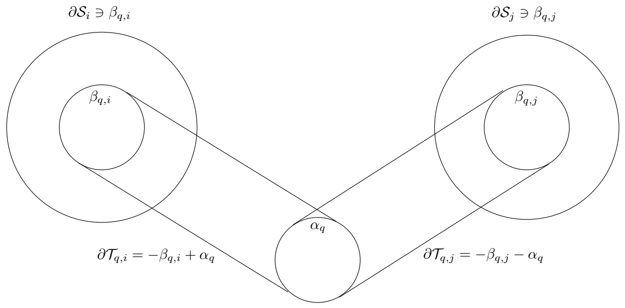
$\partial\mathcal{S}_i \ni \beta_{q,i}$
$\partial\mathcal{S}_j \ni \beta_{q,j}$
$\beta_{q,i}$
$\beta_{q,j}$
α_q
$\partial\mathcal{T}_{q,i} = -\beta_{q,i} + \alpha_q$
$\partial\mathcal{T}_{q,j} = -\beta_{q,j} - \alpha_q$

Figure 3 The construction for a difference equation $\boldsymbol{x}(i) - \boldsymbol{x}(j) = \boldsymbol{b}(q)$. For a topological space X, we use ∂X to denote its boundary.

3. For each $q \in \{d_1+1, \ldots, d\}$, which corresponds to an average equation $\boldsymbol{x}(i)+\boldsymbol{x}(j)-2\boldsymbol{x}(k) = \boldsymbol{b}(q) = 0$, we add a loop α_q with zero net flow demand. Then,
   a. we add a boundary component $\beta_{q,i}$ on $\mathcal{S}_i$, a boundary component $\beta_{q,j}$ on $\mathcal{S}_j$, and two boundary components $\beta_{q,k,1}, \beta_{q,k,2}$ on $\mathcal{S}_k$;
   b. we construct a tube $\mathcal{T}_{q,i}$ with boundary components $\{-\beta_{q,i}, \alpha_q\}$, a tube $\mathcal{T}_{q,j}$ with boundary components $\{-\beta_{q,j}, \alpha_q\}$, and two tubes $\mathcal{T}_{q,k,1}, \mathcal{T}_{q,k,2}$ with boundary components $\{-\beta_{q,k,1}, -\alpha_q\}$ and $\{-\beta_{q,k,2}, -\alpha_q\}$, respectively.

   See Figure 4 for an illustration[7].

[4] In topology, a loop in a topological space X is a path whose initial point is equal to its terminal point.
[5] In topology, a boundary of a topological space X is a set of points that can be approached from both X and the outside of X. A boundary component is a connected component of the boundary. Here, sphere $\mathcal{S}_i$ does not have a boundary, but we can hollow a "hole" by adding a boundary component.
[6] Note that since the loop α_q has demand $\boldsymbol{b}(q)$, our construction is different from identifying the boundary component α_q of $\mathcal{T}_{q,i}$ and the boundary component $-\alpha_q$ of $\mathcal{T}_{q,j}$.
[7] As four tubes are connected to a single loop, to avoid the intersection of tubes before attaching the loop, a higher-dimensional space is required.

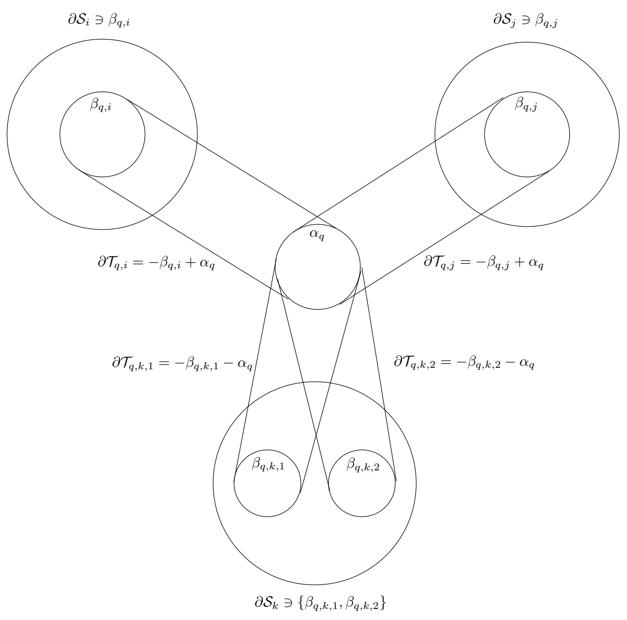
$\partial\mathcal{S}_i \ni \beta_{q,i}$
$\partial\mathcal{S}_j \ni \beta_{q,j}$
$\beta_{q,i}$
$\beta_{q,j}$
α_q
$\partial\mathcal{T}_{q,i} = -\beta_{q,i} + \alpha_q$
$\partial\mathcal{T}_{q,j} = -\beta_{q,j} + \alpha_q$
$\partial\mathcal{T}_{q,k,1} = -\beta_{q,k,1} - \alpha_q$
$\partial\mathcal{T}_{q,k,2} = -\beta_{q,k,2} - \alpha_q$
$\beta_{q,k,1}$
$\beta_{q,k,2}$
$\partial\mathcal{S}_k \ni \{\beta_{q,k,1}, \beta_{q,k,2}\}$

Figure 4 The construction for an average equation $\boldsymbol{x}(i) + \boldsymbol{x}(j) - 2\boldsymbol{x}(k) = 0$.

4. For each $i \in [n]$, the punctured sphere $\mathcal{S}_i$ and the tubes connected to $\mathcal{S}_i$ form a continuous topological space. We construct an oriented triangulation for this space such that the induced orientation of each edge on a loop α_q is consistent with the orientation of α_q. We will describe this oriented triangulation subroutine in Section 4.1.1. Let $\mathcal{K}$ be the oriented 2-complex. Let ∂_2 be the boundary operator of $\mathcal{K}$.
5. Each edge on a loop α_q has net demand $\boldsymbol{b}(q)$; each other edge has net demand 0. Let γ be the vector of the net flow demands.
6. On each triangulated sphere $\mathcal{S}_i$, we choose an arbitrary triangle $\Delta_i \in \mathcal{S}_i$ as the *central triangle*. Let $\boldsymbol{\Delta}^c$ be the set of all the central triangles.
7. We return $(\partial_2, \gamma, \boldsymbol{\Delta}^c)$.

The following algorithm MapSoln$\mathcal{B}_2$to$\mathcal{DA}$ maps a solution $\boldsymbol{f}$ to LE (∂_2, γ) to a solution $\boldsymbol{x}$ to LE $(\boldsymbol{A}, \boldsymbol{b})$.

Algorithm MapSoln$\boldsymbol{\mathcal{B}_2}to\boldsymbol{\mathcal{DA}}$.
Input: a tuple $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{f}, \boldsymbol{\Delta}^c)$, where $\boldsymbol{A} \in \mathcal{DA}$ is a $d \times n$ matrix, $\boldsymbol{b} \in \mathbb{R}^d$, $\boldsymbol{f} \in \mathbb{R}^t$, and $\boldsymbol{\Delta}^c$ is the set of n central triangles.
Output: a vector $\boldsymbol{x} \in \mathbb{R}^n$.
1. If $\boldsymbol{A}^\top \boldsymbol{b} = 0$, we return $\boldsymbol{x} = \boldsymbol{0}$.
2. Otherwise, we set $\boldsymbol{x}(i) = \boldsymbol{f}(\Delta_i)$, where $\Delta_i \in \boldsymbol{\Delta}^c$ is the central triangle on sphere $\mathcal{S}_i$.

4.1.1 Oriented Triangulation for Punctured Spheres and Tubes

We provide a concrete triangulation subroutine for the benefit of algorithm analysis.

Oriented Triangulation for Punctured Spheres

By our construction, each sphere $\mathcal{S}_i$ has $b_i = \sum_{q=1}^{d} |\boldsymbol{A}(q,i)|$ boundary components. We will create $\tilde{t}_i$ triangles and $\tilde{m}_i$ edges on $\mathcal{S}_i$, based on b_i.

1. If $b_i = 1$ (see Figure 5 (a)), the punctured sphere is topologically equivalent to a disk. In this case, $\mathcal{S}_i$ can be triangulated using a single triangle $[v^1_{(1)}, v^2_{(1)}, v^3_{(1)}]$, thus $\tilde{t}_i = 1, \tilde{m}_i = 3$.
2. If $b_i = 2$ (see Figure 5 (b)), the punctured sphere is topologically equivalent to an annulus. We subdivide the triangle $[v^1_{(1)}, v^2_{(1)}, v^3_{(1)}]$ obtained in the previous case by adding 6 interior edges between vertices of the inner and the outer boundaries: $[v^1_{(1)}, v^1_{(2)}], [v^1_{(1)}, v^2_{(2)}], [v^2_{(1)}, v^1_{(2)}], [v^2_{(1)}, v^3_{(2)}], [v^3_{(1)}, v^2_{(2)}], [v^3_{(1)}, v^3_{(2)}]$, thus $\tilde{t}_i = 6, \tilde{m}_i = 12$.
3. Generally, if $b_i = k$ (see Figure 5 (c)), we subdivide the rightmost triangle $[v^1_{(1)}, v^2_{(1)}, v^1_{(k-1)}]$ obtained in the case of $b_i = k - 1$ with the same method. By induction, we have

$$\tilde{t}_i = 5b_i - 4, \qquad \tilde{m}_i = 9b_i - 6, \qquad \text{for} \;\; b_i \geq 1. \tag{3}$$

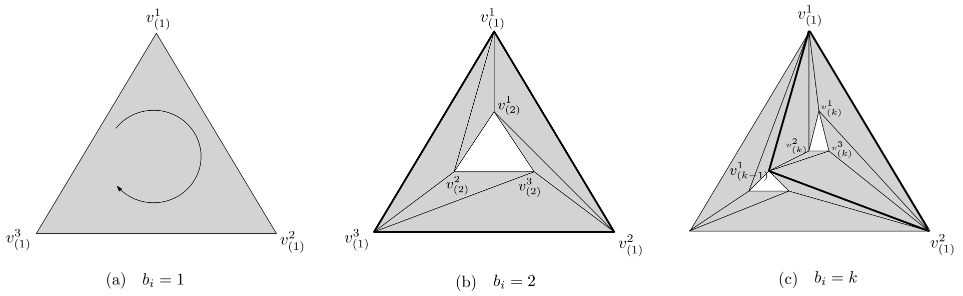

Figure 5 Oriented triangulation of punctured spheres. The light area represents the "holes" defined by boundary components.

The orientation for triangles on the same sphere should be identical. Without loss of generality, we orient all triangles clockwise. Note that with this triangulation method, all boundary components are composed of 3 edges.

Oriented Triangulation for Tubes

A tube is defined by two boundary components. By our construction, for every tube connected to $\mathcal{S}_i$, one of the two boundary components is always $-\beta_{q,i,*}$[8], and the other one is $\pm\alpha_q$, whose orientation depends on the sign of the entry $\boldsymbol{A}(q,i)$. Without loss of generality, we orient anti-clockwise for all α_q, thus clockwise for all $-\alpha_q$. Therefore, there are two possibilities of boundary component combinations.

1. If $\boldsymbol{A}(q,i) > 0$ (see Figure 6 (a)), then the two boundary components have opposite orientations: $-\beta_{q,i,*} = [v^1_{q,i,*}, v^3_{q,i,*}, v^2_{q,i,*}]$ and $\alpha_q = [v^1_q, v^2_q, v^3_q]$. We triangulate by matching $v^1_{q,i,*}$ to v^1_q, $v^2_{q,i,*}$ to v^2_q, and $v^3_{q,i,*}$ to v^3_q.
2. If $\boldsymbol{A}(q,i) < 0$ (see Figure 6 (b)), then the two boundary components have identical orientations: $-\beta_{q,i,*} = [v^1_{q,i,*}, v^3_{q,i,*}, v^2_{q,i,*}]$ and $-\alpha_q = [v^1_q, v^3_q, v^2_q]$. We triangulate by matching $v^1_{q,i,*}$ to v^1_q, $v^3_{q,i,*}$ to v^2_q, and $v^2_{q,i,*}$ to v^3_q.

In either case, only 6 triangles and 12 edges are required for an oriented triangulation of any tube $\mathcal{T}_{q,i,*}$. Again, we orient all triangles clockwise.

[8] We introduce a third element $* \in \{1,2\}$ in the subscript of $\beta_{q,k,*}$, which is activated only when $\boldsymbol{A}(q,k) = -2$.

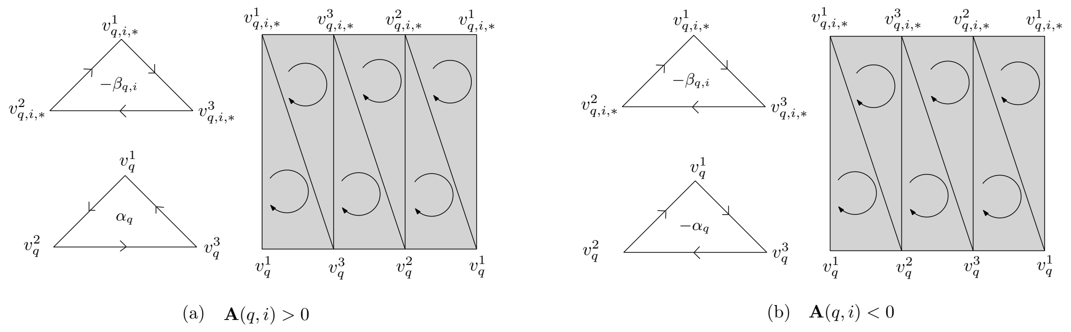

Figure 6 Oriented triangulation of tubes with opposite or identical boundary orientations.

4.2 Algorithm Runtime and Problem Size

In this section, we show that the reduction algorithm ReduceDAToB$_2$ and the solution mapping algorithm MapSolnB$_2$toDA both run in linear time, and ReduceDAToB$_2$ constructs a 2-complex whose size is linear in the number of nonzeros in the input linear equations.

▶ **Lemma 4.1** (Runtime). *Given a difference-average instance* LE $(\boldsymbol{A}, \boldsymbol{b})$ *where* $\boldsymbol{A} \in \mathbb{R}^{d\times n}$, *Algorithm* ReduceDAToB$_2(\boldsymbol{A}, \boldsymbol{b})$ *returns* $(\partial_2, \gamma, \boldsymbol{\Delta}^c)$ *in time* $O(\mathrm{nnz}(\boldsymbol{A}))$. *Given a solution* $\boldsymbol{f}$ *to* LE (∂_2, γ), *Algorithm* MapSolnB$_2$toDA$(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{f}, \boldsymbol{\Delta}^c)$ *returns* $\boldsymbol{x}$ *in time* $O(n)$.

Proof. For reduction, ReduceDAToB$_2(\boldsymbol{A}, \boldsymbol{b})$ calls the tube triangulation subroutine for $\|\boldsymbol{A}\|_1$ times, and the punctured sphere triangulation subroutine for n times. The tube triangulation subroutine runs in time $O(1)$ since the there are a constant number of triangles in a tube; and the punctured sphere triangulation subroutine runs in time $O(\|\boldsymbol{A}(:,j)\|_1)$ for the jth call, $j \in [n]$. Putting all together, the total runtime of ReduceDAToB$_2(\boldsymbol{A}, \boldsymbol{b})$ is $O\left(\|\boldsymbol{A}\|_1 + \sum_{j\in[n]} \|\boldsymbol{A}(:,j)\|_1\right) \le O(\mathrm{nnz}(\boldsymbol{A}))$, where we use the fact $\|\boldsymbol{A}\|_{\max} = 2$.

For solution mapping, the runtime of the algorithm MapSolnB$_2$toDA is obvious. ◀

▶ **Lemma 4.2** (Size of ∂_2). *Given a difference-average instance* LE $(\boldsymbol{A}, \boldsymbol{b})$, *let* $(\partial_2, \gamma, \boldsymbol{\Delta}^c)$ *be returned by* ReduceDAToB$_2(\boldsymbol{A}, \boldsymbol{b})$. *Suppose* $\partial_2 \in \mathbb{R}^{m\times t}$. *Then,*

1. $t \le 22\,\mathrm{nnz}(\boldsymbol{A})$*;*
2. $m \le 33\,\mathrm{nnz}(\boldsymbol{A})$*;*
3. $\mathrm{nnz}(\partial_2) \le 66\,\mathrm{nnz}(\boldsymbol{A})$.

Proof. We first compute the total number of triangles in the constructed 2-complex $\mathcal{K}$. For sphere $\mathcal{S}_j$, we have $\tilde{t}_j = 5b_j - 4$ triangles by (3), where $b_j = \sum_{i\in[d]} |\boldsymbol{A}(i,j)|$. Therefore, the number of triangles of all spheres is

$$\sum_{j=1}^{n} \tilde{t}_j = \sum_{j=1}^{n}(5\sum_{i\in[d]} |\boldsymbol{A}(i,j)| - 4) = 5\,\|\boldsymbol{A}\|_1 - 4n.$$

Moreover, each boundary component on spheres corresponds to a tube, and each tube has 6 triangles. Hence, the number of triangles of all tubes is $6\,\|\boldsymbol{A}\|_1$. Putting spheres and tubes together, we get

$$t = 11\,\|\boldsymbol{A}\|_1 - 4n \le 22\,\mathrm{nnz}(\boldsymbol{A}),$$

where the last inequality is because entries of $\boldsymbol{A}$ are bounded by 2.

Next, we compute the total number of edges in $\mathcal{K}$. By construction, each triangle has 3 incident edges and each edge is shared by a constant number of triangles (2 for interior edges, and 4 for boundary edges). Thus, we have

$$m \leq 1.5t \leq 33\,\mathrm{nnz}(\boldsymbol{A}).$$

Since each column of ∂_2 has exactly 3 nonzero entries, we have

$$\mathrm{nnz}(\partial_2) = 3t \leq 66\,\mathrm{nnz}(\boldsymbol{A}).$$ ◀

References

1 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

2 Mitali Bafna and Nikhil Vyas. Optimal fine-grained hardness of approximation of linear equations. In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

3 Mitchell Black, William Maxwell, Amir Nayyeri, and Eli Winkel. Computational topology in a collapsing universe: Laplacians, homology, cohomology. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2022.

4 Erik G Boman, Bruce Hendrickson, and Stephen Vavasis. Solving elliptic finite element systems in near-linear time with support preconditioners. *SIAM Journal on Numerical Analysis*, 46(6):3264–3284, 2008.

5 Gunnar Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, 2009.

6 Frédéric Chazal, Vin de Silva, Marc Glisse, and Steve Oudot. *The Structure and Stability of Persistence Modules*. Springer, October 2016.

7 Charles K Chui, HN Mhaskar, and Xiaosheng Zhuang. Representation of functions on big data associated with directed graphs. *Applied and Computational Harmonic Analysis*, 44(1):165–188, 2018.

8 Michael B Cohen, Brittany Terese Fasy, Gary L Miller, Amir Nayyeri, Richard Peng, and Noel Walkington. Solving 1-laplacians in nearly linear time: Collapsing and expanding a topological ball. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 204–216. SIAM, 2014.

9 Michael B Cohen, Jonathan Kelner, Rasmus Kyng, John Peebles, Richard Peng, Anup B Rao, and Aaron Sidford. Solving directed laplacian systems in nearly-linear time through sparse lu factorizations. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 898–909. IEEE, 2018.

10 Michael B. Cohen, Jonathan Kelner, John Peebles, Richard Peng, Anup B. Rao, Aaron Sidford, and Adrian Vladu. Almost-linear-time algorithms for markov chains and new spectral primitives for directed graphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 410–419, New York, NY, USA, 2017. ACM. doi:10.1145/3055399.3055463.

11 Michael B Cohen, Rasmus Kyng, Gary L Miller, Jakub W Pachocki, Richard Peng, Anup B Rao, and Shen Chen Xu. Solving sdd linear systems in nearly m log 1/2 n time. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 343–352. ACM, 2014.

12 Samuel I Daitch and Daniel A Spielman. Support-graph preconditioners for 2-dimensional trusses. *arXiv preprint cs/0703119*, 2007. arXiv:cs/0703119.

13 Mathieu Desbrun, Eva Kanso, and Yiying Tong. Discrete differential forms for computational modeling. In *Discrete differential geometry*, pages 287–324. Springer, 2008.

14 Ming Ding, Rasmus Kyng, Maximilian Probst Gutenberg, and Peng Zhang. Hardness results for laplacians of simplicial complexes via sparse-linear equation complete gadgets. *arXiv preprint*, 2022. `arXiv:2202.05011`.

15 Xun Dong and Michelle L Wachs. Combinatorial laplacian of the matching complex. *the electronic journal of combinatorics*, pages R17–R17, 2002.

16 Art Duval, Caroline Klivans, and Jeremy Martin. Simplicial matrix-tree theorems. *Transactions of the American Mathematical Society*, 361(11):6073–6114, 2009.

17 Art M Duval, Caroline J Klivans, and Jeremy L Martin. Cuts and flows of cell complexes. *Journal of Algebraic Combinatorics*, 41(4):969–999, 2015.

18 Beno Eckmann. Harmonische funktionen und randwertaufgaben in einem komplex. *Commentarii Mathematici Helvetici*, 17(1):240–255, 1944.

19 Herbert Edelsbrunner and John Harer. *Computational Topology: An Introduction.* American Mathematical Soc., 2010.

20 Joel Friedman. Computing betti numbers via combinatorial laplacians. *Algorithmica*, 21(4):331–346, 1998.

21 Robert Ghrist. Barcodes: The persistent topology of data. *Bulletin of the American Mathematical Society*, 45(1):61–75, 2008.

22 Allen Hatcher. *Algebraic topology.* Cambridge University Press, 2000.

23 M R Hestenes and E Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 1952.

24 Arun Jambulapati and Aaron Sidford. Ultrasparse Ultrasparsifiers and Faster Laplacian System Solvers. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 540–559. SIAM, 2021.

25 Xiaoye Jiang, Lek-Heng Lim, Yuan Yao, and Yinyu Ye. Statistical ranking and combinatorial hodge theory. *Mathematical Programming*, 127(1):203–244, 2011.

26 Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching Optimality for Solving SDD Linear Systems. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS '10, pages 235–244, USA, October 2010. IEEE Computer Society. `doi:10.1109/FOCS.2010.29`.

27 Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-m log n time solver for sdd linear systems. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 590–598. IEEE, 2011.

28 Ioannis Koutis, Gary L Miller, and David Tolliver. Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing. *Computer Vision and Image Understanding*, 115(12):1638–1646, 2011.

29 Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A. Spielman. Sparsified cholesky and multigrid solvers for connection laplacians. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 842–850, New York, NY, USA, 2016. ACM. `doi:10.1145/2897518.2897640`.

30 Rasmus Kyng, Richard Peng, Robert Schwieterman, and Peng Zhang. Incomplete nested dissection. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 404–417, 2018.

31 Rasmus Kyng and Sushant Sachdeva. Approximate gaussian elimination for laplacians-fast, sparse, and simple. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 573–582. IEEE, 2016.

32 Rasmus Kyng, Di Wang, and Peng Zhang. Packing LPs are hard to solve accurately, assuming linear equations are hard. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 279–296. SIAM, 2020.

33 Rasmus Kyng and Peng Zhang. Hardness results for structured linear systems. *SIAM Journal on Computing*, 49(4):FOCS17–280, 2020.

34 Lek-Heng Lim. Hodge laplacians on graphs. *Siam Review*, 62(3):685–715, 2020.

35 Wenye Ma, Jean-Michel Morel, Stanley Osher, and Aichi Chien. An l 1-based variational model for retinex theory and its application to medical images. In *CVPR 2011*, pages 153–160. IEEE, 2011.

36 William Maxwell and Amir Nayyeri. Generalized max-flows and min-cuts in simplicial complexes. *arXiv preprint*, 2021. `arXiv:2106.14116`.

37 James R Munkres. *Elements of algebraic topology.* CRC press, 2018.

38 Zipei Nie. Matrix anti-concentration inequalities with applications. *arXiv preprint*, 2021. `arXiv:2111.05553`.

39 Richard Peng and Daniel A. Spielman. An efficient parallel solver for SDD linear systems. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, pages 333–342, 2014.

40 Richard Peng and Santosh Vempala. Solving sparse linear systems faster than matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 504–521. SIAM, 2021.

41 Michael T Schaub, Austin R Benson, Paul Horn, Gabor Lippner, and Ali Jadbabaie. Random walks on simplicial complexes and the normalized hodge 1-laplacian. *SIAM Review*, 62(2):353–391, 2020.

42 Daniel A Spielman and Shang-Hua 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.

43 Volker Strassen. Gaussian elimination is not optimal. *Numerische mathematik*, 13(4):354–356, 1969.

44 Yiying Tong, Santiago Lombeyda, Anil N Hirani, and Mathieu Desbrun. Discrete multiscale vector field decomposition. *ACM transactions on graphics (TOG)*, 22(3):445–452, 2003.

45 Qianqian Xu, Qingming Huang, Tingting Jiang, Bowei Yan, Weisi Lin, and Yuan Yao. Hodgerank on random graphs for subjective video quality assessment. *IEEE Transactions on Multimedia*, 14(3):844–857, 2012.

46 Ke Ye and Lek-Heng Lim. Cohomology of cryo-electron microscopy. *SIAM Journal on Applied Algebra and Geometry*, 1(1):507–535, 2017.

47 Afra J. Zomorodian. *Topology for Computing*, volume 16. Cambridge university press, 2005.