DISS. ETH NO. 31123

Fine-Grained Complexity and Algorithms for Structured Linear Equations and Linear Programs

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES
(Dr. sc. ETH Zurich)

presented by

Ming Ding

born on 16.02.1995

accepted on the recommendation of

Prof. Dr. Rasmus Kyng, examiner
Prof. Dr. David Steurer, co-examiner
Prof. Dr. Peng Zhang, co-examiner

2025

Abstract

Linear Equations (LEs) and Linear Programs (LPs) are fundamental tools in scientific research and practical applications, with widespread use in optimization, engineering, and data analysis. Many real-world problems exhibit specific structural properties, enabling structured problems to be solved significantly faster than general-purpose solvers for LEs and LPs. This thesis explores several structured LEs and LPs from both complexity and algorithmic perspectives.

For LPs, we investigate the computational complexity of the *two-commodity flow* (2CF) problem, a natural generalization of the well-studied single-commodity flow (i.e., maximum flow) problem. Both 2CF and maximum flow problems can be formulated as LPs, each defining a distinct family of structured LPs. Given the success of efficient algorithms for maximum flow and the structural similarities between the two problems, it was widely conjectured that 2CF could also be solved more efficiently than general LPs. Contrary to this belief, we prove that solving 2CF — either exactly or approximately to high accuracy — is as computationally hard as solving general LPs. We prove this by giving a nearly-linear time reduction that encodes any LP as a 2CF problem on a sparse directed graph with only a polylogarithmic blow-up in problem size. Furthermore, careful error analysis guarantees that any approximate 2CF solution can be mapped back into an approximate LP solution with only a polynomial increase in error.

For LEs, we focus on systems of linear equations whose coefficient matrices are *combinatorial Laplacians*, a class of generalized graph Laplacians that arise in higher-dimensional problems on *simplicial complexes*. Combinatorial Laplacians play a crucial role in homology (a central tool in topology), and have various applications in data analysis and physical modeling problems. While nearly-linear time solvers are known for graph Laplacians, efficient solvers for combinatorial Laplacians are only known for restricted classes of simplicial complexes. By constructing a nearly-linear time reduction from general LEs to combinatorial Laplacians of 2-complexes, we show that solving linear equations in combinatorial Laplacians is computationally as hard as solving general LEs. Notably, our reduction preserves problem sparsity up to polylogarithmic factors.

Finally on the algorithmic side, despite our hardness results for combinatorial Laplacians, we demonstrate that imposing geometric structure on simplicial complexes can lead to substantial computational improvements. In particular, we develop sub-quadratic time algorithms for approximately solving LEs in *1-Laplacians* (i.e., 1-dimensional combinatorial Laplacians) on well-shaped simplicial complexes up to high accuracy.

Zusammenfassung

Lineare Gleichungen (LEs) und Lineare Programme (LPs) sind grundlegende Werkzeuge in der wissenschaftlichen Forschung und in praktischen Anwendungen. Sie finden breite Anwendung in den Bereichen Optimierung, Ingenieurwesen und Datenanalyse. Viele reale Probleme weisen spezifische strukturelle Eigenschaften auf, wodurch sich strukturierte Probleme deutlich schneller lösen lassen als mit allgemeinen Lösungsverfahren für LEs und LPs. Diese Arbeit untersucht verschiedene strukturierte LEs und LPs sowohl aus komplexitätstheoretischer als auch aus algorithmischer Perspektive.

Für LPs analysieren wir die rechnerische Komplexität des *Zwei-Warenfluss-Problems* (2CF), einer natürlichen Verallgemeinerung des gut erforschten Ein-Warenfluss-Problems (d.h. Maximalflussproblem). Sowohl das 2CF- als auch das Maximum-Flow-Problem lassen sich als LPs formulieren, wobei jedes eine eigene Familie von strukturierten LPs definiert. Aufgrund des Erfolgs effizienter Algorithmen für das Maximalflussproblem und der strukturellen Ähnlichkeit beider Probleme wurde häufig vermutet, dass auch 2CF effizienter lösbar sein könnte als allgemeine LPs. Im Gegensatz zu dieser Annahme zeigen wir, dass die exakte oder hochgenaue approximative Lösung des 2CF-Problems genauso schwierig ist wie die Lösung allgemeiner LPs. Dies beweisen wir durch eine Reduktion in nahezu linearer Zeit, die jedes LP auf ein 2CF-Problem auf einem dünnbesetzten gerichteten Graphen abbildet, wobei die Problemgrösse lediglich um einen polylogarithmischen Faktor wächst. Eine sorgfältige Fehleranalyse stellt zudem sicher, dass jede approximative 2CF-Lösung in eine approximative LP-Lösung umgewandelt werden kann, wobei sich der Fehler nur polynomial vergrössert.

Bei LEs konzentrieren wir uns auf lineare Gleichungssysteme, deren Koeffizientenmatrizen *kombinatorische Laplace-Operatoren* sind, eine Klasse verallgemeinerter Graph-Laplace-Operatoren, welche bei höherdimensionalen Problemen auf *Simplizialkomplexen* auftreten. Kombinatorische Laplace-Operatoren spielen eine zentrale Rolle in der Homologie (einem wesentlichen Werkzeug der Topologie) und besitzen vielfältige Anwendungen in der Datenanalyse sowie in der physikalischen Modellierung. Während für Graph-Laplace-Operatoren nahezu lineare Lösungsverfahren bekannt sind, existieren für kombinatorische Laplace-Operatoren effiziente Algorithmen bislang nur für eingeschränkte Klassen von Simplizialkomplexe. Mittels einer nahezu linearen Reduktion von allgemeinen linearen Gleichungssystemen auf kombinatorische Laplace-Gleichungssysteme auf 2-Komplexen zeigen wir, dass das Lösen linearer Gleichungen mit kombinatorischen Laplace-Operatoren ebenso komplex ist wie das Lösen allgemeiner linearer Gleichungssysteme. Bemerkenswerterweise erhält unsere Reduktion dabei die Problem-Sparsität bis auf polylogarithmische Faktoren.

Abschliessend demonstrieren wir auf algorithmischer Ebene, dass trotz unserer Härteresultate für kombinatorische Laplace-Operatoren eine geometrische Strukturierung der Simplizialkomplexe erhebliche algorithmische Verbesserungen ermöglicht. Insbesondere entwickeln wir subquadratische Algorithmen zur hochgenauen Approximation lin-

earer Gleichungssysteme mit *1-Laplace-Operatoren* (also eindimensionalen kombinatorischen Laplace-Operatoren) auf geometrisch wohldefinierten Simplizialkomplexe.

Acknowledgements

I would like to express my deepest gratitude to my Ph.D. supervisor, Rasmus Kyng. I still vividly remember our enjoyable first interview, where we discussed the graph sparsification paper—it was my first exposure to the world of theoretical computer science. I was so excited to join your research group and start this journey.

I am especially grateful for your remarkable generosity with your time, as well as your patience in coaching me and solving problems together. I have always been amazed by your exceptional problem-solving skills—how you develop deep intuition and break down complex problems into tractable pieces. Your optimism and persistence, even when things seemed impossible, have been truly inspiring. Your guidance has not only shaped the research in this thesis but has also deeply influenced the way I approach challenges in life. Beyond that, I also sincerely appreciate the freedom you gave me to explore my own research interests and career aspirations.

I would also like to extend my sincere thanks to my co-examiners, David Steurer and Peng Zhang, for kindly agreeing to join my doctoral examination committee and for taking the time and effort to support my Ph.D. defense and graduation. A special thanks to Peng Zhang, who has also been my mentor and co-author. Peng, I am truly grateful for your hands-on tutoring in numerical linear algebra, and for the extra time you spent with me after our project meetings with Rasmus—patiently answering my questions and offering encouragement when I needed it most.

I am also lucky to have had the chance to work with amazing collaborators: Shunhua Jiang, Omri Weinstein, Maximilian Probst Gutenberg, and Deeksha Adil. I really enjoyed our discussions, tackling tough research questions together, and sharing both the excitement and struggles along the way.

A big thanks to my colleagues at ETH Zurich. I thank all the members of Rasmus's research group. In particular, I want to thank my office mates—Federico Soldà, Simon Meierhans, and Aurelio Sulser—for the fun conversations, lunchtime discussions, and other group activities. I also want to thank Claudia Günthart and Andrea Salow for all their help with administrative matters.

Outside of research, I owe thanks to my friends, too many to name, who have made life outside of work fun and fulfilling. I feel incredibly lucky to have met my partner, Alexandre Binninger, through the summer retreat, one of the most memorable events organized by VMI at ETH Zurich. Alexandre, thank you for your support, encouragement, and companionship throughout this journey. You have brought immense happiness into my life. I am also grateful to your family for their warmth and kindness.

Finally, I want to thank my parents and brother, who, even from far away in China, have always given me unconditional love and support throughout my life.

Contents

Chapter 1

Introduction

Linear equations (LEs) and linear programs (LPs) are fundamental tools in both theory and practice. They arise in a wide range of fields, including computer science, engineering, economics, operations research, and the natural sciences. Moreover, efficient methods for solving LEs and LPs form the backbone of many core algorithmic paradigms, such as interior point methods, graph algorithms, and machine learning techniques. Given the importance of linear equations and linear programs, understanding their computational complexity is a central focus in theoretical computer science, numerical linear algebra, and optimization. Advances in this area not only deepen our theoretical understanding but also lead to faster and more scalable algorithms, driving progress in practical applications across science and engineering.

Given a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, a vector $\boldsymbol{b} \in \mathbb{R}^m$, a system of linear equations is defined as solving a vector $\boldsymbol{x} \in \mathbb{R}^n$ such that

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}.$$

In practice, we allow small error $\epsilon > 0$, and the goal is to compute an approximate vector $\boldsymbol{x} \in \mathbb{R}^n$ such that

$$\left\| \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \right\|_2 \leq \epsilon \left\| \boldsymbol{b} \right\|_2.$$

A linear program is an optimization problem with a linear objective function and linear constraints. Given a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, a vector $\boldsymbol{b} \in \mathbb{R}^m$, and a vector $\boldsymbol{c} \in \mathbb{R}^n$, a canonical form of LP is as follows:

$$\max_{\boldsymbol{x} \in \mathbb{R}^n} \{ \boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0} \}.$$

Both LE and LP can be considered as special cases of a more general problem – *linear regression* problem. Given a (tall) matrix $\boldsymbol{A}$ and a vector $\boldsymbol{b}$, it seeks a vector $\boldsymbol{x}$ to minimize $\left\| \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \right\|_p$ for some ℓ_p-norm. When $p = 1$ or ∞, the regression problem is equivalent to solving linear programs; when $p = 2$, the regression problem is equivalent to solving linear equations.

LE and LP are also related via Interior Point Methods (IPM), a family of algorithms that solve linear programs in provably polynomial time. IPM reduces solving LP to solving a sequence of $\widetilde{O}(\sqrt{\text{rank}(\boldsymbol{A})} L)$ systems of linear equations, where L is the bit complexity of the input [LS14]. Thus, any improvement in solving linear equations would imply a direct improvement in solving linear programs. Moreover, both linear programs and linear equations are frequently used as subroutines for other optimization problems.

Although significant progress has been made in designing fast solvers, the runtime of algorithms for solving linear equations and linear programs get stuck at the current matrix multiplicative time: the fastest solver for general system of n linear equations over $O(n)$ variables runs in time $O(n^\omega)$, where $\omega \approx 2.3728596\ldots$ [AW21]; the fastest solver for general linear programs with n constraints over $O(n)$ variables runs in time $\widetilde{O}(n^\omega)$[1] [CLS21; Jia+21]. However, these runtimes can be impractically slow for modern large-scale datasets.

In contrast to general LE and LP, many problems in practice possess additional structures that enable practically and provably faster solvers. Notable examples include: fast solvers for linear equations in graph Laplacians, which run in nearly-linear time [ST04; ST08; DS07; KS16; Kyn+16]; and fast algorithms for the maximum flow problem, a special structured LP problem, which can be solved in sub-quadratic and even almost-linear time [Mad13; Mad16; LS20; GLP21; Che+22]. These breakthroughs lead to a compelling open question:

> To what extent can these efficient algorithms be generalized to broader classes of structured linear equations and linear programs?

There are two complementary perspectives to approach this question: the *complexity perspective* and the *algorithmic perspective*. From the complexity perspective, our goal is to establish computational lower bounds for structured problems. Unlike classical complexity theory, which focuses on distinguishing "P" from "NP", our analysis falls under *fine-grained complexity*, as both LE and LP are already known to be in "P". Specifically, we aim to show that certain structured LEs and LPs problems cannot be solved faster than a given time complexity, unless widely accepted complexity assumptions for LE and LP fail. Such lower bounds help identify the fundamental limits of the structured problems and how much room there is for further progress. From the algorithmic perspective, the focus is on designing faster algorithms by leveraging the problem's specific structure. This helps narrow the gap between theoretical complexity barriers and performance of practical algorithms. Together, these two perspectives provide insights into the limitations and potentials of efficient solvers for structured problems.

In this thesis, we examine certain structured linear equations and linear programs from both the complexity perspective and the algorithmic perspective. From the complexity perspective, we investigate natural generalizations of problems with known fast algorithms. The two-commodity flow problem (2CF) generalizes the classical maximum flow problem.[2] Similarly, combinatorial Laplacians generalize graph Laplacians to higher-dimensional graphs – simplicial complexes. While 2CF and combinatorial Laplacians share structural similarities with their classical counterparts, no efficient algorithms have been discovered for these problems, apart from some specific restricted cases. In this thesis, we explain that the absence of efficient algorithms is not a coincidence. More specifically, we establish hardness results for 2CF, showing that approximately solving 2CF is equally hard as approximately solving general LP to high accuracy. And we prove that approximately solving linear equations in combinatorial Laplacians is as computationally hard as solving general linear equations to high accuracy. These results are achieved by constructing efficient reduction algorithms from a general LP or LE into these structured problems under study.

[1]We use $\widetilde{O}(\cdot)$ to hide a polylogarithmic factor of dimensions, condition number, and accuracy.
[2]The maximum flow problem can be viewed as a single-commodity flow problem.

Analogous to NP-completeness, we introduce new complexity classes, called *sparse-linear-equation (or sparse-linear-program) completeness*. The term "completeness" reflects the idea that if an algorithm with runtime $\widetilde{O}((\text{number of non-zeros})^c)$ is known for the structured LEs (or LPs), then an algorithm with runtime $\widetilde{O}((\text{number of non-zeros})^c)$ exists for general LEs (or LPs). In this terminology, the above results can be equivalently stated as 2CF and combinatorial Laplacians being sparse-LP and sparse-LE complete, respectively. Additionally, we introduce a simplified structured problem called *1-or-3 LE* (or *1-or-3 LP*), which is likewise sparse-LE (or sparse-LP) complete. We find that it plays a role similar to 3-SAT in NP-completeness. Our findings coincide with those of Průša and Werner (SODA'2017) [PW17], who further encoded 1-or-3 LP into a number of LP relaxations of NP-hard problems, showing that these LP relaxations are also sparse-LP complete.

Moreover, we investigate combinatorial Laplacians from an algorithmic perspective as well. While our hardness results indicate that the general structure of combinatorial Laplacians does not inherently provide computational advantages, we show that introducing mild *geometric assumptions* on the underlying simplicial complexes can lead to significant algorithmic improvements. Specifically, we develop a sub-quadratic time solver for linear equations in *1-Laplacians* (i.e., 1-dimensional combinatorial Laplacians) when the underlying simplicial complexes are *well-shaped* tetrahedral meshes.

1.1 Solving Linear Equations

Given linear equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ where $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, one can solve it by *Gaussian Elimination*, which runs in time $O(n^3)$. If applying the fast matrix multiplication by Strassen [Str69], one can speed up solving the linear equations in runtime $\widetilde{O}(n^\omega)$ by inverting $\boldsymbol{A}$ and then multiplying $\boldsymbol{A}^{-1}\boldsymbol{b}$, and it is by far the best known algorithm for solving general dense linear equations in dimensions $n \times n$ [AW21].

When the coefficient matrix $\boldsymbol{A}$ is sparse, i.e., $\boldsymbol{A}$ has $\widetilde{O}(n)$ non-zero entries, the direct matrix inversion approach is not desirable as $\boldsymbol{A}^{-1}$ cannot preserve sparsity of $\boldsymbol{A}$. Instead, iterative methods are more desirable to be used for solving sparse problems, where only matrix-vector multiplication and vector-vector operations are conducted in each iteration. For sparse linear equations with N non-zero 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 [PV21; Nie22] and the second is by conjugate gradient [HS+52]. In particular, if the coefficient matrix is symmetric positive semi-definite, the runtime for conjugate gradient is $\widetilde{O}(N\sqrt{\kappa})$.

Graph Laplacians and Nearly-Linear Time Solvers. An $n \times n$ Laplacian matrix can be viewed as an undirected graph over n vertices and m edges. Each vertex of the graph corresponds to a single row and a single column of the matrix. Each edge corresponds to an n-dimensional vector whose non-zeros are only at the position of its two endpoints. Formally, a graph Laplacian is defined as

$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A},$$

where $\boldsymbol{D} \in \mathbb{R}^{n \times n}$ is a diagonal matrix with vertex degrees on the diagonal entries; and $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is a *adjacency matrix*, defined such that $\boldsymbol{A}(u, v) = 1$ if (u, v) is an edge and

0 otherwise. Alternatively, graph Laplacians can also be defined using the vertex-edge incidence matrix $\boldsymbol{B} \in \mathbb{R}^{n\times m}$, where

$$\boldsymbol{B}(v,e) = \begin{cases} 1 & \text{if } e = (u,v) \\ -1 & \text{if } e = (v,u) \\ 0 & \text{otherwise} \end{cases}. \tag{1.1}$$

In this case, the graph Laplacian is given by

$$\boldsymbol{L} = \boldsymbol{B}\boldsymbol{B}^\top.$$

It is well-known that linear equations in graph Laplacians can be approximately solved in nearly-linear time in the number of non-zeros of the graph Laplacians [ST14; KMP10; KMP11; Kel+13; LS13; PS14; Coh+14b; KS16; Kyn+16; JS21].

Linear equations in graph Laplacians arise from using IPM for combinatorial problems on graphs. As a result, these fast Laplacian solvers have led to significant developments in algorithm design for graph problems such as maximum flow [Mad13; Mad16; Che+22], minimum cost flow and lossy flow [LS14; DS08], and graph sparsification [SS08]. Collectively, using fast Laplacian solvers to speed up graph algorithms was formulated as the "Laplacian Paradigm" [Ten10]. Beyond graph Laplacians, nearly-linear time solvers were extended to symmetric M-matrices [DS08], Block Diagonally Dominant matrices and Connection Laplacians [Kyn+16], Directed Laplacians [Coh+17b].

Combinatorial Laplacians. Combinatorial Laplacians generalize graph Laplacians to higher dimensional simplicial complexes – a collection of 0-simplexes (vertices), 1-simplexes (edges), 2-simplexes (triangles), and their higher dimensional counterparts. Given a d-dimensional simplicial complex $\mathcal{K}$, for each $0 \le i \le d$, ∂_i is a linear map that maps every i-simplex to a signed sum of its boundary $(i-1)$-faces. We define the i-Laplacian $\boldsymbol{L}_i$ to be

$$\boldsymbol{L}_i = \partial_i^\top \partial_i + \partial_{i+1}\partial_{i+1}^\top. \tag{1.2}$$

In particular, ∂_1 is the vertex-edge incidence matrix defined in Equation (1.1), and $\boldsymbol{L}_0$ is the graph Laplacian (following the convention, we define $\partial_0 = \boldsymbol{0}$).

Combinatorial Laplacians play an important role in both pure mathematics and applied areas. These matrices originate in the study of discrete Hodge decomposition [Eck44]: The kernel of $\boldsymbol{L}_i$ is isomorphic to the ith homology space of $\mathcal{K}$. The properties of combinatorial Laplacians have been studied in many subsequent works [Fri98; DW02; DKM09; DKM15; MN21]. A central problem in homology theory is evaluating the *Betti number*[3] of the ith homology space, which equals the rank of $\boldsymbol{L}_i$. In the case of homology over the reals, computing the rank of $\boldsymbol{L}_i$ can be reduced to solving a polylogarithmic number of linear equations in $\boldsymbol{L}_i$ [BV21]. In applied areas, the computation of Betti numbers over the reals is a key step in numerous problems in applied topology, computational topology, and topological data analysis [Zom05; Ghr08a; Car09; EH10; Cha+16]. In addition, combinatorial Laplacians have applications in statistical ranking [Jia+11; Xu+12], graphics and image processing [Ma+11a; Ton+03a], electromagnetism

[3]Informally, the ith Betti number is the number of i-dimensional holes on a topological surface. For example, the zeroth, first, and second Betti numbers represent the numbers of connected components, one-dimensional "circular" holes, and two-dimensional "voids" or "cavities," respectively.

and fluids mechanics [DKT08], data representations [CMZ18], cryo-electron microscopy [YL17a], biology [Sch+20]. We refer to the readers to [Lim20] for an accessible survey.

Inspired by the success of graph Laplacians, [Coh+14a] initiated the study of fast approximate solvers for 1-Laplacian linear equations. They designed a nearly-linear time approximate solver for simplicial complexes *with zero Betti numbers and known collapsing sequences.*[4] Later, [Bla+22] and [BN22] generalized this algorithm to subcomplexes of such a complex *with bounded first Betti numbers.*[5] One concrete example studied in these papers is convex simplicial complexes that piecewise linearly triangulate a convex ball in $\mathbb{R}^3$, for which a collapsing sequence exists and can be computed in linear time [Chi67; Chi80].

Lately, an algorithm about spectral sparsification for dense simplicial complexes was proposed by [Sav+25]. They proposed a probabilistic sampling method that reduces the number of $(k+1)$-dimensional simplexes from $m_{k+1} \approx O(m_k^{\frac{k+1}{k}})$ (the complete-graph case) down to $\widetilde{O}(m_k)$, in time $O(\delta^{-3} m_k^{1+\frac{4}{1+k}})$, where δ controls the approximation error. A direct application of this method is solving (dense) linear systems in the up-Laplacian $\partial_{k+1}\partial_{k+1}^\top \in \mathbb{R}^{m_k \times m_k}$. Specifically, by firstly sparsifying the $(k+1)$-simplexes via [Sav+25]'s approach and then using the state-of-the-art sparse linear solver from [Nie22], the resulting runtime is $\widetilde{O}((k^2 m_k)^{2.27})$. This combined approach beats the direct pseudo-inverse approach $O(m_{k+1}^\omega) = O\left(m_k^{(1+\frac{1}{k})\omega}\right)$ for small k.

Geometric Structures and Nested Dissection. Another widely studied class of structured linear equations consists of those whose non-zero entries exhibit geometrically embedded structures. Given an $n \times n$ matrix $\boldsymbol{A}$, its non-zero structures can be viewed as a graph over n vertices, and the (i, j)th entry or the (j, i)th entry of $\boldsymbol{A}$ is non-zero if there is an edge between vertex i and vertex j.

Asymptotically faster solvers exist if the geometric structures can be solved efficiently by *Nested Dissection* [Geo73; LRT79; MT90; AY10]. Specifically, Nested Dissection is a method to compute an ordering for Gaussian Elimination such that the number of *fill-ins* is small. Fill-ins refer to the number of new non-zeros introduced in the matrix when eliminating a variable. Using this idea, one can solve linear equations in a 2D planar in time $O(n^{1.5})$ [LRT79]. Additionally, faster solvers are also known for restricted classes of total-variation matrices [KMT11], stiffness matrices from elliptic finite element systems [BHV08a], and 2D and 3D truss stiffness matrices [DS07; Kyn+18].

1.2 Solving Linear Programs

The development of modern LP algorithms began with the introduction of the *Simplex Method* by George Dantzig in 1947. While this method has proven to be highly efficient in practice, its worst-case performance classifies it as an exponential-time algorithm in theory. In the 1970s, the *Ellipsoid Method* was introduced for solving certain non-linear optimization problems. In 1979, Khachiyan [Kha80a] demonstrated how this method could be adapted to solve LPs, marking a significant theoretical milestone as the first

[4]A collapsing sequence gives a good ordering of Gaussian Elimination so that it runs in linear time. However, deciding whether a general simplicial complex has a collapsing sequence is NP hard [Tan16].

[5]The solver has cubic dependence on the first Betti number.

polynomial-time algorithm for linear programming. However, the Ellipsoid Method is not competitive with the Simplex Method in practice.

The *Interior Point Methods* (IPMs) represent another major advancement in LP algorithms. Unlike the Simplex Method, which traverses along the boundary of the feasible polytope from one extreme point to another, or the Ellipsoid Method, which encircles the polytope, IPMs navigate through the interior of the feasible solution space. In 1984, Karmarkar [Kar84] proposed a polynomial-time IPM algorithm, further advancing the field of LP. In practice, IPMs often rival the Simplex Method and, in particular, outperform it on large-scale problems, making them a critical tool in modern optimization.

The fastest known solvers for general linear programs are based on Interior Point Methods, and in particular the central path methods [Ren88]. Recently, there has been significant progress on solvers for general linear programs, but the running time required to solve a linear program with n variables and $\widetilde{O}(n)$ constraints (assuming polynomially bounded entries and polytope radius) remains stuck at the $\widetilde{O}(n^{\max\{\omega,2+1/18\}})$ [Jia+21].

Maximum Flow/Single-Commodity Flow Problem. Single-commodity flow problems have been an area of tremendous success for the development of graph algorithms, starting with an era of algorithms influenced by early results on maximum flow and minimum cut [FF56] and later the development of powerful combinatorial algorithms for maximum flow with polynomially bounded edge capacities [Din70; ET75; GR98]. Later, a breakthrough nearly-linear time algorithm for electrical flows by Spielman and Teng [ST04] lead to the *Laplacian paradigm*. A long line of work explored direct improvements and simplifications of this result [KMP10; KMP11; Kel+13; PS14; KS16; JS21]. The Laplacian paradigm also motivated research into undirected maximum flow [Chr+11; LRS13; Kel+14; She13], which subsequently led to faster algorithms for directed maximum flow and minimum cost flow problems [Mad13; Mad16; LS20; KLS20; van+21; GLP21] building on powerful tools using mixed-ℓ_2, ℓ_p-norm minimizing flows [Kyn+19] and inverse-maintenance ideas [Che+20]. A recent breakthrough is the development of an almost-linear time algorithm for maximum flow and minimum-cost flow by Chen et al. [Che+22], which currently represents the fastest approach. This algorithm computes exact maximum flows and minimum-cost flows on directed graphs with $|E|$ edges and polynomially bounded integral demands, costs, and capacities in $|E|^{1+o(1)}$ time.

To compare these advances with general-purpose LP solvers, consider the running times for state-of-the-art single-commodity maximum flow algorithms on a graph with $|V|$ vertices and $|E|$ edges. For sparse graphs where $|E| = \widetilde{O}(|V|)$, formulating the maximum flow problem as a linear program and solving it with general LP solvers results in a running time of $\widetilde{O}(|V|^{2.372\cdots})$. In contrast, specialized maximum flow solvers achieve a much faster $\widetilde{O}(|V|^{1+o(1)})$ runtime. On dense graphs with $|E| = \Theta(|V|^2)$, the gap narrows but remains significant: maximum flow solvers achieve a runtime of $\widetilde{O}(|V|^2)$, compared to $\widetilde{O}(|V|^{2.372\cdots})$ for general LP solvers. Therefore, the structure of maxflow can be used to develop special-purpose solvers that outperform general LP solvers.

Multi-Commodity Flow Problem. Multi-commodity flow problems are a well-studied area in algorithm design and have been the subject of numerous surveys [Ken78; AMO93; OMV00; BKV09; Wan18], in part because a large number of problems can be expressed as variants of multi-commodity flow. Significant advancements have been made in solving the undirected multi-commodity flow problem, particularly in the low-accuracy regime. Leighton et al. [Lei+95] showed that undirected capacitated k-commodity flow in a graph with $|E|$ edges and $|V|$ vertices can be approximately solved in $\widetilde{O}(k|E||V|)$ time, completely routing all demands with $(1+\epsilon)$ times the optimal congestion, albeit with a poor dependence on the error parameter ϵ. This beats the solve-times for linear programming in sparse graphs for small k, even with today's LP solvers that run in current matrix multiplication time, albeit with much worse error. This result inspired a series of follow-up works focusing on improving low-accuracy algorithms [GK07; Fle00; Mad10]. Later breakthroughs in achieving almost- and nearly-linear time algorithms for undirected single-commodity maximum flow also led to faster algorithms for undirected k-commodity flow [Kel+14; She13; Pen16], culminating in Sherman's introduction of area-convexity to build a $\widetilde{O}(k|E|\epsilon^{-1})$ time algorithm for approximate undirected k-commodity flow [She17].

In contrast, directed 2-commodity flow problems were seemingly harder, despite the discovery of non-trivial algorithms for some special cases [Eva76; Eva78]. Alon Itai [Ita78] proved a polynomial-time reduction from linear programming to directed 2-commodity flow, before a polynomial-time algorithm for linear programming was known. For decades, the only major progress on solving directed multi-commodity flow to high accuracy came from improvements to general linear program solvers [Kha80b; Kar84; Ren88; Vai89].

Recently, a surprising advancement was made by van den Brand and Zhang [DZ23a], who developed a high-accuracy algorithm for the k-commodity flow problem with a runtime of $\widetilde{O}(k^{2.5}\sqrt{|E|}|V|^{\omega-1/2})$. For comparison, solving the problem using the state-of-the-art general LP solver [CLS21; Jia+21] takes runtime is $\widetilde{O}((k|E|)^{\omega})$. While this algorithm is slightly less efficient than general LP solvers in sparse graphs (i.e., $|E| = \widetilde{O}(|V|)$), it outperforms them on dense graphs (i.e., $|E| = \Theta(|V|^2)$). This marks the first improvement to high accuracy multi-commodity flow algorithms that does not just stem from improvements to general linear program solvers.

1.3 Fine-Grained Complexity Analysis

In classical complexity theory, we typically classify decision problems into complexity classes such as "P" or "NP", but do not often measure how difficult a polynomial-time problem might be. Fine-grained complexity theory addresses this gap by quantifying the precise difficulty of problems in P, going beyond the standard "P vs NP" framework. It explains conditional time-complexity barriers for polynomial-time solvable problems, relying on "core" conjectures (like the Strong Exponential-Time Hypothesis [IP01], the Orthogonal Vectors Conjecture [Wil05], and assumptions about matrix multiplication). By designing so-called *fine-grained reductions* from problem A to problem B, one can show that a seemingly faster algorithm for problem B would imply a surprising speedup for problem A. If problem A is believed to require a certain runtime, then problem B also inherits this hardness barrier. Over the last decade, fine-grained complexity has blossomed, yielding breakthroughs in explaining why certain polynomial-time problems remain "as slow as" the best-known methods. It has also led to the discovery of many meaningful relationships between problems, and equivalent classes. For a thorough treatment of the foundational ideas and recent developments in fine-grained complexity and algorithms, we refer interested readers to [Wil18].

Fine-Grained Reductions. The basic setup of fine-grained lower bounds is similar to classic NP-harness reductions: A fine-grained reduction from problem A to problem B

is an algorithm that given an instance I of size n for problem A, computes in time $t(n)$ an equivalent instance J of size $s(n)$ for problem B. Thus, if there exists an algorithm that solves problem B in time $T(n)$, then by this reduction there is an algorithm solving problem A in time $t(n) + T(s(n))$. In particular, if $t(n) + T(s(n))$ is faster than the hypothesized optimal time complexity of problem A, then problem B cannot be solved in time $T(n)$ assuming the hypothesis for A. This provides a conditional lower bound of problem B. For instance, under the rank-finding conjecture (i.e., no sub-n^ω method can compute the rank of and $\Theta(n) \times \Theta(n)$ matrix), [BV21] gave conditional lower bounds for general (dense) system of linear equations that even to get constant-factor or $(1 - O(1/n^c))$-factor approximation, we likely need $O(n^\omega)$ time.

Sometimes the chain of reductions goes in both directions, showing that problems A and B are, in effect, “fine-grained equivalent”. In this case, if either problem's time complexity were improved, it would yield an improvement for the other. For example, [WW18] introduced “subcubic reducibility”, which requires $t(n) = O(n^c)$ for some constant $c < 3$. This work showed that many important problems on graphs and matrices solvable in $O(n^3)$ time are equivalent under subcubic reductions (e.g., Negative-Triangle Detection, Matrix-Product Verification). A single-direction reduction can also suffice if problem B is a special case of A. Itai [Ita78] devised a polynomial-time reduction from a linear program to a 2-commodity flow problem. Since 2-commodity flow is a special case of linear programming, Itai concluded that solving 2-commodity flow and solving linear programs are polynomial-time equivalent. In a similar vein, Kyng and Zhang [KZ17] studied several structured linear systems, including two-commodity Laplacians, 2D truss stiffness matrices, and total variation matrices. By constructing reduction algorithms from general linear equations to these structured linear equations, it is concluded that if we can quickly solve either of these structured linear equations, then we can quickly solve linear equations in any matrix.

Error Analysis in Reductions. Many fine-grained reductions focus on exact versions of problems. For instance, Itai [Ita78] established that exactly solving linear programming is polynomial-time equivalent to exactly solving a sequence of related problems (e.g., linear equalities under nonnegative constraints, homologous flow, two-commodity flow). Dobkin and Reiss [DR80] later identified additional LP-complete problems in computational geometry, such as hyperplane intersection and extreme-point detection. Trevisan and Xhafa [TX98] proved that exactly solving packing LPs is P-complete. And [PW17] showed that solving LP relaxations of many NP-hard combinatorial optimization problems exactly is as hard as solving general LPs exactly.

In practice, however, approximate algorithms are more widely used. In fine-grained complexity analysis, they typically use the same reduction frameworks as in the exact setting, but must also account for *error analysis*: tracking the error propagation when mapping solutions to the reduced problem (problem B) back to that of the original problem (problem A). For instance, Kyng and Zhang [KZ17] not only provided reduction algorithms from general linear equations to certain structured systems, but also conducted error analysis to show that errors remain polynomially bounded when mapping solutions back to the original system. Moreover, they explicitly bounded the condition number of the new matrix with respect to that of the original one.

Approximate algorithms usually have runtimes dependent on error parameters. Error analysis in fine-grained complexity analysis can sometimes help to set a separation between low-accuracy and high-accuracy algorithms. A low-accuracy algorithm's runtime is polynomial in $1/\epsilon$, whereas high-accuracy algorithms depend polylogarithmically on $1/\epsilon$. Musco et al. [Mus+19] developed a low-accuracy algorithm for approximating several spectral sum problems, breaking the $O(n^{\omega})$ barrier. On the hardness side, they showed that achieving milder error dependencies would imply breakthroughs on the triangle detection algorithms for general graphs running in faster than the state-of-the-art matrix multiplication time, indicating that fast high-accuracy spectrum approximation algorithms are unlikely without major algorithmic breakthroughs. With a similar spirit, Kyng, Wang, and Zhang [KWZ20] studied whether the error dependence in packing LP solvers can be improved, beyond the known $\widetilde{O}(N/\epsilon)$ bound (where N is the number of non-zero entries). By creating a fine-grained reduction from solving dense or sparse systems of linear equations to such packing LPs, the paper shows that faster ϵ-dependence in packing LP solvers would contradict known hardness assumptions for systems of linear equations.

1.4 Our Results

Hardness Results for Two-Commodity Flow. We explore the hardness of *2-commodity maximum throughput flow*, which for brevity we refer to as the 2-commodity flow problem or 2CF. Given a directed graph with edge capacities and two source-sink pairs, this problem requires us to maximize the sum of the flows routed between the two source-sink pairs, while satisfying capacity constraints and flow conservation at the remaining nodes. We relate the difficulty of 2CF to that of LP by developing a highly efficient reduction from the former to the latter. The result presented below is joint work with Kyng and Zhang [DKZ22].

Theorem 1.4.1 (Hardness for 2CF (Informal))**.** *Consider any polynomially bounded linear program* $\max\{\boldsymbol{c}^{\top}\boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$ *with integer coefficients and N non-zero entries. In nearly-linear time, this linear program can be transformed into a 2-commodity flow problem that is feasible if and only if the original linear program is feasible. The resulting 2-commodity flow instance has $\widetilde{O}(N)$ edges and polynomially bounded integral edge capacities. Furthermore, any solution to the 2-commodity flow instance with at most ϵ additive error on each constraint and value at most ϵ from the optimum can be converted into a solution to the original linear program with additive error $\widetilde{O}(\mathrm{poly}(N)\epsilon)$ on each constraint and similarly value within $\widetilde{O}(\mathrm{poly}(N)\epsilon)$ of the optimum.*

This implies that, for any constant $c > 1$, if any 2-commodity flow instance with polynomially bounded integer capacities can be solved with ϵ additive error in time $\widetilde{O}\left(|E|^{c} \cdot \mathrm{poly}\log(1/\epsilon)\right)$, then any polynomially bounded linear program can be solved with ϵ additive error in time $\widetilde{O}\left(N^{c} \cdot \mathrm{poly}\log(1/\epsilon)\right)$.

We say a problem is *sparse-linear-program complete* if any polynomially bounded linear program with N non-zero entries can be reduced, in $\widetilde{O}(N)$ time, to an instance of the problem such that: (1) The size of the resulting instance increases by at most a polylogarithmic factor in N; (2) Any solution to the reduced problem with additive error ϵ can be efficiently mapped into a solution of the original linear program with additive error $\mathrm{poly}(N)\epsilon$. With this terminology, the hardness result for 2CF can be concisely stated as: 2CF is sparse-linear-program complete.

The formal definitions of LP and 2CF are provided later in Chapter 3 (in Definitions 3.3.1 and 3.3.7). It is important to note that, under our definitions, any 2CF problem is already an LP, and so no reduction in the other direction is necessary. Additionally, our definitions of approximate solutions for LPs (Definition 3.3.4) and 2CF problems (Definition 3.3.16) ensure compatibility: solving a 2CF problem as an LP and obtaining an approximate solution guarantees that the 2CF problem is also approximately solved according to our definition of approximate solutions for 2CF.

Our proof follows the outline of Itai's polynomial-time reduction of a linear program to a 2-commodity flow problem [Ita78], which showed that *exactly* solving 2-commodity flow and *exactly* solving linear programming are polynomial-time equivalent. We improve Itai's reduction to nearly preserve the problem representation size in each step. In addition, we establish an error bound for approximately solving each intermediate problem in the reduction, and show that the accumulated error is polynomially bounded. We remark that our hardness results only rule out possibilities of fast multi-commodity flow algorithms for *sparse directed* graphs in the *high accuracy* regime. Outside of this setting, efficient multi-commodity flow solvers may still exist [She17; Mad10; DZ23a]. We also remark that our reduction does not run in *strongly polynomial* time, and it remains an open question whether 2-commodity flow and linear programming are equivalent in the strongly polynomial time regime.

Hardness Results for Combinatorial Laplacians. We study linear equations in combinatorial Laplacians of k-dimensional simplicial complexes (k-complexes), a natural generalization of graph Laplacians. 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. Our findings demonstrate that solving linear equations in combinatorial Laplacians of 2-complexes is as computationally hard as solving general linear equations. The following results are from the paper [Din+22] coauthored with Kyng, Probst Gutenberg, and Zhang.

Recall that approximately solving linear equation $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is to find $\tilde{\boldsymbol{x}}$ such that $\left\|\boldsymbol{A}\tilde{\boldsymbol{x}} - \boldsymbol{b}\right\|_2 \leq \epsilon \left\|\boldsymbol{b}\right\|_2$ for some ϵ. To formalize our result, we introduce the notion of *sparse-linear-equation complete*. Consider a family of matrices $\mathcal{B}$. We say that $\mathcal{B}$ is sparse-linear-equation complete if, for any instance $(\boldsymbol{A}, \boldsymbol{b}, \epsilon)$ we can construct an instance $(\boldsymbol{B} \in \mathcal{B}, \boldsymbol{c}, \delta)$ in $\widetilde{O}(\text{nnz}(\boldsymbol{A}))$ time such that: (1) $\text{nnz}(\boldsymbol{B}) = \widetilde{O}(\text{nnz}(\boldsymbol{A}))$; (2) $\delta = \epsilon / \text{poly}(\text{nnz}(\boldsymbol{A}))$, and any approximate solution $\boldsymbol{y}$ satisfying $\left\|\boldsymbol{B}\boldsymbol{y} - \boldsymbol{c}\right\|_2 \leq \delta \left\|\boldsymbol{c}\right\|_2$ can be mapped back to an approximate solution $\tilde{\boldsymbol{x}}$ to $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ with error at most ϵ.

In this terminology, we succinctly state the main theorems derived in this study.

Theorem 1.4.2 (Hardness for Combinatorial Laplacians (Informal))**.** *Linear equations in combinatorial Laplacians of 2-complexes are sparse-linear-equation complete.*

The result is established through a three-step reduction process:

1. **Reduction to Difference-Average Equations:** In the first step, we show that a simple class of linear equations, which we term *difference-average equations*, is sparse-linear-equation complete. These equations take one of two forms:

$$\boldsymbol{x}(i) - \boldsymbol{x}(j) = b \quad \text{or} \quad \boldsymbol{x}(i) + \boldsymbol{x}(j) = 2\boldsymbol{x}(k).$$

   This reduction was implicitly demonstrated in [KZ17] as an intermediate step, and leads to the following result:

Theorem 1.4.3 (Hardness for Difference-Average Matrices (Informal))**.** *Linear equations in the difference-average matrices are sparse-linear-equation complete.*

For completeness, we provide an explicit and simplified proof in Section 5.1.

2. **Reduction to Boundary Operators of 2-Complexes:** Next, we reduce the problem of solving difference-average equations to solving linear equations in the boundary operator ∂_2 of a 2-complex. This leads to the following result:

   Theorem 1.4.4 (Hardness for Boundary Operators (Informal))**.** *Linear equations in the boundary operators ∂_2 of 2-complexes are sparse-linear-equation complete.*

   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 $\boldsymbol{d}$. Our reduction is inspired by a reduction in [MN21] that proves NP-hardness of computing maximum *integral* flows in 2-complexes via a reduction from graph 3-coloring problem. An important aspect of our contribution is that we carefully control the number of non-zeros 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. Details are provided in Section 4.5 and 4.6.

3. **Reduction to Combinatorial Laplacians of 2-Complexes:** In the final step, we reduce the problem of solving in a boundary operator ∂_2 to solving in the corresponding combinatorial Laplacian $\boldsymbol{L}_1$. Consequently, the combinatorial Laplacian problem is at least as hard as the boundary operator problem. This is captured in the following lemma:

   Lemma 1.4.5 (Reduction from Boundary Operators to Combinatorial Laplacians (Informal))**.** *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 of this lemma relies on standard arguments in linear algebra. A formal statement of the theorem, along with a detailed proof, is provided in Appendix B.1.

Again, we remark that our hardness results rule out the possibility of fast combinatorial Laplacian solvers for sparse simplicial complexes in the high-accuracy regime. However, efficient solvers may still exist outside this setting [Sav+25].

More Hardness Results. Motivated by the hardness results for two-commodity flow and combinatorial Laplacians, we aim to broaden the classes of sparse-linear-program complete and sparse-linear-equation complete problems, in a way analogous to the well-established theory of NP-completeness. However, we later found that our ideas align with those of Průša and Werner [PW17], who proved that LP relaxations of some NP-hard combinatorial optimization problems are sparse-LP complete. While the results presented here remain unpublished, we include them for completeness—particularly since our derivation techniques differ from those used in [PW17].

Our approach begins with *difference-average* equations, which we reduce to a simpler form that we call *1-or-3 equations*. In this form, each row has either one or three non-zero entries, taking one of two shapes:

$$\boldsymbol{x}(i) + \boldsymbol{x}(j) = \boldsymbol{x}(q) \quad \text{or} \quad \boldsymbol{x}(i) = 1.$$

We extend this reduction from linear equalities (LE) to linear programs (LP), obtaining *(simplified) 1-or-3 linear programs* of the form

$$\{\boldsymbol{A}\boldsymbol{x} = \mathbf{1}, \quad \boldsymbol{x} \geq \mathbf{0}\},$$

where each equality constraint is either

$$\boldsymbol{x}(i) + \boldsymbol{x}(j) + \boldsymbol{x}(q) = 1 \quad \text{or} \quad \boldsymbol{x}(i) = 1.$$

From this simplified LP form, one can construct relaxations of various NP-hard combinatorial optimization problems—such as Set Cover, Set Packing, Maximum Satisfiability, and Maximum Independent Set. For additional details on these constructions, we refer readers to [PW17]. These constructions highlight how fundamental the 1-or-3 form is to understanding the complexity of other structured linear equations and linear programs.

Efficient Algorithm for 1-Laplacian Solvers. Previously, nearly-linear time approximate solvers for 1-Laplacian were developed for simplicial complexes with *known collapsing sequences* and *bounded Betti numbers*, such as those triangulating a three-ball in $\mathbb{R}^3$ [Coh+14a; Bla+22; BN22]. Additionally, Nested Dissection provides quadratic-time exact solvers for more general systems where the non-zero structures correspond to well-shaped simplicial complexes embedded in $\mathbb{R}^3$. We generalize the specialized solvers for 1-Laplacians to simplicial complexes with additional geometric structures but *without collapsing sequences and bounded Betti numbers*. Furthermore, we improve the runtime of Nested Dissection for these broader settings. The results below are from the paper [DZ23b], coauthored with Zhang.

We focus on simplicial complexes possessing the following two geometric structures:

1. Each simplex has a bounded *aspect ratio*,[6] referred to as being *"stable"*.
2. The complex can be partitioned into disjoint, balanced regions with well-shaped *interiors* and *boundaries*, a property we call *"r-hollowing"*.

Our first result applies to *pure* 3-complex[7] with a given r-hollowing.

Theorem 1.4.6 (1-Laplacian Solver for a Pure 3-Complex (Informal)). *Let $\mathcal{K}$ be a pure 3-complex embedded in $\mathbb{R}^3$ and composed of n stable simplexes. Given an r-hollowing for $\mathcal{K}$, for any $\epsilon > 0$, we can approximately solve a system in the 1-Laplacian of $\mathcal{K}$ within error ϵ in time:*

$$O\left(nr + n^{4/3}r^{5/18}\log(n/\epsilon) + n^2r^{-2/3}\right).$$

[6]The aspect ratio of a geometric shape S is the radius of the smallest ball containing S divided by the radius of the largest ball contained in S.

[7]A simplicial complex is *pure* if every maximal simplex (i.e., a simplex that is not a proper subset of any other simplex in the complex) has the same dimension. For example, a pure 3-complex is a tetrahedron mesh that consists of tetrahedra and their subsimplexes.

In particular, the runtime is minimized (up to constant factors) when $r = \Theta(n^{3/5})$, resulting in:

$$O(n^{8/5}\log(n/\epsilon)).$$

In Section 9 of [DZ23b], sufficient conditions are established for 3-complexes that enable the computation of an r-hollowing in nearly-linear time. We omit this part from the thesis.

We remark that the runtime in Theorem 1.4.6 does not depend on the Betti numbers of $\mathcal{K}$ and does not require collapsing sequences. When $r = o(n)$ and $r = \omega(1)$, the runtime is $o(n^2)$, asymptotically faster than Nested Dissection [MT90]. The solver in [BN22] for a 1-Laplacian system on $\mathcal{K}$, as stated in Theorem 1.4.6, has a runtime of $\widetilde{O}(\beta^3 m)$, where m is the number of simplexes in $\mathcal{X} \supset \mathcal{K}$ with a known collapsing sequence and β is the first Betti number of $\mathcal{K}$. In the worst-case scenario, m can be as large as $\Omega(n^2)$. However, [BN22] does not require the simplexes to be stable or $\mathcal{K}$ to have a known r-hollowing.

Next, we examine unions of pure 3-complexes glued together by identifying subsets of simplexes on the boundary components (called *exterior simplexes*). Each 3-complex chunk has a $\Theta(n_i^{3/5})$-hollowing, where n_i is the number of simplexes in the ith chunk. We remark that such a union of 3-complexes, called $\mathcal{U}$, *may not be embeddable in* $\mathbb{R}^3$. As a result, previously established methods, including those from [Coh+14a; Bla+22; BN22] and Nested Dissection, cannot be directly applied in this scenario. Building on our algorithm from Theorem 1.4.6, we develop an efficient solver for $\mathcal{U}$ with a runtime that scales sub-quadratically with the size of $\mathcal{U}$ and polynomially with the number of chunks and the number of shared exterior simplexes by more than one chunk.

Theorem 1.4.7 (1-Laplacian Solver for a Union of Pure 3-Complexes (Informal))**.** *Let $\mathcal{U}$ be a union of pure 3-complexes that are glued together by identifying certain subsets of their exterior simplexes. Assume that each 3-complex chunk is embedded in $\mathbb{R}^3$, contains n_i stable simplexes, and has a known $\Theta(n_i^{3/5})$-hollowing. For any $\epsilon > 0$, we can solve a system in the 1-Laplacian of $\mathcal{U}$ within error ϵ in time*

$$\widetilde{O}\left(n^{8/5} + n^{3/10}k^2 + k^3\right),$$

where n is the number of simplexes in $\mathcal{U}$ and k is the number of exterior simplexes shared by more than one complex chunk.

When $h = \widetilde{O}(1)$ and $k = \widetilde{O}(n^{1/2})$, the runtime of the solver in Theorem 1.4.7 matches that of Theorem 1.4.6. Moreover, when $h = o(n^{2/5}), k = o(n^{3/5})$, the runtime is $o(n^2)$, asymptotically faster than Nested Dissection.

1.5 Organization of Thesis

In Chapter 2, we introduce the notations and foundational concepts from linear algebra and topology that will be used throughout the thesis. In Chapter 3, we establish the hardness of solving two-commodity flow problems, proving Theorem 1.4.1. Chapter 4 focuses on the hardness of solving linear equations in combinatorial Laplacians of simplicial complexes and proves Theorem 1.4.2. Chapter 5 presents additional hardness results, including reduction algorithms for difference-average equations, 1-or-3 equations, and 1-or-3 linear programs. In Chapter 6, we present an efficient algorithm for solving 1-Laplacian systems, proving Theorems 1.4.6 and 1.4.7. In Appendix A, B, and C, we include missing proofs of Chapter 2, 4, and 6, respectively.

Chapter 2

Preliminaries

2.1 Vectors and Matrices

Indexing. Given a vector $\boldsymbol{x} \in \mathbb{R}^n$, for $1 \le i \le n$, we let $\boldsymbol{x}(i)$ be the ith entry of $\boldsymbol{x}$; for $1 \le i < j \le n$, let $\boldsymbol{x}(i:j)$ be $\begin{bmatrix} \boldsymbol{x}(i) & \boldsymbol{x}(i+1) & \ldots & \boldsymbol{x}(j) \end{bmatrix}^\top$. We use $\mathbf{1}_n, \mathbf{0}_n$ to denote an n-dimensional all-one vector and all-zero vector, respectively.

Given a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, for $1 \le i \le m, 1 \le j \le n$, we let $\boldsymbol{A}(i,j)$ be the (i,j)th entry of $\boldsymbol{A}$; for $S_1 \subseteq \{1, \ldots, m\}, S_2 \subseteq \{1, \ldots, n\}$, let $\boldsymbol{A}(S_1, S_2)$ be the submatrix with row indices in S_1 and column indices in S_2. Furthermore, we let $\boldsymbol{A}(S_1, :) = \boldsymbol{A}(S_1, \{1, \ldots, n\})$ and $\boldsymbol{A}(:, S_2) = \boldsymbol{A}(\{1, \ldots, m\}, S_2)$. In particular, we use $\boldsymbol{A}(i)$ to denote the ith row of $\boldsymbol{A}$. In addition, we use $\mathrm{nnz}(\boldsymbol{A})$ to denote the number of non-zero entries of $\boldsymbol{A}$. Without loss of generality, we assume that $\mathrm{nnz}(\boldsymbol{A}) \ge \max\{m, n\}$.

Subspaces. The image of $\boldsymbol{A}$ is the linear span of the columns of $\boldsymbol{A}$, denoted by $\mathrm{Im}(\boldsymbol{A})$, and the kernel of $\boldsymbol{A}$ to be $\{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} = \mathbf{0}\}$, denoted by $\mathrm{Ker}(\boldsymbol{A})$. A fundamental theorem of Linear Algebra states

$$\mathbb{R}^m = \mathrm{Im}(\boldsymbol{A}) \oplus \mathrm{Ker}(\boldsymbol{A}^\top).$$

Fact 2.1.1. *For any matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $Im(\boldsymbol{A}) = Im(\boldsymbol{A}\boldsymbol{A}^\top)$.*

All the facts in this section are well-known. For completeness, we include their proofs in Appendix A.

Pseudo-Inverse and Projection Matrix. The pseudo-inverse of $\boldsymbol{A}$ is defined to be a matrix $\boldsymbol{A}^\dagger$ that satisfies all the following four criteria: (1) $\boldsymbol{A}\boldsymbol{A}^\dagger\boldsymbol{A} = \boldsymbol{A}$, (2) $\boldsymbol{A}^\dagger\boldsymbol{A}\boldsymbol{A}^\dagger = \boldsymbol{A}^\dagger$, (3) $(\boldsymbol{A}\boldsymbol{A}^\dagger)^\top = \boldsymbol{A}\boldsymbol{A}^\dagger$, (4) $(\boldsymbol{A}^\dagger\boldsymbol{A})^\top = \boldsymbol{A}^\dagger\boldsymbol{A}$. The orthogonal projection matrix onto $\mathrm{Im}(\boldsymbol{A})$ is

$$\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})} = \boldsymbol{A}(\boldsymbol{A}^\top\boldsymbol{A})^\dagger\boldsymbol{A}^\top. \tag{2.1}$$

Eigenvalues and Condition Numbers. Given a square matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, let $\lambda_{\max}(\boldsymbol{A})$ be the maximum eigenvalue of $\boldsymbol{A}$ and $\lambda_{\min}(\boldsymbol{A})$ the minimum *non-zero* eigenvalue of $\boldsymbol{A}$. The condition number of $\boldsymbol{A}$ is defined as the ratio of these two values:

$$\kappa(\boldsymbol{A}) = \frac{\lambda_{\max}(\boldsymbol{A})}{\lambda_{\min}(\boldsymbol{A})}.$$

A symmetric matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is called *positive semi-definite (PSD)* if and only if $\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} \geq 0$ for all $\boldsymbol{x} \in \mathbb{R}^n$. Equivalently, $\boldsymbol{A}$ is PSD if and only if all its eigenvalues of $\boldsymbol{A}$ are non-negative.

For a non-square matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, the singular values of $\boldsymbol{A}$ are defined as the square roots of the eigenvalues of $\boldsymbol{A}^\top \boldsymbol{A}$, which is always symmetric and PSD. Let $\sigma_{\max}(\boldsymbol{A})$ denote the maximum singular value of $\boldsymbol{A}$ and $\sigma_{\min}(\boldsymbol{A})$ the minimum *non-zero* singular value of $\boldsymbol{A}$. If $\boldsymbol{A}$ is a symmetric PSD matrix, then its eigenvalues and singular values coincide.

For two symmetric matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$, we say $\boldsymbol{A} \succcurlyeq \boldsymbol{B}$ if $\boldsymbol{A} - \boldsymbol{B}$ is PSD. The *condition number of $\boldsymbol{A}$ relative to $\boldsymbol{B}$* is defined as:

$$\kappa(\boldsymbol{A}, \boldsymbol{B}) \overset{\text{def}}{=} \min \left\{ \frac{\alpha}{\beta} : \beta \cdot \boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})} \boldsymbol{B} \boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})} \preccurlyeq \boldsymbol{A} \preccurlyeq \alpha \cdot \boldsymbol{B} \right\},$$

where $\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}$ is the projection matrix as defined in Equation (2.1).

Fact 2.1.2. *Let $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$ be symmetric matrices such that $\boldsymbol{A} \preccurlyeq \boldsymbol{B}$. Then, for any $\boldsymbol{V} \in \mathbb{R}^{m\times n}$,*

$$\boldsymbol{V}\boldsymbol{A}\boldsymbol{V}^\top \preccurlyeq \boldsymbol{V}\boldsymbol{B}\boldsymbol{V}^\top.$$

Norms. Given a vector $\boldsymbol{x} \in \mathbb{R}^n$, let[1]

$$\|\boldsymbol{x}\|_1 = \sum_{i\in[n]} |\boldsymbol{x}(i)|, \qquad \|\boldsymbol{x}\|_\infty = \max_{i\in[n]} |\boldsymbol{x}(i)|, \qquad \|\boldsymbol{x}\|_2 = \sqrt{\sum_{i=1}^n \boldsymbol{x}(i)^2},$$

where $\|\boldsymbol{x}\|_2$ is called the *Euclidean norm.* When $\boldsymbol{M} \in \mathbb{R}^{n\times n}$ is a PSD, we also define

$$\|\boldsymbol{x}\|_{\boldsymbol{M}} = \sqrt{\boldsymbol{x}^\top \boldsymbol{M}\boldsymbol{x}}.$$

Next, for a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, we define its operator norm induced by a vector p-norm as

$$\|\boldsymbol{A}\|_p \overset{\text{def}}{=} \max_{\boldsymbol{x}\neq \boldsymbol{0}} \frac{\|\boldsymbol{A}\boldsymbol{x}\|_p}{\|\boldsymbol{x}\|_p}.$$

In particular, for $p = 1, \infty, 2$, we have

$$\|\boldsymbol{A}\|_1 = \max_{j\in[n]} \sum_{i\in[m]} |\boldsymbol{A}(i,j)|, \ \|\boldsymbol{A}\|_\infty = \max_{i\in[m]} \sum_{j\in[n]} |\boldsymbol{A}(i,j)|, \ \|\boldsymbol{A}\|_2 = \sqrt{\lambda_{\max}(\boldsymbol{A}^\top \boldsymbol{A})} = \sigma_{\max}(\boldsymbol{A}).$$

Moreover, we use $\|\boldsymbol{A}\|_{\max}$ to denote the maximum absolute entry of $\boldsymbol{A}$, i.e.,

$$\|\boldsymbol{A}\|_{\max} = \max_{i,j} |\boldsymbol{A}(i,j)|.$$

We define a function X that takes an arbitrary number of matrices $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{k_1}$, vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{k_2}$, and scalars $K_1, \ldots, K_{k_3}$ as arguments, and returns the maximum of the entries of all the arguments, i.e.,

$$\begin{aligned} &X(\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{k_1}, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_{k_2}, K_1, \ldots, K_{k_3}) \\ &\qquad = \max\left\{\|\boldsymbol{A}_1\|_{\max}, \ldots, \|\boldsymbol{A}_{k_1}\|_{\max}, \|\boldsymbol{b}_1\|_\infty, \ldots, \|\boldsymbol{b}_{k_2}\|_\infty, |K_1|, \ldots, |K_{k_3}|\right\}. \end{aligned}$$

[1] We sometimes use $\|\boldsymbol{x}\|_{\max}$ for $\|\boldsymbol{x}\|_\infty$ interchangeably.

2.2 Simplicial Homology

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-simplexes 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 simplexes 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 d dimensions as a d-complex. For $1 \leq k \leq d$, the *k-skeleton* of a d-complex $\mathcal{K}$ is the subcomplex consisting of all the simplexes of $\mathcal{K}$ of dimensions at most k. In particular, the 1-skeleton of $\mathcal{K}$ is a graph. A simplicial complex is *pure* if every maximal simplex (i.e., one not properly contained in any larger simplex in $\mathcal{K}$) has the same dimension. For example, a tetrahedron (together with all its faces) forms a 3-dimensional pure simplicial complex: the only maximal simplex in it is the tetrahedron itself, which is of dimension 3.

Embedding and Triangulation. A *piecewise linear embedding* of a 3-complex in $\mathbb{R}^3$ maps a 0-simplex to a point, a 1-simplex to a line segment, a 2-simplex to a triangle, and a 3-simplex to a tetrahedron. In addition, the interiors of the images of simplexes are disjoint and the boundary of each simplex is mapped to the appropriate simplexes. Such an embedding of a simplicial complex $\mathcal{K}$ defines an *underlying topological space* $\mathbb{K}$ – the union of the images of all the simplexes of $\mathcal{K}$. We say $\mathcal{K}$ is *convex* if $\mathbb{K}$ is convex. We say $\mathcal{K}$ *triangulates* a topological space $\mathbb{X}$ if $\mathbb{K}$ is homeomorphic to $\mathbb{X}$. A simplex σ of $\mathcal{K}$ is a *exterior* simplex if σ is contained in the boundary of $\mathbb{K}$. A connected component of exterior simplexes is called a *boundary component* of $\mathcal{K}$.

More concretely, if $\mathbb{K}$ is a 2-dimensional manifold, it can be triangulated by 2-complexes, 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. Refer to Figure 2.1 for 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 for each triangle is clockwise.

Boundary Operators. Given an oriented d-dimensional simplicial complex $\mathcal{K}$, for each $0 \leq i \leq d$, let $\mathcal{C}_k$ denote *k-chain*, which is a formal sum of the oriented k-simplexes in $\mathcal{K}$ with the coefficients over $\mathbb{R}$. We can define a sequence of boundary operators:

$$\mathcal{C}_d(\mathcal{K}) \xrightarrow{\partial_d} \mathcal{C}_{d-1}(\mathcal{K}) \xrightarrow{\partial_{d-1}} \cdots \xrightarrow{\partial_2} \mathcal{C}_1(\mathcal{K}) \xrightarrow{\partial_1} \mathcal{C}_0(\mathcal{K}),$$

where each linear map $\partial_k : \mathcal{C}_k(\mathcal{K}) \to \mathcal{C}_{k-1}(\mathcal{K})$ is a *boundary operator* that maps every k-simplex to a signed sum of its boundary $(k-1)$-faces. Specifically, 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 $|\mathcal{C}_{k-1}| \times |\mathcal{C}_k|$ dimensions, where $|\mathcal{C}_{k-1}|, |\mathcal{C}_k|$ is the number of $(k-1)$-simplexes and k-simplexes in $\mathcal{K}$, respectively. 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 2.1 and Equation (2.2) for an example.

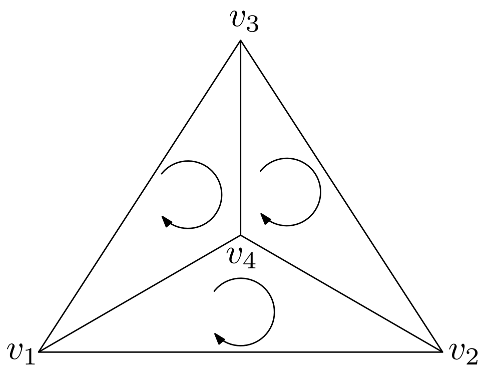

Figure 2.1: An example of boundary operator and oriented triangulation. We set a clockwise orientation for 2-simplexes, and set the orientation for 1-simplexes as the order of increasing vertex indices.

$$\partial_2 = \begin{array}{c} \\ [v_1,v_2] \\ [v_2,v_3] \\ [v_1,v_3] \\ [v_1,v_4] \\ [v_2,v_4] \\ [v_3,v_4] \end{array} \begin{array}{c} \begin{array}{ccc} [v_1,v_4,v_2] & [v_2,v_4,v_3] & [v_1,v_3,v_4] \end{array} \\ \left[\begin{array}{ccc} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \end{array}\right] \end{array}. \tag{2.2}$$

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

$$\partial_k \partial_{k+1} = \mathbf{0}. \tag{2.3}$$

This implies

$$\mathrm{Im}(\partial_{k+1}) \subseteq \mathrm{Ker}(\partial_k).$$

Thus, we can define the *quotient space* $H_k = \mathrm{Ker}(\partial_k) \setminus \mathrm{Im}(\partial_{k+1})$, referred to as the kth *homology space* of $\mathcal{K}$. The dimension of H_k is the kth *Betti number* of $\mathcal{K}$, denoted as β_k. In particular, if $\beta_k = 0$, then

$$\mathrm{Im}(\partial_k^\top) \oplus \mathrm{Im}(\partial_{k+1}) = \mathbb{R}^{|\mathcal{C}_k|}.$$

Betti numbers play an important role in understanding the homology spaces. Intuitively, the kth Betti number refers to the number of k-dimensional "holes" on a topological surface. For example, β_0 represents the number of connected components; β_1 represents the number of "circular" holes; β_2 represents the number of "voids" or "cavities".

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

Theorem 2.2.1 (Hodge Decomposition [Lim20])**.** *Let $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{B} \in \mathbb{R}^{n \times p}$ be matrices satisfying $\boldsymbol{AB} = \boldsymbol{0}$. Then, there is an orthogonal direct sum decomposition*

$$\mathbb{R}^n = Im(\boldsymbol{A}^\top) \oplus Ker(\boldsymbol{A}^\top \boldsymbol{A} + \boldsymbol{B}\boldsymbol{B}^\top) \oplus Im(\boldsymbol{B}).$$

By Equation (2.3), 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*:

$$\boldsymbol{L}_k \stackrel{\text{def}}{=} \partial_k^\top \partial_k + \partial_{k+1} \partial_{k+1}^\top.$$

In particular, $\boldsymbol{L}_0 = \partial_1 \partial_1^\top$ is the graph Laplacian. The kth homology space $H_k(\mathcal{K})$ is isomorphic to $\text{Ker}(\boldsymbol{L}_k)$, and thus the kth Betti number of $\mathcal{K}$ equals the dimension of $\text{Ker}(\boldsymbol{L}_k)$.

2.3 Approximately Solving Linear Equations

We define approximately solving linear equations in a general form, following [KZ17]. For more details, we refer the readers to Section 2.1 of [KZ17].

Definition 2.3.1 (Linear Equation Problem (LE))**.** Given a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, a vector $\boldsymbol{b} \in \mathbb{R}^m$, 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

$$\boldsymbol{x} \in \arg\min_{\boldsymbol{x} \in \mathbb{R}^n} \|\boldsymbol{Ax} - \boldsymbol{b}\|_2 .$$

The exact solution to LE has a closed-form solution, as shown in the following fact.

Fact 2.3.2. *Let $\boldsymbol{x}^* \in \arg\min_{\boldsymbol{x} \in \mathbb{R}^n} \|\boldsymbol{Ax} - \boldsymbol{b}\|_2$. Then,*

$$\boldsymbol{Ax}^* = \boldsymbol{A}(\boldsymbol{A}^\top \boldsymbol{A})^\dagger \boldsymbol{A}^\top \boldsymbol{b} = \boldsymbol{\Pi}_{Im(\boldsymbol{A})} \boldsymbol{b}$$

and

$$\|\boldsymbol{Ax}^* - \boldsymbol{b}\|_2^2 = \left\|(\boldsymbol{I} - \boldsymbol{\Pi}_{Im(\boldsymbol{A})})\boldsymbol{b}\right\|_2^2 ,$$

where $\boldsymbol{\Pi}_{Im(\boldsymbol{A})}$ is the projection matrix as defined in Equation (2.1).

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

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

Definition 2.3.3 (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

$$\left\|\boldsymbol{Ax} - \boldsymbol{\Pi}_{\text{Im}(\boldsymbol{A})} \boldsymbol{b}\right\|_2 \le \epsilon \left\|\boldsymbol{\Pi}_{\text{Im}(\boldsymbol{A})} \boldsymbol{b}\right\|_2 .$$

The following fact demonstrates that the approximate error in Definition 2.3.3 is equivalent to several error notions that are commonly used in solving linear equations.

Fact 2.3.4.

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

After introducing the approximate solution problem in Definition 2.3.3, it is important to understand how the approximate solution relates to the exact solution. The following fact establishes a relationship between the error of the approximate solution and the exact solution error.

Fact 2.3.5. *Let $\boldsymbol{x}$ be a solution to* LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon)$ *and $\boldsymbol{x}^*$ be the exact solution. Then,*

$$\left\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\right\|_2^2 \leq \left\|\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{b}\right\|_2^2 + \epsilon^2\left\|\boldsymbol{\Pi}_{Im(\boldsymbol{A})}\boldsymbol{b}\right\|_2^2.$$

Linear Equations in PSD Matrices. Consider the eigen-decomposition of a PSD rank-k matrix $\boldsymbol{A} = \sum_{1\leq i\leq k}\lambda_i\boldsymbol{u}_i\boldsymbol{u}_i^\top$, where $\lambda_i \geq 0$ are eigenvalues and $\boldsymbol{u}_i$ are the corresponding orthogonal eigenvectors. Let $\boldsymbol{A}^\dagger = \sum_{1\leq i\leq k}\lambda_i^{-1}\boldsymbol{u}_i\boldsymbol{u}_i^\top$ be the pseudo-inverse of $\boldsymbol{A}$, and we have

$$\boldsymbol{A}\boldsymbol{A}^\dagger = \boldsymbol{A}^\dagger\boldsymbol{A} = \sum_{1\leq i\leq k}\boldsymbol{u}_i\boldsymbol{u}_i^\top = \boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}.$$

Furthermore, let $\boldsymbol{A}^{1/2} = \sum_{1\leq i\leq k}\lambda_i^{1/2}\boldsymbol{u}_i\boldsymbol{u}_i^\top$ be the square root $\boldsymbol{A}$, and we have $\boldsymbol{A}^{1/2}\boldsymbol{A}^{1/2} = \boldsymbol{A}$. Similarly, let $\boldsymbol{A}^{\dagger/2} = \sum_{1\leq i\leq k}\lambda_i^{-1/2}\boldsymbol{u}_i\boldsymbol{u}_i^\top$ be square root of $\boldsymbol{A}^\dagger$. By Fact 2.1.1, we have $\mathrm{Im}(\boldsymbol{A}^{1/2}) = \mathrm{Im}(\boldsymbol{A})$, thus

$$\boldsymbol{A}^{1/2}\boldsymbol{A}^\dagger\boldsymbol{A}^{1/2} = \boldsymbol{A}^{1/2}\boldsymbol{A}^{\dagger/2} = \boldsymbol{A}^{\dagger/2}\boldsymbol{A}^{1/2} = \boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}. \tag{2.4}$$

It is easy to see that $\boldsymbol{A}^{1/2}, \boldsymbol{A}^\dagger, \boldsymbol{A}^{\dagger/2}$ are also PSD.

The following fact provides two useful results about the perturbation of PSD matrices, which will help analyze approximation errors and stability properties in such systems of linear equations.

Fact 2.3.6. *Let $\boldsymbol{A}, \boldsymbol{Z} \in \mathbb{R}^{n\times n}$ be two symmetric PSD matrices.*

1. *If $(1-\epsilon)\boldsymbol{A}^\dagger \preccurlyeq \boldsymbol{Z} \preccurlyeq (1+\epsilon)\boldsymbol{A}^\dagger$, then for any $\boldsymbol{b} \in Im(\boldsymbol{A})$,*

$$\left\|\boldsymbol{A}\boldsymbol{Z}\boldsymbol{b} - \boldsymbol{b}\right\|_2 \leq \epsilon\sqrt{\kappa(\boldsymbol{A})}\left\|\boldsymbol{b}\right\|_2.$$

2. *If $\left\|\boldsymbol{A}\boldsymbol{Z}\boldsymbol{b} - \boldsymbol{b}\right\|_2 \leq \epsilon\left\|\boldsymbol{b}\right\|_2$ for any $\boldsymbol{b} \in Im(\boldsymbol{A})$, then*

$$(1-\epsilon)\boldsymbol{A}^\dagger \preccurlyeq \boldsymbol{\Pi}_{Im(\boldsymbol{A})}\boldsymbol{Z}\boldsymbol{\Pi}_{Im(\boldsymbol{A})} \preccurlyeq (1+\epsilon)\boldsymbol{A}^\dagger.$$

Schur Complement. Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, and let $F \cup C$ be a partition of $\{1, \dots, n\}$. We write $\boldsymbol{A}$ as a block matrix:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}(F,F) & \boldsymbol{A}(F,C) \\ \boldsymbol{A}(C,F) & \boldsymbol{A}(C,C) \end{bmatrix}. \tag{2.5}$$

We define the *(generalized) Schur complement of* $\boldsymbol{A}$ *onto* C to be

$$\mathrm{Sc}(\boldsymbol{A})_C = \boldsymbol{A}(C,C) - \boldsymbol{A}(C,F)\boldsymbol{A}(F,F)^{\dagger}\boldsymbol{A}(F,C).$$

The Schur complement appears in performing a block Gaussian Elimination on matrix $\boldsymbol{A}$ to eliminate the indices in F.

Fact 2.3.7. *For any fixed vector* $\boldsymbol{y}$,

$$\min_{\boldsymbol{x}} \begin{bmatrix} \boldsymbol{x}^\top & \boldsymbol{y}^\top \end{bmatrix} \boldsymbol{A} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} = \boldsymbol{y}^\top Sc(\boldsymbol{A})_C \boldsymbol{y}.$$

Using Fact 2.3.7, we can obtain the following fact that Schur complement preserves PSD.

Fact 2.3.8. *Let* $\boldsymbol{A}$ *be a PSD matrix defined in Equation* (2.5). *Then,* $Sc(\boldsymbol{A})_C$ *is also PSD.*

The following fact provides a useful matrix identity that factorizes the matrix $\boldsymbol{A}$ in terms of its submatrices and Schur complement.

Fact 2.3.9. *Let* $\boldsymbol{A}$ *be a PSD matrix defined in Equation* (2.5). *Then,*

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{I} & \\ \boldsymbol{A}(C,F)\boldsymbol{A}(F,F)^{\dagger} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{A}(F,F) & \\ & Sc(\boldsymbol{A})_C \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{A}(F,F)^{\dagger}\boldsymbol{A}(F,C) \\ & \boldsymbol{I} \end{bmatrix}.$$

In the special case where $\boldsymbol{A} = \boldsymbol{B}\boldsymbol{B}^\top$, the Schur complement $\mathrm{Sc}(\boldsymbol{A})_C$ can be expressed directly in terms of $\boldsymbol{B}$.

Fact 2.3.10. *Let* $\boldsymbol{A}$ *be a PSD matrix defined in Equation* (2.5). *Let* $\boldsymbol{A} = \boldsymbol{B}\boldsymbol{B}^\top$, *and we decompose* $\boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}_F \\ \boldsymbol{B}_C \end{bmatrix}$ *accordingly. Then,*

$$Sc(\boldsymbol{A})_C = \boldsymbol{B}_C \boldsymbol{\Pi}_{Ker(\boldsymbol{B}_F)} \boldsymbol{B}_C^\top,$$

where $\boldsymbol{\Pi}_{Ker(\boldsymbol{B}_F)}$ *is the projection onto the kernel of* $\boldsymbol{B}_F$: $\boldsymbol{\Pi}_{Ker(\boldsymbol{B}_F)} = \boldsymbol{I} - \boldsymbol{B}_F^\top \left(\boldsymbol{B}_F \boldsymbol{B}_F^\top\right)^{\dagger} \boldsymbol{B}_F$.

Chapter 3

Hardness Results for Two-Commodity Flow

This chapter is based on [DKZ22] and focuses on proving Theorem 1.4.1. Here in Section 3.5 and 3.6, we improve the proof structure from the published version [DKZ22], making it modular and streamlined. Furthermore, we introduce a unified linear programming (LP) transformation framework in Section 3.7. This framework provides a unified characterization and a systematic analysis of various reduction steps.

3.1 Prior Works

Previous work by Kyng and Zhang [KZ17] had shown that fast algorithms for multi-commodity flow were unlikely to arise from combining interior point methods with special-purpose linear equation solvers. Concretely, they showed that the linear equations that arise in interior point methods for multi-commodity flow are as hard to solve as arbitrary linear equations. This ruled out algorithms following the pattern of the known fast algorithms for high-accuracy single-commodity flow problems. However, it left open the broader question if some other families of algorithms could succeed. We now show that, in general, a separation between multi-commodity flow and linear programming is not possible.

Our paper follows the outline of Itai's polynomial-time reduction of a linear program to a 2-commodity flow problem. In addition, it is also inspired by recent works on hardness for structured linear equations [KZ17] and packing/covering LPs [KWZ20], which focused on obtaining nearly-linear time reductions in related settings. These works in turn were motivated by the last decade's substantial progress on fine-grained complexity for a range of polynomial time solvable problems, e.g. see [WW18]. Also notable is the result by Musco et al. [Mus+19] on hardness for matrix spectrum approximation.

3.2 Our Contributions

The many successes in developing high-accuracy algorithms for single-commodity flow problems highlight an important open question: Can multi-commodity flow be solved to high accuracy faster than general linear programs? We rule out this possibility by proving that any linear program (assuming it is polynomially bounded and has integer entries) can be encoded as a multi-commodity flow problem in nearly-linear time. This implies that any improvement in the running time of (high-accuracy) algorithms for sparse multi-commodity flow problems would directly translate to a faster algorithm for solving sparse linear programs to high accuracy, with only a polylogarithmic increase in running time.

Theorem 1.4.1 demonstrates several key improvements over Itai's original polynomial time reduction from LP to 2CF. Firstly, while Itai produced a 2CF with the number of edges on the order of $\Theta\left(N^2 \log^2 X\right)$, where N is the number of non-zeros and X is the magnitude of the largest entry in the original LP, we show that an improved gadget can reduce this to $O\left(N \log X\right)$. Thus, in the case of polynomially bounded linear programs, where $\log X = O(\log N)$, we get only a polylogarithmic multiplicative increase in the number of non-zero entries from N to $\widetilde{O}(N)$, whereas Itai had an increase in the number of non-zeros by a factor $\widetilde{O}(N)$, i.e., from N to $\widetilde{O}(N^2)$.

Secondly, Itai used very large graph edge capacities that require $O\left((N \log X)^{1.01}\right)$ many bits *per edge*, letting the capacities grow exponentially given an LP with polynomially bounded entries. We show that when the feasible polytope radius R is bounded, we can ensure capacities remain a polynomial function of the initial parameters N, R, and X. In the important case of polynomially bounded linear programs, this means the capacities stay polynomially bounded.

Thirdly, while Itai only analyzed the chain of reductions under the case with exact solutions, we generalize the analysis to the case with approximate solutions by establishing an error analysis along the chain of reductions. We show that the error only grows polynomially during the reduction.

Finally, we introduce a framework that unifies all steps in the reduction chain. Rather than analyzing each reduction individually, this framework offers a unified characterization of problem reductions and enables a systematic approach to analyzing the correctness and error propagation of these reductions. It serves not only as a powerful tool for validating existing reductions but also as a valuable guide for designing and verifying new reduction steps within the LP realm.

3.3 Problem Definitions

This section introduces the intermediate problems that form our nine-step chain of reduction algorithms. We group these problems into two categories for clarity: the *Algebra Space*, which includes linear programming and linear equation problems, and the *Flow Space*, which involves flow routing in graphs. We also define the error notions used to evaluate approximate solutions in both domains. Before delving into the formal problem definitions, we begin by discussing the key assumptions involved.

3.3.1 Discussion of the Problem Assumptions

Polynomially-Bounded LP. Current research on fast algorithms for solving linear programs generally relies on assuming bounds on: (1) the size of the program entries, and (2) the norm of all feasible solutions. Generally, the algorithm running time depends logarithmically on these quantities, and hence to make these factors negligible, entry size and feasible solution norms are assumed to be polynomially bounded, for example in [CLS21; Jia+21]. We will refer to a linear program satisfying these assumptions as *polynomially bounded*. More precisely, we say a linear program with N non-zero coefficients is *polynomially bounded* if it has coefficients in the range $[-X, X]$ and $\|\boldsymbol{x}\|_1 \leq R$

for all feasible $\boldsymbol{x}$ (i.e., the polytope of feasible solutions has radius of R in ℓ_1 norm), and $X, R \leq O(N^c)$ for some constant c. In fact, if there exists a feasible solution $\boldsymbol{x}$ satisfying $\|\boldsymbol{x}\|_1 \leq R$, then we can add a constraint $\|\boldsymbol{x}\|_1 \leq R$ to the LP (which can be rewritten as linear inequality constraints) so that in the new LP, all feasible solutions have ℓ_1 norm at most R. This only increases the number of non-zeros in the LP by at most a constant factor.

Interior Point Methods and Reductions with Fixed Point Arithmetic. Modern fast interior point methods for linear programming, such as [CLS21], are analyzed in the RealRAM model. In order to implement these algorithms using *fixed point arithmetic* with polylogarithmic bit complexity per number, instead of RealRAM, additional assumptions are required. For example, this class of algorithms relies on computing matrix inverses, and these must be approximately representable using polylogarithmic bit complexity per entry. This is not possible, if the inverses have exponentially large entries, which may occur even in polynomially bounded linear programs. For example, consider a linear program feasibility problem $\{\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$, with constraint matrix $\boldsymbol{A} \in \mathbb{R}^{2n \times 2n}$ given by

$$\boldsymbol{A}(i,j) = \begin{cases} 1 & \text{if } i < n \text{ and } i = j \\ -2 & \text{if } i < n \text{ and } i+1 = j \\ 2 & \text{if } i \geq n \text{ and } i = j \\ -1 & \text{if } i \geq n \text{ and } i+1 = j \\ 0 & \text{o.w.} \end{cases}.$$

Such a linear program is polynomially bounded for many choices of $\boldsymbol{b}$, e.g., $\boldsymbol{b} = \boldsymbol{e}_{2n}$. Unfortunately, for the vector $\boldsymbol{x} \in \mathbb{R}^{2n}$ given by

$$\boldsymbol{x}(i) = \begin{cases} 2^{-i-1} & \text{if } i \leq n \\ 0 & \text{o.w.} \end{cases},$$

we have $\boldsymbol{A}\boldsymbol{x} = 2^{-n}\boldsymbol{e}_n$, and from this one can see that $\boldsymbol{A}^{-1}$ must have entries of size at least $\Omega(2^n/n)$. This will cause algorithms such as [CLS21] to perform intermediate calculations with n bits per number, increasing the running time by a factor roughly n.

Modern interior point methods can be translated to fixed precision arithmetic with various different assumptions leading to different per entry bit complexity (see [CLS21] for a discussion of one standard sufficient condition). Furthermore, if the problem has polynomially bounded condition number (when appropriately defined), then we expect that polylogarithmic bit complexity per entry should suffice, at least for highly accurate approximate solutions, by relying on fast stable numerical linear algebra [DDH07], although we are not aware of a complete analysis of this translation.

If a linear program with integer entries is solved to sufficiently small additive error, the approximate solution can be converted into an exact solution, e.g. see [Ren88; LS14; CLS21] for a discussion of the necessary precision and for a further discussion of numerical stability properties of interior point methods. Polynomially bounded linear programs with integer coefficients may still require exponentially small additive error for this rounding to succeed.

We give a reduction from general linear programming to 2-commodity flow, and like [CLS21], we assume the program is polynomially bounded to carry out the reduction. We also use an additional assumption, namely that the linear program is written using integral entries.[1] We do not make additional assumptions about polynomially bounded condition number of the problem. This means we can apply our reduction to programs such as the one above, despite [CLS21] not obtaining a reasonable running time on such programs using fixed point arithmetic.

Our analysis of our reduction uses the RealRAM model like [CLS21; Jia+21] and other modern interior point method analysis, however, it should be straightforward to translate our reduction and error analysis to fixed point arithmetic with polylogarithmically many bits, because all our mappings are simple linear transformations, and we never need to compute or apply a matrix inverse.

Rounding Linear Programs to Have Integer Entries. It is possible to give some fairly general and natural sufficient conditions for when a polynomially bounded linear program can be rounded to have integral entries, one example of this is having a polynomially bounded *Renegar's condition number*. Renegar introduced this condition number for linear programs in [Ren95]. For a given linear program, suppose that perturbing the entries of the program by at most δ each does not change the feasibility of the the linear program, and let δ^* be the largest such δ. Let U denote the maximum absolute value of entries in the linear program. Then $\kappa = U/\delta^*$ is Renegar's condition number for the linear program.

Suppose we are given a polynomially bounded linear program $\max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$ (also referred to as $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$), with polytope radius at most R, and Renegar's condition number κ is also bounded by a polynomial. We wish to compute a vector $\boldsymbol{x} \geq \boldsymbol{0}$ with an ϵ additive error on each constraint and in the optimal value. We can reduce this problem, for instance, $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$ to a polynomially bounded linear program instance with integral input numbers. Specifically, we round the entries of $\boldsymbol{A}$ down to $\widetilde{\boldsymbol{A}}$ and those of $\boldsymbol{b}, \boldsymbol{c}$ up to $\tilde{\boldsymbol{b}}, \tilde{\boldsymbol{c}}$ all by at most $\min\{\frac{\epsilon}{3R}, \frac{U}{\kappa R}\}$ such that each entry of $\widetilde{\boldsymbol{A}}, \tilde{\boldsymbol{b}}, \tilde{\boldsymbol{c}}$ only needs a logarithmic number of bits. Suppose $\tilde{\boldsymbol{x}}$ is a solution to $(\widetilde{\boldsymbol{A}}, \tilde{\boldsymbol{b}}, \tilde{\boldsymbol{c}})$ with $\frac{\epsilon}{3}$ additive error on each constraint and the optimal value. Then,

$$\begin{aligned} \boldsymbol{A}\tilde{\boldsymbol{x}} &= \widetilde{\boldsymbol{A}}\tilde{\boldsymbol{x}} + (\boldsymbol{A} - \widetilde{\boldsymbol{A}})\tilde{\boldsymbol{x}} \leq \boldsymbol{b} + \epsilon\boldsymbol{1}, \\ \boldsymbol{c}^\top \tilde{\boldsymbol{x}} &\geq \tilde{\boldsymbol{c}}^\top \tilde{\boldsymbol{x}}^* - \frac{2\epsilon}{3}. \end{aligned}$$

Here, $\boldsymbol{1}$ is the all-one vector, $\tilde{\boldsymbol{x}}^*$ is an optimal solution to $(\widetilde{\boldsymbol{A}}, \tilde{\boldsymbol{b}}, \tilde{\boldsymbol{c}})$. In addition, the optimal value of $(\widetilde{\boldsymbol{A}}, \tilde{\boldsymbol{b}}, \tilde{\boldsymbol{c}})$ is greater than or equal to that of $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$. So, $\tilde{\boldsymbol{x}}$ is a solution to $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$ with ϵ additive error as desired. Since each entry of $\widetilde{\boldsymbol{A}}, \tilde{\boldsymbol{b}}, \tilde{\boldsymbol{c}}$ has a logarithmic number of bits, we can scale all of them to polynomially bounded integers without changing the feasible set and the optimal solutions. Thus we see that if we restrict ourselves to polynomially linear programs with polynomially bounded Renegar's condition number, and we wish to solve the program with small additive error, we can assume, without loss of generality, that the program has integer coefficients.

3.3.2 Problems in Algebra Space

For the convenience of our reduction, we define linear programming as a "decision" problem. We can solve the optimization problem $\max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$ by binary

[1] W.l.o.g., by scaling, this is the same as assuming the program is written with polynomially bounded fixed precision numbers of the form k/D where k is an integer, and D is an integral denominator shared across all entries, and both k and D are polynomially bounded.

searching its optimal value via the decision problem.

Definition 3.3.1 (Linear Programming (LP))**.** Given a matrix $\boldsymbol{A} \in \mathbb{Z}^{m \times n}$, vectors $\boldsymbol{b} \in \mathbb{Z}^m$ and $\boldsymbol{c} \in \mathbb{Z}^n$, an integer K, and $R \geq \max\{1, \max\{\|\boldsymbol{x}\|_1 : \boldsymbol{Ax} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}\}$, we refer to the LP problem for $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, K, R)$ as the problem of finding a vector $\boldsymbol{x} \in \mathbb{R}^n_{\geq 0}$ satisfying

$$\boldsymbol{Ax} \leq \boldsymbol{b} \text{ and } \boldsymbol{c}^\top \boldsymbol{x} \geq K$$

if such an $\boldsymbol{x}$ exists and returning "infeasible" otherwise.

We will reduce linear programs to linear equations with non-negative variables (LEN), and then to linear equations with non-negative variables and small integer coefficients (k-LEN).

Definition 3.3.2 (Linear Equations with Non-negative Variables (LEN))**.** Given $\boldsymbol{A} \in \mathbb{Z}^{m \times n}, \boldsymbol{b} \in \mathbb{Z}^m$, and $R \geq \max\{1, \max\{\|\boldsymbol{x}\|_1 : \boldsymbol{Ax} = \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}\}$, we refer to the LEN problem for $(\boldsymbol{A}, \boldsymbol{b}, R)$ as the problem of finding a vector $\boldsymbol{x} \in \mathbb{R}^n_{\geq 0}$ satisfying $\boldsymbol{Ax} = \boldsymbol{b}$ if such an $\boldsymbol{x}$ exists and returning "infeasible" otherwise.

Definition 3.3.3 (k-LEN (k-LEN))**.** The k-LEN problem is an LEN problem $(\boldsymbol{A}, \boldsymbol{b}, R)$ where the entries of $\boldsymbol{A}$ are integers in $[-k, k]$ for some given $k \in \mathbb{Z}_{>0}$.

We employ the following additive error notion. We append a letter "A" to each problem name to denote its approximation version, e.g., LP Approximate Problem is abbreviated to LPA.

Definition 3.3.4 (Error Notions in the Algebra Space)**.** We always require $\boldsymbol{x} \geq \boldsymbol{0}$. In addition,

- The inequality constraint $\boldsymbol{Ax} \leq \boldsymbol{b}$ is relaxed to $\boldsymbol{Ax} - \boldsymbol{b} \leq \epsilon \boldsymbol{1}$, where $\boldsymbol{1}$ is the all-1 vector;
- The equality constraint $\boldsymbol{Ax} = \boldsymbol{b}$ is relaxed to $\|\boldsymbol{Ax} - \boldsymbol{b}\|_\infty \leq \epsilon$.

3.3.3 Problems in Flow Space

A *flow network* is a directed graph $G = (V, E)$, where V is the set of vertices and $E \subseteq V \times V$ is the set of edges, together with a vector of edge capacities $\boldsymbol{u} \in \mathbb{Z}^{|E|}_{>0}$ that upper bounds the amount of flow passing each edge. A *2-commodity flow network* is a flow network together with two source-sink pairs $s_i, t_i \in V$ for each commodity $i \in \{1, 2\}$.

Definition 3.3.5 (Capacity Constraints)**.** Given a 2-commodity flow network, we say the flows $\boldsymbol{f}_1, \boldsymbol{f}_2 \geq \boldsymbol{0}$ satisfy *capacity constraints* if

$$\boldsymbol{f}_1(e) + \boldsymbol{f}_2(e) \leq \boldsymbol{u}(e), \quad \forall e \in E.$$

Definition 3.3.6 (Flow Conservation Constraints)**.** Given a 2-commodity flow network, we say the flows $\boldsymbol{f}_1, \boldsymbol{f}_2 \geq \boldsymbol{0}$ satisfy *flow conservation constraints* if

$$\sum_{u:(u,v) \in E} \boldsymbol{f}_i(u, v) = \sum_{w:(v,w) \in E} \boldsymbol{f}_i(v, w), \quad \forall i \in \{1, 2\},\ v \in V \setminus \{s_i, t_i\}.$$

Given a 2-commodity flow network $(G = (V, E), \boldsymbol{u}, s_1, t_1, s_2, t_2)$, a *feasible 2-commodity flow* is a pair of flows $\boldsymbol{f}_1, \boldsymbol{f}_2 \in \mathbb{R}^{|E|}_{\geq 0}$ that satisfies both the capacity and flow conservation constraints.

In particular, in a *1-commodity flow network*, capacity constraints refer to $\mathbf{0} \leq \boldsymbol{f}(e) \leq \boldsymbol{u}(e),\ \forall e \in E$; and flow conservation constraints become $\sum_{u:(u,v)\in E} \boldsymbol{f}(u, v) = \sum_{w:(v,w)\in E} \boldsymbol{f}(v, w), v \in V \setminus \{s, t\}$. A *feasible 1-commodity flow* is a flow $\boldsymbol{f} \in \mathbb{R}^{|E|}_{\geq 0}$ that satisfies both the capacity and flow conservation constraints.

Similar to the definition of LP, we define 2-commodity flow problem as a decision problem. The corresponding optimization problem can be solved by reducing it to a sequence of decision problems.

Definition 3.3.7 (2-Commodity Flow Problem (2CF))**.** Given a 2-commodity flow network $(G, \boldsymbol{u}, s_1, t_1, s_2, t_2)$ together with $R \geq 0$, we refer to the 2CF problem for $(G, \boldsymbol{u}, s_1, t_1, s_2, t_2, R)$ as the problem of finding a feasible 2-commodity flow $\boldsymbol{f}_1, \boldsymbol{f}_2 \geq \mathbf{0}$ satisfying

$$F_1 + F_2 \geq R$$

if such flows exist and returning "infeasible" otherwise.

A closely related flow problem to 2CF is 2CFR, which imposes an additional requirement that a specific amount of flow must be routed for each commodity.

Definition 3.3.8 (2-Commodity Flow with Required Flow Amount (2CFR))**.** Given a 2-commodity flow network $(G, \boldsymbol{u}, s_1, t_1, s_2, t_2)$ together with $R_1, R_2 \geq 0$, we refer to the 2CFR for $(G, \boldsymbol{u}, s_1, t_1, s_2, t_2, R_1, R_2)$ as the problem of finding a feasible 2-commodity flow $\boldsymbol{f}_1, \boldsymbol{f}_2 \geq \mathbf{0}$ satisfying

$$F_1 \geq R_1, \quad F_2 \geq R_2$$

if such flows exist and returning "infeasible" otherwise.

Reducing LP to 2CF needs to go through a sequence of variants of flow problems with additional constraints beyond capacity and flow conservation constraints.

Definition 3.3.9 (Homologous Flow Constraints)**.** Given a collection of disjoint subsets of edges $\mathcal{H} = \{H_1, \ldots, H_h\}$ in a 1-commodity flow network, we say the flow $\boldsymbol{f} \geq \mathbf{0}$ satisfies *homologous flow constraints on* $\mathcal{H}$ if

$$\boldsymbol{f}(e_1) = \boldsymbol{f}(e_2), \quad \forall e_1, e_2 \in H_k, \quad \forall H_k \in \mathcal{H}.$$

If $|H_k| = 2$ for all $H_k \in \mathcal{H}$, we say $\boldsymbol{f}$ satisfy *pairwise homologous flow constraints.*

Definition 3.3.10 (Fixed Flow Constraints)**.** Given a set $F \subseteq E$ in a 2-commodity flow network, we say the flows $\boldsymbol{f}_1, \boldsymbol{f}_2 \geq \mathbf{0}$ satisfy *fixed flow constraints on* F if

$$\boldsymbol{f}_1(e) + \boldsymbol{f}_2(e) = \boldsymbol{u}(e), \quad \forall e \in F.$$

Similarly, given a set $F \subseteq E$ in a 1-commodity flow network, we say the flow $\boldsymbol{f} \geq \mathbf{0}$ satisfies *fixed flow constraints on* F if

$$\boldsymbol{f}(e) = \boldsymbol{u}(e), \quad \forall e \in F.$$

Definition 3.3.11 (Selective Flow Constraints)**.** Given two disjoint subsets of edges $S_1, S_2 \subseteq E$ in a 2-commodity flow network, we say the flows $\boldsymbol{f}_1, \boldsymbol{f}_2 \geq \boldsymbol{0}$ satisfy *selective flow constraints on* $S_1 \cup S_2$ if

$$\boldsymbol{f}_i(e) > 0, \quad \boldsymbol{f}_{\bar{i}} = 0, \quad \forall e \in S_i,\ i \in \{1,2\},\ \bar{i} \neq i.$$

Intermediate problems are defined with some combinations of the above additional flow constraints.

Definition 3.3.12 (Fixed Homologous Flow Problem (FHF))**.** Given a flow network with a single source-sink pair $(G, \boldsymbol{u}, s, t)$ together with a collection of disjoint subsets of edges $\mathcal{H} = \{H_1, \ldots, H_h\}$ and a subset of edges $F \subseteq E$ such that F is disjoint from all the sets in $\mathcal{H}$, we refer to the FHF problem for $(G, F, \mathcal{H}, \boldsymbol{u}, s, t)$ as the problem of finding a feasible flow $\boldsymbol{f} \geq \boldsymbol{0}$ such that $\boldsymbol{f}$ satisfies the homologous constraints on $\mathcal{H}$, and satisfies fixed flow constraints on F, if such flows exist, and returning "infeasible" otherwise.

Definition 3.3.13 (Fixed Pair Homologous Flow Problem (FPHF))**.** An FPHF is an FHF problem $(G, F, \mathcal{H}, \boldsymbol{u}, s, t)$ where every set in $\mathcal{H}$ has size 2.

Definition 3.3.14 (Selective Fixed Flow Problem (SFF))**.** Given a 2-commodity network $(G, \boldsymbol{u}, s_1, t_1, s_2, t_2)$ together with three edge sets $F, S_1, S_2 \subseteq E$, we refer to the SFF problem for $(G, F, S_1, S_2, \boldsymbol{u}, s_1, t_1, s_2, t_2)$ as the problem of finding a feasible 2-commodity flow $\boldsymbol{f}_1, \boldsymbol{f}_2 \geq \boldsymbol{0}$ such that $\boldsymbol{f}_1, \boldsymbol{f}_2$ satisfy the selective flow constraints on S_1, S_2, and satisfy the fixed flow constraints on F, if such flows exist, and returning "infeasible" otherwise.

Definition 3.3.15 (2-Commodity Fixed Flow Problem (2CFF))**.** Given a 2-commodity flow network $(G, \boldsymbol{u}, s_1, t_1, s_2, t_2)$ together with a subset of edges $F \subseteq E$, we refer to the 2CFF problem for the tuple $(G, F, \boldsymbol{u}, s_1, t_1, s_2, t_2)$ as the problem of finding a feasible 2-commodity flow $\boldsymbol{f}_1, \boldsymbol{f}_2 \geq \boldsymbol{0}$ which also satisfies the fixed flow constraints on F if such flows exist and returning "infeasible" otherwise.

Now, we define error notions for the above flow problems.

Definition 3.3.16 (Error Notions in the Flow Space)**.** We always require flows $\boldsymbol{f}_1, \boldsymbol{f}_2 \geq \boldsymbol{0}$.[2] In addition,

1. *Error in congestion.*
   - Relax the capacity constraints: $\boldsymbol{f}_1(e) + \boldsymbol{f}_2(e) \leq \boldsymbol{u}(e) + \epsilon, \quad \forall e \in E$;
   - Relax the fixed flow constraints: $|\boldsymbol{f}_1(e) + \boldsymbol{f}_2(e) - \boldsymbol{u}(e)| \leq \epsilon, \quad \forall e \in F$.
2. *Error in demand.*
   - Relax the flow conservation constraints for vertices other than the source and sink:

$$\left| \sum_{u:(u,v)\in E} \boldsymbol{f}_i(u,v) - \sum_{w:(v,w)\in E} \boldsymbol{f}_i(v,w) \right| \leq \epsilon,\ \forall v \in V \backslash \{s_i, t_i\},\ i \in \{1,2\}.$$

[2]Let $\boldsymbol{f}_2 = \boldsymbol{0}$ in 1-commodity flow problems.

- Relax the amount of flow requirements for source and sink nodes:

$$\sum_{w:(s_i,w)\in E} \boldsymbol{f}_i(s_i,w) \geq F_i - \epsilon, \qquad \sum_{u:(u,t_i)\in E} \boldsymbol{f}_i(u,t_i) \geq F_i - \epsilon, \quad i \in \{1,2\}.$$

3. *Error in selection.*
   Relax the selective flow constraints: $\boldsymbol{f}_{\bar{i}}(e) \leq \epsilon, \quad \forall e \in S_i, \ \bar{i} \neq i.$

4. *Error in (pairwise) homology.*
   Relax the (pairwise) homologous flow constraints: $|\boldsymbol{f}(e_1) - \boldsymbol{f}(e_2)| \leq \epsilon, \ \forall e_1, e_2 \in H_k, H_k \in \mathcal{H}$.

We remark that the error notions in the flow space are essentially equivalent to those in the algebra space, as defined in Definition 3.3.4. For instance, relaxing capacity constraints corresponds to relaxing inequality constraints, while relaxing flow conservation constraints corresponds to relaxing equality constraints.

3.4 Main Results

Before formally stating our main results, we first formalize the approximate versions of LP and 2CF by applying the error notions defined earlier in Definition 3.3.4 and 3.3.16.

Definition 3.4.1 (LP Approximate Problem (LPA))**.** An LPA instance is given by an LP instance $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, K, R)$ and an error parameter $\epsilon \in [0,1]$, which we collect in a tuple $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, K, R, \epsilon)$. We say an algorithm solves the LPA problem, if, given any LPA instance, it returns a vector $\boldsymbol{x} \geq \boldsymbol{0}$ such that

$$\begin{aligned} \boldsymbol{c}^\top \boldsymbol{x} &\geq K - \epsilon, \\ \boldsymbol{A}\boldsymbol{x} &\leq \boldsymbol{b} + \epsilon \boldsymbol{1}, \end{aligned}$$

where $\boldsymbol{1}$ is the all-1 vector, or it correctly declares that the associated LP instance is infeasible.

Definition 3.4.2 (2CF Approximate Problem (2CFA))**.** A 2CFA instance is given by a 2CF instance $(G, \boldsymbol{u}, s_1, t_1, s_2, t_2, R)$ and an error parameter $\epsilon \in [0,1]$, which we collect in a tuple $(G, \boldsymbol{u}, s_1, t_1, s_2, t_2, R, \epsilon)$. We say an algorithm solves the 2CFA problem, if, given any 2CFA instance, it returns a pair of flows $\boldsymbol{f}_1, \boldsymbol{f}_2 \geq 0$ that satisfies

$$\boldsymbol{f}_1(e) + \boldsymbol{f}_2(e) \leq \boldsymbol{u}(e) + \epsilon, \ \forall e \in E, \qquad \text{(error in congestion)}$$

$$\left| \sum_{u:(u,v)\in E} \boldsymbol{f}_i(u,v) - \sum_{w:(v,w)\in E} \boldsymbol{f}_i(v,w) \right| \leq \epsilon, \ \forall v \in V \backslash \{s_i, t_i\}, i \in \{1,2\}, \quad \text{(error in demand)}$$

$$\left| \sum_{w:(s_i,w)\in E} \boldsymbol{f}_i(s_i,w) - F_i \right| \leq \epsilon, \qquad \left| \sum_{u:(u,t_i)\in E} \boldsymbol{f}_i(u,t_i) - F_i \right| \leq \epsilon, \ i \in \{1,2\},$$

$$\text{(error in demand)}$$

where $F_1 + F_2 = R$;[3] or it correctly declares that the associated 2CFinstance is infeasible.

[3]If we encode 2CF as an LP instance, and approximately solve the LP with at most ϵ additive error. Then, the approximate solution also agrees with the error notions of 2CF, except that we get $F_1 + F_2 \geq R - \epsilon$ instead of $F_1 + F_2 \geq R$. This inconsistency can be eliminated by setting $\epsilon' = 2\epsilon$, and slightly adjusting F_1, F_2 to F_1', F_2' such that $F_1' + F_2' \geq R$. This way, we obtain an approximate solution to 2CF with at most ϵ' additive error.

With the approximate versions of LP and 2CF defined, we now present the main result formally, which provides a fine-grained complexity on the problem size and error tolerance in the reduction from LPA to 2CFA.

Theorem 3.4.3 (Main Theorem). *Given an* LPA *instance* $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, K, R, \epsilon^{lp})$ *where* $\boldsymbol{A} \in \mathbb{Z}^{m\times n}, \boldsymbol{b} \in Z^m, \boldsymbol{c} \in \mathbb{Z}^n, K \in \mathbb{Z}, \epsilon^{lp} \geq 0$ *and* $\boldsymbol{A}$ *has* $\mathrm{nnz}\,(\boldsymbol{A})$ *non-zero entries, we can reduce it to a* 2CFA *instance* $(G = (V,E), \boldsymbol{u}, s_1, t_1, s_2, t_2, R^{2cf}, \epsilon^{2cf})$ *in time* $O(\mathrm{nnz}(\boldsymbol{A}) \log X)$ *where* $X = X(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, K)$*, such that*

$$\begin{aligned} &|V|, |E| = O(\mathrm{nnz}(\boldsymbol{A}) \log X), \\ &\|\boldsymbol{u}\|_{\max}, R^{2cf} = O(\mathrm{nnz}^3(\boldsymbol{A}) R X \log^2 X), \\ &\epsilon^{2cf} = \Omega\left(\frac{1}{\mathrm{nnz}^6(\boldsymbol{A}) R X \log^6 X}\right)\epsilon^{lp}. \end{aligned}$$

If the LP *instance* $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, K, R)$ *has a solution, then the* 2CF *instance* $(G^{2cf}, \boldsymbol{u}^{2cf}, s_1, t_1, s_2, t_2, R^{2cf})$ *has a solution. Furthermore, if* $\boldsymbol{f}^{2cf}$ *is a solution to the* 2CFA *(*2CF*) instance, then in time* $O(\mathrm{nnz}(\boldsymbol{A}) \log X)$*, we can compute a solution* $\boldsymbol{x}$ *to the* LPA *(*LP*, respectively) instance, where the exact case holds when* $\epsilon^{2cf} = \epsilon^{lp} = 0$*.*

Our main theorem immediately implies the following corollary.

Corollary 3.4.4. *If we can solve any* 2CFA *instance* $(G = (V,E), \boldsymbol{u}, s_1, t_1, s_2, t_2, R^{2cf}, \epsilon)$ *in time* $O\left(|E|^c \operatorname{poly}\log\left(\frac{\|\boldsymbol{u}\|_1}{\epsilon}\right)\right)$ *for some small constant* $c \geq 1$*, then we can solve any* LPA *instance* $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, K, R, \epsilon)$ *in time* $O\left(\mathrm{nnz}^c(\boldsymbol{A}) \operatorname{poly}\log\left(\frac{\mathrm{nnz}(\boldsymbol{A}) R X(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, K)}{\epsilon}\right)\right)$.

3.5 Algebra Space

3.5.1 Overview

In this section, we describe the general process of reduction and solution mapping in the algebra space. Consider the task of reducing an instance A of the form $\{\boldsymbol{A}^A \boldsymbol{x}^A = \boldsymbol{b}^A, \boldsymbol{x}^A \geq \boldsymbol{0}\}$ to another instance B, defined as $\{\boldsymbol{A}^B \boldsymbol{x}^B = \boldsymbol{b}^B, \boldsymbol{x}^B \geq \boldsymbol{0}\}$. We analyze the reductions for both the exact and approximate cases.

Exact Case. The reduction algorithms are designed such that A has a feasible solution if and only if the reduced instance B also has a feasible solution. It implies that instance A is infeasible if and only if B is infeasible.

Reduction:

Given $\boldsymbol{A}^A$ and $\boldsymbol{b}^A$, while the reduction algorithm to construct $\boldsymbol{A}^B$ and $\boldsymbol{b}^B$ is problem-specific, the corresponding $\boldsymbol{x}^B$ is constructed by introducing auxiliary variables $\boldsymbol{x}^{\text{aux}}$ such that:

$$\boldsymbol{x}^B = \begin{bmatrix} \boldsymbol{x}^A \\ \boldsymbol{x}^{\text{aux}} \end{bmatrix}.$$

We keep track of the increase in the problem size and problem magnitude for the reduction algorithm.

Solution mapping:

The solution mapping algorithm is the same for all reduction steps, with a straightforward runtime analysis: If a solver returns a feasible solution $\boldsymbol{x}^B$ for instance B, then we extract the appropriate subset of entries $\boldsymbol{x}^A$ from $\boldsymbol{x}^B$ in time $O(\dim(\boldsymbol{x}^A))$ and return $\boldsymbol{x}^A$ as the solution to the instance A; if the solver returns "infeasible" for instance B, then we return "infeasible" for instance A as well.

Approximate Case. For approximate problems, both the reduction and solution mapping algorithms are applied in the same way. However, there is a key difference from the exact case: the approximate problem does not require to provide a certificate of "infeasibility", but never incorrectly asserts "infeasibility".

In addition to tracking the increase in problem size and magnitude, we perform an error analysis to monitor error propagation. Specifically, let $(\boldsymbol{x}^B, \epsilon^B)$ be an approximate solution to instance B with error ϵ^B, and let $(\boldsymbol{x}^A, \epsilon^A)$ be an approximate solution to instance A with error ϵ^A, obtained by applying the solution mapping algorithm. Our goal is to analyze how ϵ^A increases compared to ϵ^B. We remark that when $\epsilon^A, \epsilon^B = 0$, the statement for the approximate case is reduced to the exact case.

In the remainder of this section, we describe each reduction step in algebra space in detail. For each step, we first describe the reduction algorithm that is used to construct $\boldsymbol{A}^B, \boldsymbol{b}^B$ from $\boldsymbol{A}^A, \boldsymbol{b}^A$. We then present a lemma for the exact case, proving the correctness of the reduction algorithm while tracking the problem size and magnitude. And finally we extend the lemma for the approximate case, analyzing how errors propagate through the solution mapping algorithm.

3.5.2 LP(A) to LEN(A): Reducing Inequalities to Equalities

Given an LP instance $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, K, R)$ where $R \geq \max\{1, \max\{\|\boldsymbol{x}\|_1 : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}\}$, we want to compute a vector $\boldsymbol{x} \geq \boldsymbol{0}$ s.t.

$$\boldsymbol{c}^\top \boldsymbol{x} \geq K, \qquad \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}.$$

To reduce it to an LEN instance, we introduce $\boldsymbol{x}^{\text{aux}}$ as slack variables $\begin{bmatrix} \boldsymbol{s} \\ \alpha \end{bmatrix} \geq \boldsymbol{0}$, and construct

$$\tilde{\boldsymbol{A}} = \begin{bmatrix} \boldsymbol{c}^\top & \boldsymbol{0} & -1 \\ \boldsymbol{A} & \boldsymbol{I} & 0 \end{bmatrix}, \qquad \tilde{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{s} \\ \alpha \end{bmatrix}, \qquad \tilde{\boldsymbol{b}} = \begin{bmatrix} K \\ \boldsymbol{b} \end{bmatrix}. \tag{3.1}$$

This defines an LEN instance $(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{b}}, \tilde{R})$ where $\tilde{R} = \max\{1, \max\{\|\tilde{\boldsymbol{x}}\|_1 : \tilde{\boldsymbol{A}}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}, \tilde{\boldsymbol{x}} \geq \boldsymbol{0}\}\}$.

Lemma 3.5.1 (LP to LEN)**.** *Given an* LP *instance* $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, K, R)$ *with* N *non-zeros and magnitude* X*, we can construct, in* $O(N)$ *time, an* LEN *instance* $(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{b}}, \tilde{R})$ *with* $\tilde{N}$ *non-zeros and magnitude* $\tilde{X}$ *such that*

$$\tilde{N} = O(N), \quad \tilde{R} = O(NRX), \quad \tilde{X} = O(X).$$

If the LP *instance has a solution, then the* LEN *instance has a solution.*

Proof. Based on the reduction construction shown in Eq. (3.1), it is obvious for the linear reduction time and $\tilde{N} = O(N)$, $\tilde{X} = O(X)$. Regarding the radius of polytope $\tilde{R} = \max\{1, \max\{\|\tilde{\boldsymbol{x}}\|_1 : \tilde{\boldsymbol{A}}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}, \tilde{\boldsymbol{x}} \geq \boldsymbol{0}\}\}$, we have

$$\begin{aligned}\|\tilde{\boldsymbol{x}}\|_1 &\leq \|\boldsymbol{x}\|_1 + \|\boldsymbol{x}^{\text{aux}}\|_1 && \text{by triangle inequality}\\ &\leq \|\boldsymbol{x}\|_1 + \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_1 + \left\|\boldsymbol{c}^\top\boldsymbol{x} - K\right\|_1\\ &\leq R + N(\|\boldsymbol{A}\|_{\max}\|\boldsymbol{x}\|_1 + \|\boldsymbol{b}\|_{\max} + \|\boldsymbol{c}\|_{\max}\|\boldsymbol{x}\|_1 + |K|)\\ &\leq O(NRX).\end{aligned}$$

If $\boldsymbol{x}$ is a solution to the LP instance, we can derive a solution $\tilde{\boldsymbol{x}} = (\boldsymbol{x}^\top, \boldsymbol{s}^\top, \alpha)^\top$ to the LEN instance by setting

$$\boldsymbol{s} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}, \qquad \alpha = \boldsymbol{c}^\top\boldsymbol{x} - K.$$

□

Now we extend Lemma 3.5.1 for the exact case to the approximate case.

Lemma 3.5.2 (LPA to LENA)**.** *Given an* LPA *instance* $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, K, R, \epsilon^{lp})$ *with* N *non-zeros and magnitude* X*, we can reduce it to an* LENA *instance* $(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{b}}, \tilde{R}, \epsilon^{le})$ *with* $\tilde{N}$ *non-zeros and magnitude* $\tilde{X}$*, by letting*

$$\epsilon^{le} = \epsilon^{lp},$$

and using Lemma 3.5.1 to construct an LEN *instance* $(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{b}}, \tilde{R})$ *from the* LP *instance* $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, K, R)$*. If* $\tilde{\boldsymbol{x}}$ *is a solution to the* LENA *(or* LEN*) instance, then in time* $O(N)$*, we can compute a solution* $\boldsymbol{x}$ *to the* LPA *(or* LP *respectively) instance, where the exact case holds when* $\epsilon^{le} = \epsilon^{lp} = 0$.

Proof. If $\tilde{\boldsymbol{x}} = \left[\boldsymbol{x}^\top, \boldsymbol{s}^\top, \alpha\right]^\top$ is an approximate solution to the LENA instance with error ϵ^{le}, according to error notions in algebra space (Definition 3.3.4),

$$\begin{aligned}\left|\boldsymbol{c}^\top\boldsymbol{x} - \alpha - K\right| &\leq \epsilon^{le},\\ |\boldsymbol{A}\boldsymbol{x} + \boldsymbol{s} - \boldsymbol{b}| &\leq \epsilon^{le}\boldsymbol{1}.\end{aligned}$$

Taking one direction of the absolute value inequalities,

$$\begin{aligned}\boldsymbol{c}^\top\boldsymbol{x} &\geq \alpha + K - \epsilon^{le} \geq K - \epsilon^{le},\\ \boldsymbol{A}\boldsymbol{x} &\leq -\boldsymbol{s} + \boldsymbol{b} + \epsilon^{le}\boldsymbol{1} \leq \boldsymbol{b} + \epsilon^{le}\boldsymbol{1},\end{aligned}$$

As we set in the reduction that $\epsilon^{le} = \epsilon^{lp}$, $\boldsymbol{x}$ is a solution to the LPA instance $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, K, R, \epsilon^{lp})$. □

3.5.3 LEN(A) to 2-LEN(A): Reducing Integer Coefficients to $\{0, \pm 1, \pm 2\}$

In this section, we show the reduction from an LEN instance $(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{b}}, \tilde{R})$ to a 2-LEN instance $(\bar{\boldsymbol{A}}, \bar{\boldsymbol{b}}, \bar{R})$, where the coefficients of $\bar{\boldsymbol{A}}$ are in $\{0, \pm 1, \pm 2\}$. The reduction is based on *bitwise decomposition.* We illustrate the reduction algorithm with a concrete example.

$$5\boldsymbol{x}_1 + 3\boldsymbol{x}_2 - 7\boldsymbol{x}_3 = -1$$

$$\Downarrow$$

$$(2^0 + 2^2)\boldsymbol{x}_1 + (2^0 + 2^1)\boldsymbol{x}_2 - (2^0 + 2^1 + 2^2)\boldsymbol{x}_3 = -2^0$$

$$\Downarrow$$

$$(\boldsymbol{x}_1 + \boldsymbol{x}_2 - \boldsymbol{x}_3)2^0 + (\boldsymbol{x}_2 - \boldsymbol{x}_3)2^1 + (\boldsymbol{x}_1 - \boldsymbol{x}_3)2^2 = -1 \cdot 2^0$$

It can be decomposed to 3 linear equations, together with carry terms $(\boldsymbol{c}_i - \boldsymbol{d}_i)$, where $\boldsymbol{c}_i, \boldsymbol{d}_i \geq 0$:

$$\begin{aligned} \boldsymbol{x}_1 + \boldsymbol{x}_2 - \boldsymbol{x}_3 - 2(\boldsymbol{c}_0 - \boldsymbol{d}_0) &= -1, \\ \boldsymbol{x}_2 - \boldsymbol{x}_3 + (\boldsymbol{c}_0 - \boldsymbol{d}_0) - 2(\boldsymbol{c}_1 - \boldsymbol{d}_1) &= 0, \\ \boldsymbol{x}_1 - \boldsymbol{x}_3 + (\boldsymbol{c}_1 - \boldsymbol{d}_1) &= 0. \end{aligned} \tag{3.2}$$

Moreover, since $\boldsymbol{c}_i, \boldsymbol{d}_i$ can be arbitrarily large, we also impose an upper bound for carry variables $\boldsymbol{c}_i, \boldsymbol{d}_i \leq \bar{X}$, where $\bar{X}$ will be specified shortly.

Finally, we introduce slack variables $\boldsymbol{s} \geq \boldsymbol{0}$ to reduce inequalities to equalities:

$$\begin{bmatrix} \boldsymbol{c} \\ \boldsymbol{d} \end{bmatrix} + \boldsymbol{s} = \bar{X}\boldsymbol{1}.$$

We repeat the above procedure for all equations in LEN. Notice that in this step, $\boldsymbol{x}^{\text{aux}}$ is composed of $\boldsymbol{c}, \boldsymbol{d}, \boldsymbol{s}$.

Lemma 3.5.3 (LEN to 2-LEN). *Given an* LEN *instance* $(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{b}}, \tilde{R})$ *with* $\tilde{N}$ *non-zeros and magnitude* $\tilde{X}$, *we can construct, in* $O\left(\tilde{N}\log\tilde{X}\right)$ *time, a* 2-LEN *instance* $(\bar{\boldsymbol{A}}, \bar{\boldsymbol{b}}, \bar{R})$ *with* $\bar{N}$ *non-zeros and magnitude* $\bar{X}$ *such that*

$$\bar{N} = O(\tilde{N}\log\tilde{X}), \quad \bar{R} = O(\tilde{N}\tilde{R}\log\tilde{X}), \quad \bar{X} = O(\tilde{R}).$$

If the LEN *instance has a solution, then the* 2-LEN *instance has a solution.*

Proof. By construction, for each equation in LEN , it is decomposed into at most $O(\log\tilde{X})$ equations so that the number of non-zero entries is increased by a factor $O(\log\tilde{X})$. Therefore, the reduction can be conducted in $O(\tilde{N}\log\tilde{X})$ time and $\bar{N} = O(\tilde{N}\log\tilde{X})$.

The key to bounding the radius of polytope $\bar{R}$ and magnitude $\bar{X}$ is to analyze the magnitude of carry terms. For an arbitrary equation $\tilde{\boldsymbol{A}}(q)\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}(q)$, denote the introduced carry terms of the decomposed equations as $\boldsymbol{c}_q, \boldsymbol{d}_q \in \mathbb{R}^{N_q}_{\geq 0}$, where $N_q \leq O(\log\tilde{X})$ denotes the number of decomposed equations corresponding to original equation. Starting from the lowest bit, we have

$$2\left|\boldsymbol{c}_q(0) - \boldsymbol{d}_q(0)\right| \leq 1 + \left\|\tilde{\boldsymbol{x}}\right\|_1 = O(\tilde{R}).$$

Then for the second lowest bit,

$$2\left|\boldsymbol{c}_q(1) - \boldsymbol{d}_q(1)\right| \leq 1 + \left\|\tilde{\boldsymbol{x}}\right\|_1 + \left|\boldsymbol{c}_q(0) - \boldsymbol{d}_q(0)\right| = \left(1 + \frac{1}{2}\right)O(\tilde{R}).$$

Repeating the same until the second-highest bit,

$$2\left|\boldsymbol{c}_q(N_q - 1) - \boldsymbol{d}_q(N_q - 1)\right| \leq \left(1 + \frac{1}{2} + \cdots + \frac{1}{2^{N_q - 1}}\right)O(\tilde{R}) = O(\tilde{R}).$$

Therefore, it suffices to bound $\boldsymbol{c}, \boldsymbol{d}$ by $O(\tilde{R})$, thus the same for the slack variables $\boldsymbol{s} \leq O(\tilde{R})$.

Now, we can bound the magnitude of 2-LENA by $\bar{X} = O(\tilde{R})$ as $\left\|\bar{\boldsymbol{A}}\right\|_{\max} = 2$ and $\left\|\bar{\boldsymbol{b}}\right\|_{\max} \leq O(\tilde{R})$. To bound the radius of the polytope, we have

$$\left\|\bar{\boldsymbol{x}}\right\|_1 \leq \left\|\tilde{\boldsymbol{x}}\right\|_1 + \left\|\boldsymbol{c}\right\|_1 + \left\|\boldsymbol{d}\right\|_1 + \left\|\boldsymbol{s}\right\|_1 \leq \tilde{R} + O(\tilde{N}\log\tilde{X}) \times \left\|\boldsymbol{c}\right\|_{\max} \leq O(\tilde{N}\tilde{R}\log\tilde{X}).$$

Finally, we describe how to construct a solution to 2-LEN given a solution to LEN. If $\tilde{\boldsymbol{x}}$ is a feasible solution to the LEN instance such that $\tilde{\boldsymbol{A}}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}$, we are able to derive a solution $\bar{\boldsymbol{x}} = (\tilde{\boldsymbol{x}}^\top, \boldsymbol{c}^\top, \boldsymbol{d}^\top, \boldsymbol{s}^\top)^\top$ to the 2-LEN instance. More concretely, we plug in $\tilde{\boldsymbol{x}}$ into 2-LEN, and for each group of equations, we start from the lowest bit and compute the smallest nonnegative $\boldsymbol{c}_0, \boldsymbol{d}_0$. Using forward substitution, we can compute all $\boldsymbol{c}, \boldsymbol{d}$. In the end, given $\boldsymbol{c}$ and $\boldsymbol{d}$, $\boldsymbol{s}$ can be computed as well. □

Lemma 3.5.4 (LENA to 2-LENA)**.** *Given an* LENA *instance* $(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{b}}, \tilde{R}, \epsilon^{le})$ *with* $\tilde{N}$ *non-zeros and magnitude* $\tilde{X}$*, we can reduce it to an* 2-LENA *instance* $(\bar{\boldsymbol{A}}, \bar{\boldsymbol{b}}, \bar{R}, \epsilon^{2le})$ *with* $\bar{N}$ *non-zeros and magnitude* $\bar{X}$ *by letting*

$$\epsilon^{2le} = \Omega\left(\frac{1}{\tilde{X}}\right)\epsilon^{le},$$

and using Lemma 3.5.3 to construct a 2-LEN *instance* $(\bar{\boldsymbol{A}}, \bar{\boldsymbol{b}}, \bar{R})$ *from the* LEN *instance* $(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{b}}, \tilde{R})$*. If* $\bar{\boldsymbol{x}}$ *is a solution to the* 2-LENA *(or* 2-LEN*) instance, then in time* $O(\tilde{n})$*, we can compute a solution* $\tilde{\boldsymbol{x}}$ *to the* LENA *(or* LEN *respectively) instance, where the exact case holds when* $\epsilon^{2le} = \epsilon^{le} = 0$.

Proof. Suppose $\bar{\boldsymbol{x}} = \left[\tilde{\boldsymbol{x}}^\top, \boldsymbol{c}^\top, \boldsymbol{d}^\top, \boldsymbol{s}^\top\right]^\top$ is an approximate solution to the 2-LENA instance. We consider an arbitrary equation $\tilde{\boldsymbol{A}}(q)\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}(q)$ in LEN. By Definition 3.3.4, for the corresponding equations in 2-LENA, for any ith bit,

$$\left|\bar{\boldsymbol{A}}_q(i)\bar{\boldsymbol{x}} - \bar{\boldsymbol{b}}_q(i)\right| \leq \epsilon^{2le}.$$

The original equation $\tilde{\boldsymbol{A}}(q)\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}(q)$ in LENA can be recovered by combining all the corresponding bit equations in 2-LENA as follows:

$$\tilde{\boldsymbol{A}}(q)\tilde{\boldsymbol{x}} = \sum_{i=0}^{N_q} 2^i \times \bar{\boldsymbol{A}}_q(i)^\top\bar{\boldsymbol{x}}, \qquad \tilde{\boldsymbol{b}}(q) = \sum_{i=0}^{N_q} 2^i \times \bar{\boldsymbol{b}}_q(i).$$

Hence,

$$\left|\tilde{\boldsymbol{A}}(q)\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{b}}(q)\right| = \sum_{i=0}^{N_q} 2^i \times \left|\bar{\boldsymbol{A}}_q(i)\bar{\boldsymbol{x}} - \bar{\boldsymbol{b}}_q(i)\right| \leq O(\tilde{X}) \cdot \epsilon^{2le}.$$

□

3.5.4 2-LEN(A) to 1-LEN(A): Reducing Coefficients from ±2 to ±1

The reduction from a 2-LEN instance $(\bar{\boldsymbol{A}}, \bar{\boldsymbol{b}}, \bar{R})$ to a 1-LEN instance $(\hat{\boldsymbol{A}}, \hat{\boldsymbol{b}}, \hat{R})$ is relatively straightforward: For each variable $\bar{\boldsymbol{x}}(j)$ that has a ± 2 coefficient, we introduce a new variable $\bar{\boldsymbol{x}}'(j)$, replace every $\pm 2\bar{\boldsymbol{x}}(j)$ with $\pm(\bar{\boldsymbol{x}}(j) + \bar{\boldsymbol{x}}'(j))$, and add an additional equation $\bar{\boldsymbol{x}}(j) - \bar{\boldsymbol{x}}'(j) = 0$.

Lemma 3.5.5 (2-LEN to 1-LEN). *Given a* 2-LEN *instance* $(\bar{\boldsymbol{A}}, \bar{\boldsymbol{b}}, \bar{R})$ *with* $\bar{N}$ *non-zeros and magnitude* $\bar{X}$*, we can construct, in* $O(\bar{N})$ *time, a* 1-LEN *instance* $(\hat{\boldsymbol{A}}, \hat{\boldsymbol{b}}, \hat{R})$ *with* $\hat{N}$ *non-zeros and magnitude* $\hat{X}$ *such that*

$$\hat{N} = O(\bar{N}), \quad \hat{R} = O(\bar{R}), \quad \hat{X} = O(\bar{X}).$$

If the 2-LEN *instance has a solution, then the* 1-LEN *instance has a solution.*

Proof. By construction, we only make changes to each ± 2 coefficient, which will generate four non-zero entries, hence $\hat{N} = O(\bar{N})$ and both $\hat{R}$ and $\hat{X}$ will remain the same.

If $\bar{\boldsymbol{x}}$ is a solution to the 2-LEN instance such that $\bar{\boldsymbol{A}}\bar{\boldsymbol{x}} = \bar{\boldsymbol{b}}$, to derive a solution $\hat{\boldsymbol{x}} = (\bar{\boldsymbol{x}}^\top, (\bar{\boldsymbol{x}}')^\top)^\top$ to the 1-LEN instance, we split each $\bar{\boldsymbol{x}}(j)$ having a ± 2 coefficient in $\bar{\boldsymbol{A}}$ into $\bar{\boldsymbol{x}}'(j), \bar{\boldsymbol{x}}(j)$ such that $\bar{\boldsymbol{x}}'(j) = \bar{\boldsymbol{x}}(j)$. We can check that $\hat{\boldsymbol{x}}$ is a solution to the 1-LEN instance. □

Lemma 3.5.6 (2-LENA to 1-LENA). *Given a 2-*LENA *instance* $(\bar{\boldsymbol{A}}, \bar{\boldsymbol{b}}, \bar{R})$ *with* $\bar{N}$ *non-zeros and magnitude* $\bar{X}$*, we can reduce it to a* 1-LENA *instance* $(\hat{\boldsymbol{A}}, \hat{\boldsymbol{b}}, \hat{R})$ *with* $\hat{N}$ *non-zeros and magnitude* $\hat{X}$ *by letting*

$$\epsilon^{1le} = \Omega\left(\frac{1}{\bar{N}}\right)\epsilon^{2le},$$

and using Lemma 3.5.5 to construct a 1-LEN *instance* $(\hat{\boldsymbol{A}}, \hat{\boldsymbol{b}}, \hat{R})$ *from the* 2-LEN *instance* $(\bar{\boldsymbol{A}}, \bar{\boldsymbol{b}}, \bar{R})$*. If* $\hat{\boldsymbol{x}}$ *is a solution to the* 1-LENA *(*1-LEN*) instance, then in time* $O(\bar{n})$*, we can compute a solution* $\bar{\boldsymbol{x}}$ *to the* 2-LENA *(*2-LEN*, respectively) instance, where the exact case holds when* $\epsilon^{1le} = \epsilon^{2le} = 0$*.*

Proof. Suppose $\hat{\boldsymbol{x}}$ is an approximate solution to the 1-LENA instance. Then for any pair of $(\bar{\boldsymbol{x}}(j), \bar{\boldsymbol{x}}'(j))$, we have

$$|\bar{\boldsymbol{x}}(j) - \bar{\boldsymbol{x}}'(j)| \le \epsilon^{1le},$$

and for any equations in 1-LENA, we have

$$\left|\hat{\boldsymbol{A}}(q)\hat{\boldsymbol{x}} - \hat{\boldsymbol{b}}(q)\right| \le \epsilon^{1le}.$$

Therefore, for any equation in 2-LENA, assume S_q is the set of indices with coefficients ± 2, then

$$\begin{aligned}
\bar{\boldsymbol{A}}(q)\bar{\boldsymbol{x}} &= \sum_{k \notin S_q} \bar{\boldsymbol{A}}(q,k)\bar{\boldsymbol{x}}(k) + \sum_{k \in S_q} (\pm 2)\bar{\boldsymbol{x}}(k) \\
&= \sum_{k \notin S_q} \bar{\boldsymbol{A}}(q,k)\bar{\boldsymbol{x}}(k) + \sum_{k \in S_q} \pm\left((\bar{\boldsymbol{x}}(k) + \bar{\boldsymbol{x}}'(k)) + (\bar{\boldsymbol{x}}(k) - \bar{\boldsymbol{x}}'(k))\right) \\
&= \hat{\boldsymbol{A}}(q)\hat{\boldsymbol{x}} + \sum_{k \in S_q} \pm\left(\bar{\boldsymbol{x}}(k) - \bar{\boldsymbol{x}}'(k)\right).
\end{aligned}$$

Therefore,

$$\begin{aligned}
\left|\bar{\boldsymbol{A}}(q)\bar{\boldsymbol{x}} - \bar{\boldsymbol{b}}(q)\right| &= \left|\hat{\boldsymbol{A}}(q)\hat{\boldsymbol{x}} - \hat{\boldsymbol{b}}(q) + \sum_{k \in S_q} \pm\left(\bar{\boldsymbol{x}}(k) - \bar{\boldsymbol{x}}'(k)\right)\right| \\
&\le \left|\hat{\boldsymbol{A}}(q)\hat{\boldsymbol{x}} - \hat{\boldsymbol{b}}(q)\right| + \sum_{k \in S_q} \left|(\bar{\boldsymbol{x}}(k) - \bar{\boldsymbol{x}}'(k))\right| \\
&\le (1 + |S_q|)\epsilon^{1le} \le O(\bar{N})\epsilon^{1le}.
\end{aligned}$$

□

3.5.5 1-LEN(A) to FHF(A): Encoding Equations as Flows

This section describes the reduction from a 1-LEN instance $(\hat{\boldsymbol{A}}, \hat{\boldsymbol{b}}, \hat{R})$ to an FHF instance $(G^h, F^h, \boldsymbol{u}^h, \mathcal{H}^h, s, t)$. Recall the form of 1-LEN $\hat{\boldsymbol{A}}\hat{\boldsymbol{x}} = \hat{\boldsymbol{b}}$, where $\hat{\boldsymbol{A}} \in \mathbb{R}^{\hat{m}\times\hat{n}}$ and entries of $\hat{\boldsymbol{A}}$ are in $\{0, \pm 1\}$. For an arbitrary equation i in the 1-LEN instance, $\hat{\boldsymbol{A}}(i)\hat{\boldsymbol{x}} = \hat{\boldsymbol{b}}(i), i \in [\hat{m}]$, let $J_i^+ = \{j|\hat{\boldsymbol{A}}(i,j) = 1\}$ and $J_i^- = \{j|\hat{\boldsymbol{A}}(i,j) = -1\}$ denote the set of indices of variables with coefficients being 1 and -1 in equation i, respectively. Then, each equation can be rewritten as a difference of the sum of variables with coefficient 1 and -1:

$$\sum_{j\in J_i^+} \hat{\boldsymbol{x}}(j) - \sum_{j\in J_i^-} \hat{\boldsymbol{x}}(j) = \hat{\boldsymbol{b}}(i), \qquad i \in [\hat{m}].$$

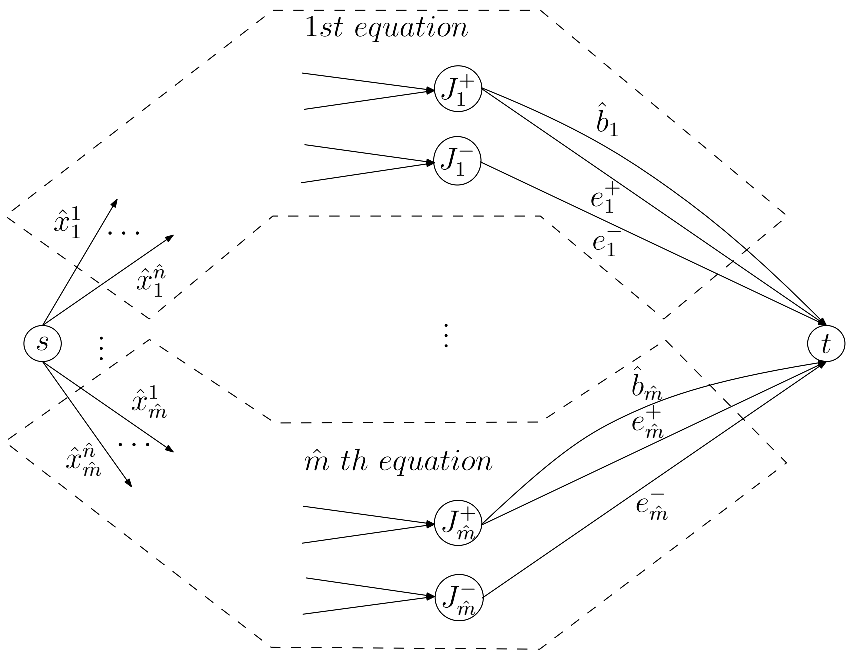

Figure 3.1: The reduction from 1-LEN to FHF.

Representing in this form, we can encode 1-LEN with a graph using the gadget as shown in Figure 3.1 – a graph that is composed of a number of *homologous* edges and *fixed flow* edges. More specifically, the fixed homologous flow network consists of a source s, a sink t, and $\hat{m}$ sections representing the $\hat{m}$ linear equations in 1-LEN. Inside each section i, there are 2 vertices $\{J_i^-, J_i^+\}$ and a number of edges:

- For the incoming edges of $\{J_i^-, J_i^+\}$,
  - if $\hat{\boldsymbol{A}}(i,j) = 1$, then s is connected to J_i^+ by edge $\hat{x}_i^j$ with capacity $\hat{R}$;
  - if $\hat{\boldsymbol{A}}(i,j) = -1$, then s is connected to J_i^- by edge $\hat{x}_i^j$ with capacity $\hat{R}$;
  - if $\hat{\boldsymbol{A}}(i,j) = 0$, no edge is needed.

  Note that those incoming edges that correspond to the same variable are forced to route the same amount of flow, i.e., $(\hat{x}_1^j, \cdots, \hat{x}_{\hat{m}}^j), j \in [\hat{n}]$ constitute a *homologous* edge set that corresponds to the variable $\hat{\boldsymbol{x}}(j)$.

- For the outgoing edges of $\{J_i^-, J_i^+\}$,

- J_i^+ is connected to t by a *fixed flow* edge $\hat{b}_i$ that routes $\hat{\boldsymbol{b}}(i)$ units of flow;
- J_i^+ and J_i^- are connected to t by a pair of *homologous* edges e_i^+, e_i^- with capacity $\hat{R}$.

Lemma 3.5.7 (1-LEN to FHF)**.** *Given a* 1-LEN *instance* $(\hat{\boldsymbol{A}}, \hat{\boldsymbol{b}}, \hat{R})$ *with* $\hat{N}$ *non-zeros and magnitude* $\hat{X}$*, we can construct, in time* $O(\hat{N})$*, an* FHF *instance* $(G^h, F^h, \boldsymbol{u}^h, \mathcal{H}^h = (H_1, \cdots, H_h), s, t)$ *such that*

$$\left|E^h\right| = O(\hat{N}), \quad \left\|\boldsymbol{u}^h\right\|_{\max} = \max\left\{\hat{R}, \hat{X}\right\}.$$

If the 1-LEN *instance has a solution, then the* FHF *instance has a solution.*

Proof. According to Figure 3.1, the constructed graph G^h preserves the sparsity of $\hat{\boldsymbol{A}}$ and the number of edges $\left|E^h\right|$ can be bounded by $O(\hat{N})$. Moreover, the maximum edge capacity is bounded by

$$\left\|\boldsymbol{u}^h\right\|_{\max} = \max\left\{\hat{R}, \left\|\hat{\boldsymbol{b}}\right\|_{\max}\right\} \le \max\left\{\hat{R}, \hat{X}\right\}.$$

If $\hat{\boldsymbol{x}}$ is a solution to the 1-LEN instance, we can derive a feasible flow $\boldsymbol{f}^h$ for the FHF instance by setting

$$\boldsymbol{f}^h(\hat{x}_1^j) = \cdots = \boldsymbol{f}^h(\hat{x}_{\hat{m}}^j) = \hat{\boldsymbol{x}}(j) \le \hat{R}, \quad \forall j \in [\hat{n}],$$

$$\boldsymbol{f}^h(\hat{b}_i) = \hat{\boldsymbol{b}}(i) \quad \text{and} \quad \boldsymbol{f}^h(e_i^+) = \boldsymbol{f}^h(e_i^-) = \sum_{j \in J_i^-} \hat{\boldsymbol{x}}(j) = \sum_{j \in J_i^+} \hat{\boldsymbol{x}}(j) - \hat{\boldsymbol{b}}(j), \quad \forall i \in [\hat{m}].$$

It is a feasible flow since both capacity constraints and flow conservation constraints are satisfied. □

In the approximate case, we first formally define FHFA by applying the corresponding error notions (Definition 3.3.16) to FHF.

Definition 3.5.8 (FHF Approximate Problem (FHFA))**.** An FHFA instance is given by an FHF instance $(G, F, \boldsymbol{u}, \mathcal{H}, s, t)$ as in Definition 3.3.12, and an error parameter $\epsilon \in [0, 1]$, which we collect in a tuple $(G, F, \mathcal{H}, \boldsymbol{u}, s, t, \epsilon)$. We say an algorithm solves the FHFA problem, if, given any FHFA instance, it returns a flow $\boldsymbol{f} \ge \boldsymbol{0}$ that satisfies

$$\begin{aligned}
&\boldsymbol{u}(e) - \epsilon \le \boldsymbol{f}(e) \le \boldsymbol{u}(e) + \epsilon, \ \forall e \in F, && \text{(error in congestion)}\\
&0 \le \boldsymbol{f}(e) \le \boldsymbol{u}(e) + \epsilon, \ \forall e \in E \backslash F, && \text{(error in congestion)}\\
&\left|\sum_{u:(u,v)\in E} \boldsymbol{f}(u,v) - \sum_{w:(v,w)\in E} \boldsymbol{f}(v,w)\right| \le \epsilon, \ \forall v \in V \backslash \{s,t\}, && \text{(error in demand)}\\
&|\boldsymbol{f}(v,w) - \boldsymbol{f}(v',w')| \le \epsilon, \ \forall (v,w),(v',w') \in H_i, \ \forall H_i \in \mathcal{H}, && \text{(error in homology)}
\end{aligned}$$

or it correctly declares that the associated FHF instance is infeasible.

Lemma 3.5.9 (1-LENA to FHFA)**.** *Given a* 1-LENA *instance* $(\hat{\boldsymbol{A}}, \hat{\boldsymbol{b}}, \hat{R}, \epsilon^{1le})$ *with* $\hat{N}$ *non-zeros and magnitude* $\hat{X}$*, we can reduce it to an* FHFA *instance* $(G^h, F^h, \mathcal{H}^h, \boldsymbol{u}^h, s, t, \epsilon^h)$ *by letting*

$$\epsilon^h = \Omega\left(\frac{1}{\hat{N}}\right)\epsilon^{1le},$$

and using Lemma 3.5.7 to construct an FHF *instance* $(G^h, F^h, \mathcal{H}^h, \boldsymbol{u}^h, s, t)$ *from the* 1-LEN *instance* $(\hat{\boldsymbol{A}}, \hat{\boldsymbol{b}}, \hat{R})$*. If* $\boldsymbol{f}^h$ *is a solution to the* FHFA *(*FHF*) instance, then in time* $O(\hat{n})$*, we can compute a solution* $\hat{\boldsymbol{x}}$ *to the* 1-LENA *(*1-LEN*, respectively) instance, where the exact case holds when* $\epsilon^h = \epsilon^{1le} = 0$*.*

Proof. Let $\boldsymbol{f}^h$ be a feasible solution to the FHFA instance, so that error in demand, error in congestion, and error in homology can all be bounded by ϵ^h. Let $\hat{\boldsymbol{x}}_i(j) = \boldsymbol{f}^h(\hat{x}_i^j)$ denote the amount of flow routed through in the ith section of G^h. For an arbitrary equation $\hat{\boldsymbol{A}}(i)\hat{\boldsymbol{x}} - \hat{\boldsymbol{b}}(i)$ in 1-LENA , its error can be bounded as follows:

$$\begin{aligned} |\hat{\boldsymbol{A}}(i)\hat{\boldsymbol{x}} - \hat{\boldsymbol{b}}(i)| &= |\hat{\boldsymbol{A}}(i)\hat{\boldsymbol{x}} - \hat{\boldsymbol{A}}(i)\hat{\boldsymbol{x}}_i + \hat{\boldsymbol{A}}(i)\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{b}}(i)| \\ &\le |\hat{\boldsymbol{A}}(i)\hat{\boldsymbol{x}} - \hat{\boldsymbol{A}}(i)\hat{\boldsymbol{x}}_i| + \left|\hat{\boldsymbol{A}}(i)\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{b}}(i)\right|. \end{aligned}$$

To bound the first term, we have

$$|\hat{\boldsymbol{A}}(i)\hat{\boldsymbol{x}} - \hat{\boldsymbol{A}}(i)\hat{\boldsymbol{x}}_i| \le \left\|\hat{\boldsymbol{A}}(i)\right\|_1 \|\hat{\boldsymbol{x}} - \hat{\boldsymbol{x}}_i\|_{\max} \le \left\|\hat{\boldsymbol{A}}(i)\right\|_1 \epsilon^h,$$

where the last step follows from the error in homology in G^h. To bound the second term, we have

$$\begin{aligned} & \left|\hat{\boldsymbol{A}}(i)\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{b}}(i)\right| \\ = & \left|\boldsymbol{f}^h(s, J_i^+) - \boldsymbol{f}^h(s, J_i^-) - \hat{\boldsymbol{b}}(i)\right| && \text{by construction} \\ \le & \left|\boldsymbol{f}^h(s, J_i^+) - (\boldsymbol{f}^h(e_i^+) + \boldsymbol{f}^h(\hat{b}_i))\right| + \left|\boldsymbol{f}^h(e_i^+) + \boldsymbol{f}^h(\hat{b}_i) - \boldsymbol{f}^h(s, J_i^-) - \hat{\boldsymbol{b}}(i)\right| \\ \le & \epsilon^h + \left|\boldsymbol{f}^h(e_i^+) + \boldsymbol{f}^h(\hat{b}_i) - \boldsymbol{f}^h(s, J_i^-) - \hat{\boldsymbol{b}}(i)\right| && \text{by error in demand of vertex } J_i^+ \\ \le & \epsilon^h + \left|\boldsymbol{f}^h(e_i^+) - \boldsymbol{f}^h(e_i^-)\right| + \left|\boldsymbol{f}^h(e_i^-) - \boldsymbol{f}^h(s, J_i^-)\right| + \left|\boldsymbol{f}^h(\hat{b}_i) - \hat{\boldsymbol{b}}(i)\right| \\ \le & 4\epsilon^h, \end{aligned}$$

where the last step follows from error in homology between edge e_i^+, e_i^-, error in demand of vertex J_i^-, and error in congestion of fixed flow edge $\hat{b}_i$, respectively.

Combining the bounds of the two terms, we obtain the error of an arbitrary equation in 1-LENA as

$$|\hat{\boldsymbol{A}}(i)\hat{\boldsymbol{x}} - \hat{\boldsymbol{b}}(i)| \le \left(\left\|\hat{\boldsymbol{A}}(i)\right\|_1 + 4\right) \epsilon^h \le O(\hat{N})\epsilon^h.$$

□

3.6 Flow Space

3.6.1 Overview

In the flow space, the starting problem is FHF (Definition 3.3.12), whose additional edge constraints include *homologous flow constraints* (Definition 3.3.9) and *fixed flow constraints* (Definition 3.3.10). The target problem is 2CF (Definition 3.3.7), which has no additional edge constraints. Therefore, we are tasked to drop additional edge constraints step by step in the flow space. Note that new additional edge constraints may emerge as we drop ones, for example, *selective flow constraints* (Definition 3.3.11).

Exact Case. To reduce instance A to instance B, we focus on dropping specific edge constraints. Let the graph of instance A be $G^A = (E^A, V^A)$ and let $S^A \subseteq E^A$ denote the set of edges in A with the constraints to be dropped. The reduced graph $G^B = (E^B, V^B)$ replaces each edge $e \in S^A$ with a constant-sized gadget.[4] These gadgets simplify or remove the associated constraints.

The general reduction and solution mapping framework is shown in Figure 3.2.

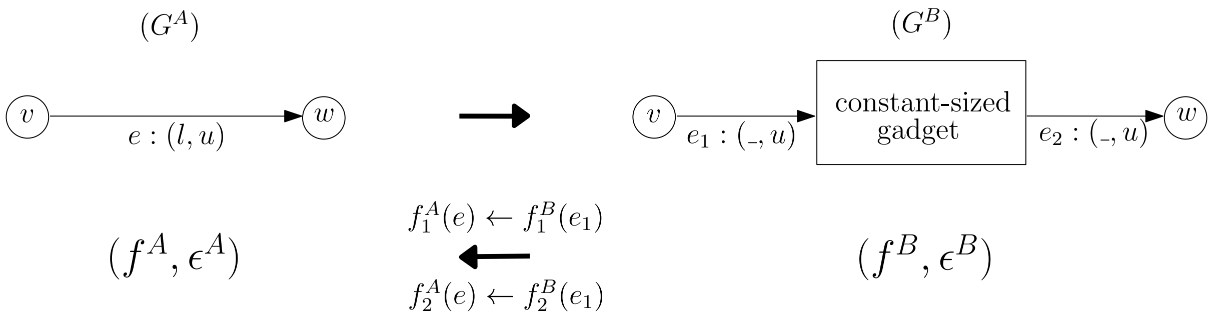

Figure 3.2: Reduction and solution mapping algorithms in the flow space.

Reduction:

For each special edge $e = (v, w) \in S^A$ with lower and upper capacities (l, u),[5] the reduction replaces e with two edges e_1 and e_2, along with a constant sized gadget between e_1 and e_2. The upper capacity of e_1 and e_2 remains u; and the lower capacity is either l or reduced to 0, depending on the reduction step.

Solution mapping:

If a solver returns $\boldsymbol{f}^B$ for instance B, the flow for instance A is constructed as follows:[6]

$$\boldsymbol{f}_i^A(e) = \boldsymbol{f}_i^B(e_1), \quad \text{for } e \in S^A, \quad i \in \{1, 2\}.$$

For regular edges in $e \in E^A \backslash S^A$, the flow is directly mapped:

$$\boldsymbol{f}_i^A(e) = \boldsymbol{f}_i^B(e), \quad \text{for } e \in E^A \backslash S^A, \quad i \in \{1, 2\}.$$

If the solver returns "infeasible" for instance B, then we return "infeasible" for instance A as well.

Based on the general reduction and solution mapping framework, the following results apply to every step within the flow space.

Lemma 3.6.1 (Problem Size and Runtime for Reduction Algorithm)**.** *In the flow space, when reducing instance A to instance B, the number of edges in B satisfies $\left|E^B\right| = O(\left|E^A\right|)$. Moreover, the reduction algorithm takes runtime $\left|E^B\right| = O(\left|E^A\right|)$.*

Proof. There are at most $|E^A|$ edges in G^A that are replaced by gadgets of constant size. The total size of G^B is thus $O(\left|E^A\right|)$. The runtime is linear to the number of edges in G^B, thus $O(\left|E^B\right|)$. □

[4] Only a few steps may utilize gadgets that deviate slightly from the general pattern. If this occurs, we will provide a note in the relevant section.

[5] The lower edge capacity is set to $l = 0$ unless the edge is a fixed-flow edge, in which case $l = u$.

[6] For 1-commodity flow problem, let $\boldsymbol{f}_2 = \boldsymbol{0}$.

Lemma 3.6.2 (Runtime for Solution Mapping Algorithm)**.** *In the flow space, mapping a solution to instance B to a solution to instance A takes $O(|E^A|)$ runtime.*

Proof. The runtime is linear to the number of edges in G^A, as solution mapping involves directly transferring or reconstructing flow values for each edge. Thus, the runtime is $O(|E^A|)$. □

Approximate Case. We apply the same reduction and solution mapping algorithm for approximate problems. Hence, Lemma 3.6.1, 3.6.2 also hold for the approximate case. Additional analyses focus on the error propagation during solution mapping. Specifically, if $(\boldsymbol{f}^B, \epsilon^B)$ is an approximate solution to instance B with error ϵ^B, then the mapped solution $(\boldsymbol{f}^A, \epsilon^A)$ to instance A has an error ϵ^A that can be upper bounded. In particular, when $\epsilon^A, \epsilon^B = 0$, the statement for the approximate case is reduced to the exact case.

Different types of errors are analyzed separately for problems in the flow space. As the capacity constraints and the flow conservation constraints are required for all problems in the flow space, the corresponding error in congestion and error in demand are involved in the error analysis for all steps. The following error bounds are applicable at every step within the flow space.

Lemma 3.6.3 (Error in Congestion)**.** *Let $\epsilon_l^A, \epsilon_u^A$ and $\epsilon_l^B, \epsilon_u^B$ be the errors in congestion w.r.t. lower and upper capacity for $\boldsymbol{f}^A$ and $\boldsymbol{f}^B$, respectively. If applying the solution mapping algorithm, then for edges in G^A with $l = 0$,*

$$\epsilon_l^A = 0,$$

and for all edges in G^A,

$$\epsilon_u^A \leq O(\epsilon_u^B).$$

Proof. With the non-negativity constraints satisfied by $\boldsymbol{f}^A$ for edges in G^A with $l = 0$, ϵ_l^A can be trivially bounded, regardless of ϵ_l^B.

For the upper edge capacity, the error bounds are straightforward if e is a regular edge, as the flow is copied directly from G^A and G^B. However, if e is a special edge, the upper-capacity error in G^A remains bounded by ϵ^B, as e_1 and e have the same upper edge capacity u according to Figure 3.2. □

Remark that for edges in G^A with $l = u$, lower-capacity error in congestion requires case-by-case analysis.

The following helper lemma is frequently used to analyze error in demand across various reduction steps.

Lemma 3.6.4 (Error in Demand)**.** *Let $\epsilon_{d,i}^A$ and $\epsilon_{d,i}^B$ be the error in demand of $\boldsymbol{f}^A$ and $\boldsymbol{f}^B$ for commodity i, $i \in \{1, 2\}$, respectively. If applying the solution mapping algorithm, we have*

$$\epsilon_{d,i}^A \leq \epsilon_{d,i}^B + \max_{v \in V^A \setminus \{s_i, t_i\}} \sum_{e=(u,v) \in S^A} \left|\boldsymbol{f}_i^B(e_1) - \boldsymbol{f}_i^B(e_2)\right|.$$

Proof. By Definition 3.3.16, error in demand for vertices other than the source/sink nodes is defined as

$$\left| \sum_{u:(u,v) \in E} \boldsymbol{f}_i(u, v) - \sum_{w:(v,w) \in E} \boldsymbol{f}_i(v, w) \right| \leq \epsilon, \quad \forall v \in V \setminus \{s_i, t_i\},\ i \in \{1, 2\}.$$

As illustrated in Figure 3.3, consider an arbitrary vertex $v \in V^B \backslash \{s_i, t_i\}$ with a number of incoming and outgoing edges. Some of these edges are derived from the reduction of special edges in G^A, as indicated by dashed boxes in the figure.

When mapping $\boldsymbol{f}^B$ to $\boldsymbol{f}^A$, the incoming/outgoing flows of v along regular edges remain unchanged, while flows along special edges may vary due to the gap between e_1 and e_2.

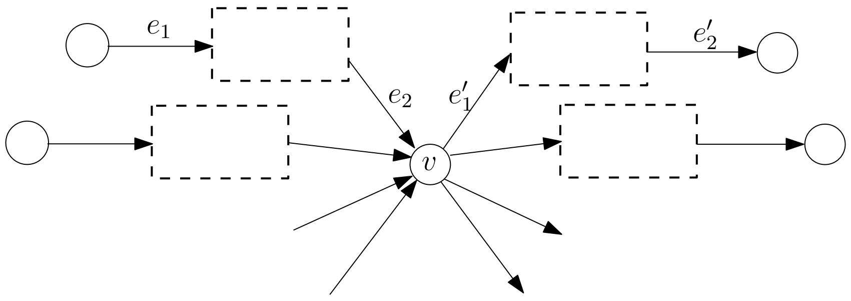

Figure 3.3: An illustration of the incoming and outgoing flows of vertex $v \in V^B \backslash \{s_i, t_i\}$ when get reduced to G^B.

For flow of commodity $i \in \{1, 2\}$, we have

$$\begin{aligned}
\epsilon^A_{d,i} &= \max_{v \in V^A \backslash \{s_i,t_i\}} \left| \sum_{(u,v) \in E^A} \boldsymbol{f}^A_i(u,v) - \sum_{(v,w) \in E^A} \boldsymbol{f}^A_i(v,w) \right| \\
&\overset{(1)}{=} \max_{v \in V^A \backslash \{s_i,t_i\}} \left| \left(\sum_{(u,v) \in S^A} \boldsymbol{f}^A_i(u,v) + \sum_{(u,v) \in E^A \backslash S^A} \boldsymbol{f}^A_i(u,v) \right) - \left(\sum_{(v,w) \in S^A} \boldsymbol{f}^A_i(v,w) + \sum_{(v,w) \in E^A \backslash S^A} \boldsymbol{f}^A_i(v,w) \right) \right| \\
&\overset{(2)}{=} \max_{v \in V^A \backslash \{s_i,t_i\}} \left| \left(\sum_{e=(u,v) \in S^A} \boldsymbol{f}^B_i(e_1) + \sum_{(u,v) \in E^A \backslash S^A} \boldsymbol{f}^B_i(u,v) \right) - \left(\sum_{e=(v,w) \in S^A} \boldsymbol{f}^B_i(e_1) + \sum_{(v,w) \in E^A \backslash S^A} \boldsymbol{f}^B_i(v,w) \right) \right| \\
&\overset{(3)}{\le} \epsilon^B_{d,i} + \max_{v \in V^A \backslash \{s_i,t_i\}} \left| \left(\sum_{e=(u,v) \in S^A} \boldsymbol{f}^B_i(e_1) + \sum_{(u,v) \in E^A \backslash S^A} \boldsymbol{f}^B_i(u,v) \right) - \left(\sum_{e=(u,v) \in S^A} \boldsymbol{f}^B_i(e_2) + \sum_{(u,v) \in E^A \backslash S^A} \boldsymbol{f}^B_i(u,v) \right) \right| \\
&= \epsilon^B_{d,i} + \max_{v \in V^A \backslash \{s_i,t_i\}} \left| \sum_{e=(u,v) \in S^A} \boldsymbol{f}^B_i(e_1) - \sum_{e=(u,v) \in S^A} \boldsymbol{f}^B_i(e_2) \right| \\
&= \epsilon^B_{d,i} + \max_{v \in V^A \backslash \{s_i,t_i\}} \left| \sum_{e=(u,v) \in S^A} (\boldsymbol{f}^B_i(e_1) - \boldsymbol{f}^B_i(e_2)) \right| \\
&\le \epsilon^B_{d,i} + \max_{v \in V^A \backslash \{s_i,t_i\}} \sum_{e=(u,v) \in S^A} \left| \boldsymbol{f}^B_i(e_1) - \boldsymbol{f}^B_i(e_2) \right| .
\end{aligned}$$

For step (1), we separate special edges from regular edges that are incident to vertex v. For step (2), we apply the solution mapping algorithm from $\boldsymbol{f}^B$ back to $\boldsymbol{f}^A$. For step (3), we can replace the sum of outgoing flows of v by the sum of its incoming flows with an error in demand of instance B being introduced. □

3.6.2 FHF(A) to FPHF(A)

In this section, we show the reduction from an FHF instance $(G^h, H^h, \mathcal{H}^h, \boldsymbol{u}^h, s, t)$ to an FPHF instance $(G^p, F^p, \mathcal{H}^p, \boldsymbol{u}^p, s, t)$. This step does not drop any additional edge constraints, but simplifies the edge constraints for future dropping steps. More specifically, in G^h, it is possible for more than two edges to be constrained to route the same amount of flow. The goal is to reduce G^h to G^p such that each edge is homologous to at most one other edge.

Suppose that $(v_1, w_1), \cdots, (v_k, w_k) \in E^h$ belong to a set of homologous edges of size k in G^h. As shown in Figure 3.4,[7] we replace $(v_i, w_i), i \in \{2, \cdots, k-1\}$ by two edges (v_i, z_i) and (z_i, w_i) such that z_i is a new vertex incident only to these two edges, and edge capacities of the two new edges are the same as that of the original edge (v_i, w_i). Then, we can construct $k-1$ pairs of homologous edges: (v_1, w_1) and (v_2, z_2); (z_2, w_2) and (v_3, z_3); $\cdots$; (z_i, w_i) and (v_{i+1}, z_{i+1}); $\cdots$; (z_{k-1}, w_{k-1}) and (v_k, w_k). In addition, no reduction is performed on non-homologous edges in G^h, and we trivially copy these edges to G^p. Compared with Figure 3.2, the constant-sized gadget in this step corresponds to the inserted vertex z_i, as indicated by the dashed boxes.

Lemma 3.6.5 (FHF to FPHF). *Given an* FHF *instance* $(G^h, F^h, \mathcal{H}^h, \boldsymbol{u}^h, s, t)$, *we can construct, in time* $O(|E^p|)$, *an* FPHF *instance* $(G^p, F^p, \mathcal{H}^p, \boldsymbol{u}^p, s, t)$ *such that*

$$|E^p| = O\left(\left|E^h\right|\right), \quad \|\boldsymbol{u}^p\|_{\max} = \left\|\boldsymbol{u}^h\right\|_{\max}.$$

If the FHF *instance has a solution, then the* FPHF *instance has a solution.*

Proof. The runtime of reduction and the bound for $|E^p|$ follow from Lemma 3.6.1. Moreover, as the reduction of this step only inserts new vertices without modifying edge capacities, we have $\|\boldsymbol{u}^p\|_{\max} = \left\|\boldsymbol{u}^h\right\|_{\max}$.

If $\boldsymbol{f}^h$ is a feasible flow to the FHF instance, it is easy to derive a solution $\boldsymbol{f}^p$ to the FPHF instance. Concretely, for any homologous edge $e \in E^h$ that is split into two edges $e_1, e_2 \in E^p$, we set $\boldsymbol{f}^p(e_1) = \boldsymbol{f}^p(e_2) = \boldsymbol{f}^h(e)$. Since the vertex between e_1 and e_2 is only incident to these two edges, then the conservation of flows is satisfied on the inserted vertices. Moreover, since the edge capacities of e_1 and e_2 are the same as that of e, and they route the same amount of flows, thus the capacity constraint is also satisfied on the split edges. In addition, the flow conservation and capacity constraints are trivially satisfied for the rest vertices and edges. Therefore, $\boldsymbol{f}^p$ is a feasible flow to the FPHF instance. □

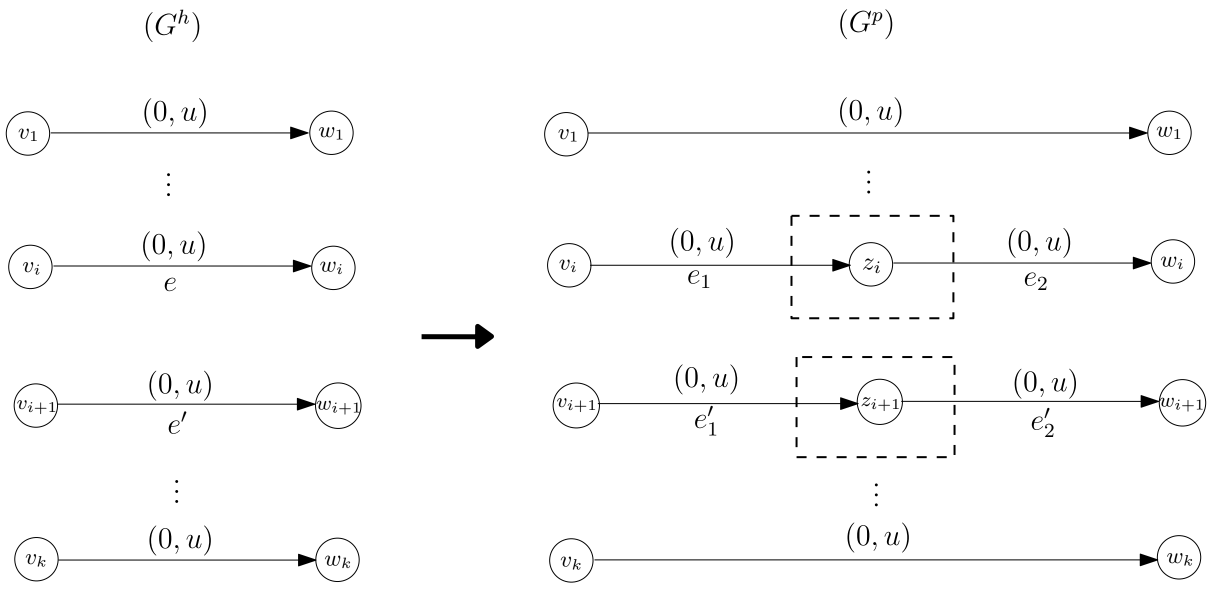

Figure 3.4: The reduction from FHF to FPHF.

[7]We use $(0, u)$ for an non-fixed flow edge of capacity u.

In the approximate case, we first formally define FPHFA by applying the corresponding error notions (Definition 3.3.16) to FPHF.

Definition 3.6.6 (FPHF Approximate Problem (FPHFA))**.** An FPHFA instance is given by an FPHF instance $(G, F, \mathcal{H}, \boldsymbol{u}, s, t)$ as in Definition 3.3.13, and an error parameter $\epsilon \in [0, 1]$, which we collect in a tuple $(G, F, \mathcal{H}, \boldsymbol{u}, s, t, \epsilon)$. We say an algorithm solves the FPHFA problem, if, given any FPHFA instance, it returns a flow $\boldsymbol{f} \geq \boldsymbol{0}$ that satisfies

$$\boldsymbol{u}(e) - \epsilon \leq \boldsymbol{f}(e) \leq \boldsymbol{u}(e) + \epsilon, \ \forall e \in F, \qquad \text{(error in congestion)}$$

$$0 \leq \boldsymbol{f}(e) \leq \boldsymbol{u}(e) + \epsilon, \ \forall e \in E \backslash F, \qquad \text{(error in congestion)}$$

$$\left| \sum_{u:(u,v)\in E} \boldsymbol{f}(u, v) - \sum_{w:(v,w)\in E} \boldsymbol{f}(v, w) \right| \leq \epsilon, \ \forall v \in V \backslash \{s, t\}, \qquad \text{(error in demand)}$$

$$|\boldsymbol{f}(v, w) - \boldsymbol{f}(y, z)| \leq \epsilon, \ \forall (v, w), (y, z) \in H_i, \ \forall H_i \in \mathcal{H}, \qquad \text{(error in pairwise homology)}$$

or it correctly declares that the associated FPHF instance is infeasible.

Lemma 3.6.7 (FHFA to FPHFA)**.** *Given an FHFA instance $(G^h, F^h, \mathcal{H}^h, \boldsymbol{u}^h, s, t, \epsilon^h)$, we can reduce it to an FPHFA instance $(G^p, F^p, \mathcal{H}^p, \boldsymbol{u}^p, s, t, \epsilon^p)$ by letting*

$$\epsilon^p = \Omega\left(\frac{1}{|E^h|}\right)\epsilon^h,$$

and using Lemma 3.6.5 to construct an FPHF instance $(G^p, F^p, \mathcal{H}^p, \boldsymbol{u}^p, s, t)$ from the FHF instance $(G^h, H^h, \mathcal{H}^h, \boldsymbol{u}^h, s, t)$. If $\boldsymbol{f}^p$ is a solution to the FPHFA (FPHF) instance, then in time $O(|E^h|)$, we can compute a solution $\boldsymbol{f}^h$ to the FHFA (FHF, respectively) instance, where the exact case holds when $\epsilon^p = \epsilon^h = 0$.

Proof. To bound error in congestion, we have $\epsilon_u^h \leq O(\epsilon^p)$ according to Lemma 3.6.3. For ϵ_l^h, we have $\epsilon_l^h = 0$ if $e \in E^p \backslash F^p$. If $e \in F^p$, since we just copy them without making reductions for fixed flow edges in this step, we also have $\epsilon_l^h \leq O(\epsilon^p)$.

To bound error in demand, by Lemma 3.6.4 and error in demand of auxiliary vertex z, we have

$$\epsilon_d^h \leq \epsilon^p + \max_{v\in V^h\backslash\{s,t\}} \sum_{e=(u,v)\in H^h} |\boldsymbol{f}^p(e_1) - \boldsymbol{f}^p(e_2)| \leq O\left(|E^h|\right)\epsilon^p.$$

Finally, we bound error in homology in G^h. It is observed that error in homology in G^h for a homologous edge set of size k gets accumulated by $(k-1)$ times of error in pair homology in G^p, and we have $k \leq |E_h|$. Thus, $\epsilon_h^h \leq O\left(|E^h|\right)\epsilon^p$. Putting it all together, we have

$$\epsilon^h = \max\{\epsilon_l^h, \epsilon_u^h, \epsilon_d^h, \epsilon_h^h\} \leq O\left(|E^h|\right)\epsilon^p.$$

□

3.6.3 FPHF(A) to SFF(A): Dropping Homologous Flow Constraints

We show the reduction from an FPHF instance $(G^p, F^p, \mathcal{H}^p, \boldsymbol{u}^p, s, t)$ to an SFF instance $(G^s, F^s, S_1, S_2, \boldsymbol{u}^s, s_1, t_1, s_2, t_2)$. Assume that $\{e, \hat{e}\} \in \mathcal{H}^p$ is an arbitrary pair of homologous edges in G^p. As shown in Figure 3.5, we map $\{e, \hat{e}\}$ in G^p to a constant-sized gadget in G^s, as indicated in the dashed boxes. The key idea to remove the homologous requirement is to introduce a second commodity between a source-sink pair (s_2, t_2). Concretely, we impose the fixed flow constraints on $(e_4, \hat{e}_4)$, the selective flow constraint of commodity 1 on $(e_1, e_2, \hat{e}_1, \hat{e}_2)$, and the selective flow constraint of commodity 2 on $(e_3, e_5/\hat{e}_3, \hat{e}_5)$. Then, there is a flow of commodity 2 that routes through the directed path $e_3 \to e_4 \to e_5/\hat{e}_3 \to \hat{e}_4 \to \hat{e}_5$, and a flow of commodity 1 through paths $e_1 \to e_4 \to e_2$ and $\hat{e}_1 \to \hat{e}_4 \to \hat{e}_2$. The fixed flow constraint on $(e_4, \hat{e}_4)$ forces the flow of commodity 1 through the two paths to be equal, since the flows of commodity 2 on $(e_4, \hat{e}_4)$ are equal. Thus, the homologous requirement for edge e and $\hat{e}$ is simulated. In addition, as no reduction is performed on non-homologous edges in G^p, we trivially copy these edges to G^s, and restrict these edges to be selective for commodity 1.

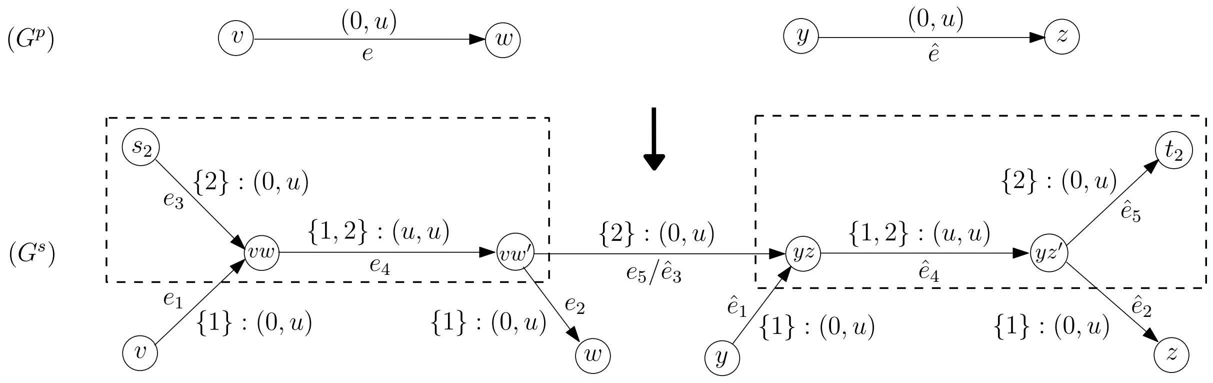

Figure 3.5: The reduction from FPHF to SFF. The constant-sized gadget is indicated by the dashed boxes.

Lemma 3.6.8 (FPHF to SFF). *Given an* FPHF *instance* $(G^p, F^p, \mathcal{H}^p, \boldsymbol{u}^p, s, t)$, *we can construct, in time* $O(|E^p|)$, *an* SFF *instance* $(G^s, F^s, S_1, S_2, \boldsymbol{u}^s, s_1, t_1, s_2, t_2)$ *such that*

$$|E^s| = O(|E^p|), \quad \|\boldsymbol{u}^s\|_{\max} = \|\boldsymbol{u}^p\|_{\max}.$$

If the FPHF *instance has a solution, then the* SFF *instance has a solution.*

Proof. The runtime of reduction and the bound for $|E^s|$ follow from Lemma 3.6.1. Moreover, we have by construction $\|\boldsymbol{u}^s\|_{\max} = \|\boldsymbol{u}^p\|_{\max}$.

If $\boldsymbol{f}^p$ is a solution to the FPHF instance, it is easy to derive a solution $\boldsymbol{f}^s$ to the SFF instance. Concretely, we construct a feasible flow $\boldsymbol{f}^s$ as follows:

- For any pair of homologous edge $\{e, \hat{e}\} \in \mathcal{H}^p$, in its corresponding gadget in G^s, we set

$$\boldsymbol{f}_1^s(e_1) = \boldsymbol{f}_1^s(e_4) = \boldsymbol{f}_1^s(e_2) = \boldsymbol{f}^p(e) = \boldsymbol{f}^p(\hat{e}) = \boldsymbol{f}_1^s(\hat{e}_1) = \boldsymbol{f}_1^s(\hat{e}_4) = \boldsymbol{f}_1^s(\hat{e}_2) \le u,$$

  where $u = \boldsymbol{u}^p(e) = \boldsymbol{u}^p(\hat{e})$, and set

$$\boldsymbol{f}_2^s(e_1) = \boldsymbol{f}_2^s(e_2) = \boldsymbol{f}_2^s(\hat{e}_1) = \boldsymbol{f}_2^s(\hat{e}_2) = 0.$$

  It is obvious that $\boldsymbol{f}^s$ satisfies the selective constraint and the capacity constraint on edges $e_1, e_2, \hat{e}_1, \hat{e}_2$, and satisfies conservation of flows for commodity 1 on vertices vw, vw', yz, yz'.

Moreover, we set

$$\boldsymbol{f}_2^s(e_3) = \boldsymbol{f}_2^s(e_4) = \boldsymbol{f}_2^s(e_5) = \boldsymbol{f}_2^s(\hat{e}_4) = \boldsymbol{f}_2^s(\hat{e}_5) = u - \boldsymbol{f}^p(e) \leq u,$$

$$\boldsymbol{f}_1^s(e_3) = \boldsymbol{f}_1^s(e_4) = \boldsymbol{f}_1^s(e_5) = \boldsymbol{f}_1^s(\hat{e}_4) = \boldsymbol{f}_1^s(\hat{e}_5) = 0.$$

Then it is obvious that $\boldsymbol{f}^s$ satisfies the selective constraint and the capacity constraint on edges $e_3, e_5, \hat{e}_5$, and satisfies the flow conservation constraint of commodity 2 on vertices vw, vw', yz, yz'.

It remains to verify if $\boldsymbol{f}^s$ satisfies the fixed flow constraint on edges $e_4, \hat{e}_4$. According to the above constructions, we have (we abuse the notation to also let $\boldsymbol{f}^s = \boldsymbol{f}_1^s + \boldsymbol{f}_1^s$)

$$\boldsymbol{f}^s(e_4) = \boldsymbol{f}_1^s(e_4) + \boldsymbol{f}_2^s(e_4) = \boldsymbol{f}^p(e) + (u - \boldsymbol{f}^p(e)) = u,$$

$$\boldsymbol{f}^s(\hat{e}_4) = \boldsymbol{f}_1^s(\hat{e}_4) + \boldsymbol{f}_2^s(\hat{e}_4) = \boldsymbol{f}^p(\hat{e}) + (u - \boldsymbol{f}^p(\hat{e})) = u.$$

- For any non-homologous edge $e' \in E^p$, we also have $e' \in E^s$ since no reduction is made on this edge, and we copy it trivially to G^s. We set

  $$\boldsymbol{f}_1^s(e') = \boldsymbol{f}^p(e'), \qquad \boldsymbol{f}_2^s(e') = 0.$$

  Since $\boldsymbol{f}^p$ is a feasible flow in G^p, it is easy to check that $\boldsymbol{f}^s$ also satisfies the selective constraint for commodity 1 and the capacity constraint on non-homologous edges, as well as conservation of flows on vertices incident to non-homologous edges.

To conclude, $\boldsymbol{f}^s$ is a feasible flow to the SFF instance. □

Definition 3.6.9 (SFF Approximate Problem (SFFA))**.** An SFFA instance is given by an SFF instance $(G, F, S_1, S_2, \boldsymbol{u}, s_1, t_1, s_2, t_2)$ as in Definition 3.3.14, and an error parameter $\epsilon \in [0, 1]$, which we collect in a tuple $(G, F, S_1, S_2, \boldsymbol{u}, s_1, t_1, s_2, t_2, \epsilon)$. We say an algorithm solves the SFFA problem, if, given any SFFA instance, it returns a pair of flows $\boldsymbol{f}_1, \boldsymbol{f}_2 \geq \boldsymbol{0}$ that satisfies

$$\boldsymbol{u}(e) - \epsilon \leq \boldsymbol{f}_1(e) + \boldsymbol{f}_2(e) \leq \boldsymbol{u}(e) + \epsilon, \quad \forall e \in F, \qquad \text{(error in congestion)}$$

$$0 \leq \boldsymbol{f}_1(e) + \boldsymbol{f}_2(e) \leq \boldsymbol{u}(e) + \epsilon, \quad \forall e \in E \backslash F, \qquad \text{(error in congestion)}$$

$$\left| \sum_{u:(u,v)\in E} \boldsymbol{f}_i(u, v) - \sum_{w:(v,w)\in E} \boldsymbol{f}_i(v, w) \right| \leq \epsilon, \quad \forall v \in V \backslash \{s_i, t_i\}, \ i \in \{1, 2\}, \qquad \text{(error in demand)}$$

$$\boldsymbol{f}_{\bar{i}}(e) \leq \epsilon, \quad \forall e \in S_i, \ \bar{i} = \{1, 2\} \backslash I, \qquad \text{(error in selection)}$$

or it correctly declares that the associated SFF instance is infeasible.

Lemma 3.6.10 (FPHFA to SFFA)**.** *Given an* FPHFA *instance* $(G^p, F^p, \mathcal{H}^p, \boldsymbol{u}^p, s, t, \epsilon^p)$*, we can reduce it to an* SFFA *instance* $(G^s, F^s, S_1, S_2, \boldsymbol{u}^s, s_1, t_1, s_2, t_2, \epsilon^s)$ *by letting*

$$\epsilon^s = \Omega\left(\frac{1}{|E^p|}\right) \epsilon^p,$$

and using Lemma 3.6.8 to construct an SFF *instance* $(G^s, F^s, S_1, S_2, \boldsymbol{u}^s, s_1, t_1, s_2, t_2)$ *from the* FPHF *instance* $(G^p, F^p, \mathcal{H}^p, \boldsymbol{u}^p, s, t)$*. If* $\boldsymbol{f}^s$ *is a solution to the* SFFA *(*SFF*) instance, then in time* $O(|E^p|)$*, we can compute a solution* $\boldsymbol{f}^p$ *to the* FPHFA *(*FPHF*, respectively) instance, where the exact case holds when* $\epsilon^s = \epsilon^p = 0$*.*

Proof. To bound error in congestion, we have $\epsilon_u^p \leq O(\epsilon^s)$ according to Lemma 3.6.3. For ϵ_l^p, we have $\epsilon_l^p = 0$ if $e \in E^p \backslash F^p$, whereas if $e \in F^p$, we have

$$\begin{aligned}
\boldsymbol{f}^p(e) &= \boldsymbol{f}_1^s(e) && \text{by solution mapping for non-homologous edges}\\
&\geq \boldsymbol{f}^s(e) - \epsilon^s && \text{by error in selection}\\
&\geq \boldsymbol{u}^s(e) - 2\epsilon^s && \text{by error in congestion}\\
&= \boldsymbol{u}^p(e) - 2\epsilon^s.
\end{aligned}$$

Hence, $\epsilon_l^p \leq 2\epsilon^s$.

To bound error in demand, by Lemma 3.6.4 and solution mapping algorithm, we have

$$\epsilon_d^p \leq \epsilon^s + \max_{w \in V^p \backslash \{s,t\}} \sum_{e=(v,w) \in H^p} \left|\boldsymbol{f}_1^s(e_1) - \boldsymbol{f}_1^s(e_2)\right|, \tag{3.3}$$

so the key is to bound $|\boldsymbol{f}_1^s(e_1) - \boldsymbol{f}_1^s(e_2)|$. Again, we can decompose the difference into several error terms.

$$\begin{aligned}
|\boldsymbol{f}_1^s(e_1) - \boldsymbol{f}_1^s(e_2)| \leq\ & |\boldsymbol{f}_1^s(e_1) + \boldsymbol{f}_1^s(e_3) - \boldsymbol{f}_1^s(e_4)| + |\boldsymbol{f}_1^s(e_4) - \boldsymbol{f}_1^s(e_3) - \boldsymbol{f}_1^s(e_2)|\\
\leq\ & \epsilon^s + |\boldsymbol{f}_1^s(e_4) - \boldsymbol{f}_1^s(e_3) - \boldsymbol{f}_1^s(e_2)| && \text{by error in demand of vertex } vw\\
\leq\ & \epsilon^s + |\boldsymbol{f}_1^s(e_4) - \boldsymbol{f}_1^s(e_2) - \boldsymbol{f}_1^s(e_5)| + |\boldsymbol{f}_1^s(e_5) - \boldsymbol{f}_1^s(e_3)|\\
\leq\ & 2\epsilon^s + |\boldsymbol{f}_1^s(e_5) - \boldsymbol{f}_1^s(e_3)| && \text{by error in demand of vertex } vw'\\
\leq\ & 2\epsilon^s + \max\{\boldsymbol{f}_1^s(e_5), \boldsymbol{f}_1^s(e_3)\}\\
\leq\ & 3\epsilon^s, && \text{by error in type of edge } e_3 \text{ and } e_5
\end{aligned}$$

Since $|H^p| = O(|E^p|)$, plugging the above back to Eq. (3.3), we have $\epsilon_d^p \leq O(|E^p|)\epsilon^s$.

Finally, we bound error in pairwise homology in G^p. For any pair of homologous edges,

$$\begin{aligned}
|\boldsymbol{f}_1^s(e_1) - \boldsymbol{f}_1^s(\hat{e}_1)| &= |(\boldsymbol{f}^s(e_1) - \boldsymbol{f}_2^s(e_1)) - (\boldsymbol{f}^s(\hat{e}_1) - \boldsymbol{f}_2^s(\hat{e}_1))|\\
&\leq |\boldsymbol{f}^s(e_1) - \boldsymbol{f}^s(\hat{e}_1)| + |\boldsymbol{f}_2^s(e_1) - \boldsymbol{f}_2^s(\hat{e}_1)|\\
&\leq \epsilon^s + |\boldsymbol{f}^s(e_1) - \boldsymbol{f}^s(\hat{e}_1)| && \text{by error in type of edge } e_1 \text{ and } \hat{e}_1\\
&\leq \epsilon^s + |(\boldsymbol{f}^s(e_1) + \boldsymbol{f}^s(e_3)) - (\boldsymbol{f}^s(\hat{e}_1) + \boldsymbol{f}^s(\hat{e}_3))| + |\boldsymbol{f}^s(e_3) - \boldsymbol{f}^s(\hat{e}_3)|.
\end{aligned}$$

To analyze $|(\boldsymbol{f}^s(e_1) + \boldsymbol{f}^s(e_3)) - (\boldsymbol{f}^s(\hat{e}_1) + \boldsymbol{f}^s(\hat{e}_3))|$, we decompose as follows:

$$\begin{aligned}
& |(\boldsymbol{f}^s(e_1) + \boldsymbol{f}^s(e_3)) - (\boldsymbol{f}^s(\hat{e}_1) + \boldsymbol{f}^s(\hat{e}_3))|\\
\leq\ & |\boldsymbol{f}^s(e_1) + \boldsymbol{f}^s(e_3) - \boldsymbol{f}^s(e_4)| + |\boldsymbol{f}^s(e_4) - (\boldsymbol{f}^s(\hat{e}_1) + \boldsymbol{f}^s(\hat{e}_3))|\\
\leq\ & 2\epsilon^s + |\boldsymbol{f}^s(e_4) - (\boldsymbol{f}^s(\hat{e}_1) + \boldsymbol{f}^s(\hat{e}_3))|\\
& && \text{by error in demand of vertex } vw \text{ for two commodities}\\
\leq\ & 2\epsilon^s + |\boldsymbol{f}^s(\hat{e}_4) - (\boldsymbol{f}^s(\hat{e}_1) + \boldsymbol{f}^s(\hat{e}_3))| + |\boldsymbol{f}^s(e_4) - \boldsymbol{f}^s(\hat{e}_4)|\\
\leq\ & 4\epsilon^s + |\boldsymbol{f}^s(e_4) - \boldsymbol{f}^s(\hat{e}_4)| && \text{by error in demand of vertex } yz \text{ for two commodities}\\
\leq\ & 6\epsilon^s, && \text{by error in congestion on } e_4 \text{ and } \hat{e}_4
\end{aligned}$$

For $|\boldsymbol{f}^s(e_3) - \boldsymbol{f}^s(\hat{e}_3)|$, similarly, we have

$$\begin{aligned}
|\boldsymbol{f}^s(e_3) - \boldsymbol{f}^s(\hat{e}_3)| &= |(\boldsymbol{f}_1^s(e_3) + \boldsymbol{f}_2^s(e_3)) - (\boldsymbol{f}_1^s(\hat{e}_3) + \boldsymbol{f}_2^s(\hat{e}_3))|\\
&\leq |\boldsymbol{f}_1^s(e_3) - \boldsymbol{f}_1^s(\hat{e}_3)| + |\boldsymbol{f}_2^s(e_3) - \boldsymbol{f}_2^s(\hat{e}_3)|\\
&\leq \epsilon^s + |\boldsymbol{f}_2^s(e_3) - \boldsymbol{f}_2^s(\hat{e}_3)| && \text{by error in type in } G^s \text{ and } e_3, \hat{e}_3\\
&\leq 4\epsilon^s, && \text{by symmetry from the bound of } |\boldsymbol{f}_1^s(e_1) - \boldsymbol{f}_1^s(e_2)|
\end{aligned}$$

Therefore, we can bound the error in pair homology by

$$\epsilon_h^p \leq 11\epsilon^s.$$

Putting it altogether, we have

$$\epsilon^p = \max\{\epsilon_l^p, \epsilon_u^p, \epsilon_d^p, \epsilon_h^p\} \leq O\left(|E^p|\right)\epsilon^s.$$

□

3.6.4 SFF(A) to 2CFF(A): Dropping Selective Flow Constraints

We show the reduction from an SFF instance $(G^s, F^s, S_1, S_2, \boldsymbol{u}^s, s_1, t_1, s_2, t_2)$ to a 2CFF instance $(G^f, F^f, \boldsymbol{u}^f, s_1, t_1, s_2, t_2)$. Assume that $e \in S_i$ is an arbitrary selective edge for commodity i in G^p. As shown in Figure 3.6, we map e in G^s to a gadget consisting of edges $\{e_1, e_2, e_3, e_4, e_5\}$ in G^f. Note that a selective edge e can be either a fixed flow edge or a non-fixed flow edge. In Figure 3.6, $l = u$ if e is a fixed flow edge and $l = 0$ if e is a non-fixed flow edge. Moreover, no reduction is performed on non-selective edges in G^s, and we trivially copy these edges to G^f.

The key idea to remove the selective requirement is utilizing edge directions and the source-sink pair (s_i, t_i) to simulate a selective edge e for commodity i. More specifically, in the gadget, the flow of commodity i routes through three directed paths: (1) $e_1 \to e_4$, (2) $e_5 \to e_3 \to e_4$, (3) $e_5 \to e_2$. The selective requirement is realized because e_4 is the only outgoing edge of xy and only the flow of commodity i is allowed in e_4 (since its tail is t_i), thus in e_1. Similarly, e_5 is the only incoming edge of xy' and only the flow of commodity i is allowed in e_5, thus in e_2. In addition, to ensure that e_1 and e_2 route the same amount of flow, flows in e_4 and e_5 must be equal by the conservation of flows. Therefore, we impose the fixed flow constraint on e_4 and e_5 by setting the fixed flow to be u. We remove e_3 if e is a fixed flow edge (in which case e_3 has capacity 0), and set the capacity of e_3 to be u otherwise (in which case $u - l = u - 0 = u$).

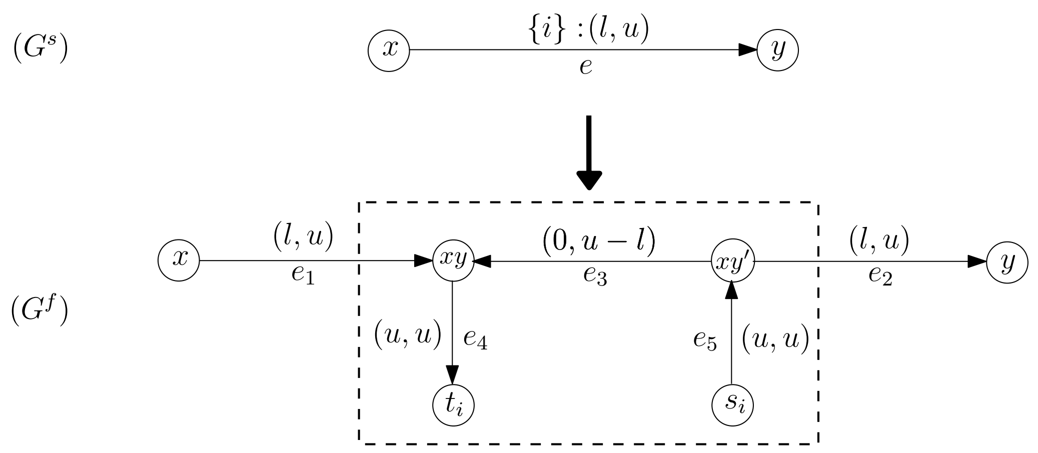

Figure 3.6: The reduction from SFF to 2CFF. $l = u$ if e is a fixed flow edge, and $l = 0$ if e is a non-fixed flow edge.

Lemma 3.6.11 (SFF to 2CFF)**.** *Given an* SFF *instance* $(G^s, F^s, S_1, S_2, \boldsymbol{u}^s, s_1, t_1, s_2, t_2)$*, we can construct, in time* $O(|E^s|)$*, a* 2CFF *instance* $(G^f, F^f, \boldsymbol{u}^f, s_1, t_1, s_2, t_2)$ *such that*

$$\left|E^f\right| = O\left(|E^s|\right), \quad \left\|\boldsymbol{u}^f\right\|_{\max} = \left\|\boldsymbol{u}^s\right\|_{\max}.$$

If the SFF *instance has a solution, then the* 2CFF *instance has a solution.*

Proof. If $\boldsymbol{f}^s$ is a solution to the SFF instance, it is easy to derive a solution $\boldsymbol{f}^f$ to the 2CFF instance as follows:

- For any selective edge $e \in S_i$, in its corresponding gadget in G^f, we set

  $$l \le \boldsymbol{f}_i^f(e_1) = \boldsymbol{f}_i^f(e_3) = \boldsymbol{f}_i^s(e) \le u,$$
  $$\boldsymbol{f}_i^f(e_4) = \boldsymbol{f}_i^f(e_5) = u,$$
  $$0 \le \boldsymbol{f}_i^f(e_2) = u - \boldsymbol{f}_i^s(e) \le u - l,$$

  where $l = 0$ if e is a non-fixed flow edge, or $l = u = \boldsymbol{u}^s(e)$ if e is a fixed flow edge. And we set

  $$\boldsymbol{f}_{\bar{i}}^f(e_1) = \boldsymbol{f}_{\bar{i}}^f(e_2) = \boldsymbol{f}_{\bar{i}}^f(e_3) = \boldsymbol{f}_{\bar{i}}^f(e_4) = \boldsymbol{f}_{\bar{i}}^f(e_5) = 0, \qquad \bar{i} = \{1,2\}\backslash i.$$

  It is obvious that $\boldsymbol{f}^f$ satisfies the capacity constraint on edges $(e_1, \ldots, e_5)$, and the flow conservation constraint on vertices xy, xy'.

- For any non-selective edge $e' \in E^s \backslash (S_1 \cup S_2)$, we also have $e' \in E^f$ since no reduction is made on this edge, and we copy it trivially to G^f. We set

  $$\boldsymbol{f}_i^f(e') = \boldsymbol{f}_i^s(e'), \qquad i \in \{1,2\}.$$

  Since $\boldsymbol{f}^s$ is a feasible flow in G^s, it is easy to check that $\boldsymbol{f}^f$ also satisfies the capacity constraint on non-selective edges, as well as the flow conservation constraint on vertices incident to non-selective edges. Moreover, if e' is a fixed flow edge, the fixed flow edge constraint is also satisfied.

To conclude, $\boldsymbol{f}^f$ is a feasible flow to the 2CFF instance. □

Definition 3.6.12 (2CFF Approximate Problem (2CFFA))**.** A 2CFFA instance is given by a 2CFF instance $(G, F, \boldsymbol{u}, s_1, t_1, s_2, t_2)$ as in Definition 3.3.15, and an error parameter $\epsilon \in [0,1]$, which we collect in a tuple $(G, F, \boldsymbol{u}, s_1, t_1, s_2, t_2, \epsilon)$. We say an algorithm solves the 2CFFA problem, if, given any 2CFFA instance, it returns a pair of flows $\boldsymbol{f}_1, \boldsymbol{f}_2 \ge \boldsymbol{0}$ that satisfies

$$\boldsymbol{u}(e) - \epsilon \le \boldsymbol{f}_1(e) + \boldsymbol{f}_2(e) \le \boldsymbol{u}(e) + \epsilon, \quad \forall e \in F, \qquad \text{(error in congestion)}$$
$$0 \le \boldsymbol{f}_1(e) + \boldsymbol{f}_2(e) \le \boldsymbol{u}(e) + \epsilon, \quad \forall e \in E\backslash F, \qquad \text{(error in congestion)}$$
$$\left|\sum_{u:(u,v)\in E} \boldsymbol{f}_i(u,v) - \sum_{w:(v,w)\in E} \boldsymbol{f}_i(v,w)\right| \le \epsilon, \quad \forall v \in V\backslash\{s_i, t_i\}, \; i \in \{1,2\}, \qquad \text{(error in demand)}$$

or it correctly declares that the associated 2CFF instance is infeasible.

Lemma 3.6.13 (SFFA to 2CFFA)**.** *Given an* SFFA *instance* $(G^s, F^s, S_1, S_2, \boldsymbol{u}^s, s_1, t_1, s_2, t_2, \epsilon^s)$, *we can reduce it to a* 2CFFA *instance* $\left(G^f, F^f, \boldsymbol{u}^f, s_1, t_1, s_2, t_2, \epsilon^f\right)$ *by letting*

$$\epsilon^f = \Omega\left(\frac{1}{|E^p|}\right)\epsilon^s,$$

and using Lemma 3.6.11 to construct a 2CFF *instance* $\left(G^f, F^f, \boldsymbol{u}^f, s_1, t_1, s_2, t_2\right)$ *from the* SFF *instance* $(G^s, F^s, S_1, S_2, \boldsymbol{u}^s, s_1, t_1, s_2, t_2)$*. If* $\boldsymbol{f}^f$ *is a solution to the* 2CFFA *(*2CFF*) instance, then in time* $O(|E^s|)$*, we can compute a solution* $\boldsymbol{f}^s$ *to the* SFFA *(*SFF*, respectively) instance, where the exact case holds when* $\epsilon^f = \epsilon^s = 0$*.*

Proof. To bound error in congestion, we have $\epsilon_u^s \le O(\epsilon^f)$ according to Lemma 3.6.3. For ϵ_l^s, we have $\epsilon_l^s = 0$ if $e \in E^s \backslash F^s$. If $e \in F^s$, since we just copy them without making reductions for fixed flow edges in this step, we also have $\epsilon_l^s \le O(\epsilon^f)$.

To bound error in demand, we analyze commodity 1 and commodity 2 separately. Let S_i be the set of selective edges for commodity i, $i \in \{1,2\}$, and let $S = S_1 \cup S_2$. Also, let $\epsilon_{d,i}^s$ be the error in demand for commodity i. By Lemma 3.6.4 and solution mapping algorithm, we have

$$\begin{aligned}\epsilon_{d,i}^s &\le \epsilon^f + \max_{w\in V^p\backslash\{s_i,t_i\}} \sum_{e=(v,w)\in S} |\boldsymbol{f}_i^s(e_1) - \boldsymbol{f}_i^s(e_2)| \\ &= \epsilon^f + \max_{w\in V^p\backslash\{s_i,t_i\}}\left(\sum_{e=(v,w)\in S_i}\left|\boldsymbol{f}_i^f(e_1) - \boldsymbol{f}_i^f(e_2)\right| + \sum_{e=(v,w)\in S_{\bar{i}}}\left|\boldsymbol{f}_i^f(e_1) - \boldsymbol{f}_i^f(e_2)\right|\right).\end{aligned} \tag{3.4}$$

The problem is reduced to bounding $\left|\boldsymbol{f}_i^f(e_1) - \boldsymbol{f}_i^f(e_2)\right|$.

$$\begin{aligned}&\left|\boldsymbol{f}_i^f(e_1) - \boldsymbol{f}_i^f(e_2)\right| \\ \le\ &\left|\boldsymbol{f}_i^f(e_1) + \boldsymbol{f}_i^f(e_3) - \boldsymbol{f}_i^f(e_4)\right| + \left|\boldsymbol{f}_i^f(e_4) - \boldsymbol{f}_i^f(e_3) - \boldsymbol{f}_i^f(e_2)\right| \\ \le\ &\epsilon^f + \left|\boldsymbol{f}_i^f(e_4) - \boldsymbol{f}_i^f(e_3) - \boldsymbol{f}_i^f(e_2)\right| && \text{by error in demand of vertex } xy \\ \le\ &\epsilon^f + \left|\boldsymbol{f}_i^f(e_4) - \boldsymbol{f}_i^f(e_5)\right| + \left|\boldsymbol{f}_i^f(e_5) - \boldsymbol{f}_i^f(e_3) - \boldsymbol{f}_i^f(e_2)\right| \\ \le\ &2\epsilon^f + \left|\boldsymbol{f}_i^f(e_4) - \boldsymbol{f}_i^f(e_5)\right|, && \text{by error in demand of vertex } xy'\end{aligned}$$

If $e \in S_{\bar{i}}$, then, by error in demand of vertex $t_{\bar{i}}$ and $s_{\bar{i}}$, we have

$$\left|\boldsymbol{f}_i^f(e_4) - \boldsymbol{f}_i^f(e_5)\right| \le 2\epsilon^f.$$

However, if $e \in S_i$, we have

$$\begin{aligned}\left|\boldsymbol{f}_i^f(e_4) - \boldsymbol{f}_i^f(e_5)\right| &= \left|\left(\boldsymbol{f}^f(e_4) - \boldsymbol{f}_{\bar{i}}^f(e_4)\right) - \left(\boldsymbol{f}^f(e_5) - \boldsymbol{f}_{\bar{i}}^f(e_5)\right)\right| \\ &\le \left|\boldsymbol{f}^f(e_4) - \boldsymbol{f}^f(e_5)\right| + \left|\boldsymbol{f}_{\bar{i}}^f(e_4) - \boldsymbol{f}_{\bar{i}}^f(e_5)\right| \\ &\le 2\epsilon^f + \left|\boldsymbol{f}_{\bar{i}}^f(e_4) - \boldsymbol{f}_{\bar{i}}^f(e_5)\right| && \text{by error in congestion of edge } e_4 \text{ and } e_5 \\ &\le 4\epsilon^f, && \text{by error in demand of vertex } t_i \text{ and } s_i\end{aligned}$$

Hence, since $|S| = O(|E^s|)$, plugging the above back to Eq. (3.4), we have

$$\epsilon_d^s = \max\{\epsilon_{d,1}^s, \epsilon_{d,2}^s\} \leq O(|E^s|)\epsilon^f.$$

Finally, we bound the error in selection in G^s. Suppose $e \in S_i$, we have

$$\begin{aligned}
\boldsymbol{f}_{\bar{i}}^s(e) &= \boldsymbol{f}_{\bar{i}}^f(e_1) \\
&\leq \boldsymbol{f}_{\bar{i}}^f(e_1) + \boldsymbol{f}_{\bar{i}}^f(e_3) && \text{by non-negativity constraint} \\
&\leq \boldsymbol{f}_{\bar{i}}^f(e_4) + \left|\boldsymbol{f}_{\bar{i}}^f(e_1) + \boldsymbol{f}_{\bar{i}}^f(e_3) - \boldsymbol{f}_{\bar{i}}^f(e_4)\right| \\
&\leq \epsilon^f + \left|\boldsymbol{f}_{\bar{i}}^f(e_1) + \boldsymbol{f}_{\bar{i}}^f(e_3) - \boldsymbol{f}_{\bar{i}}^f(e_4)\right|, && \text{by error in demand of vertex } t_i \\
&\leq 2\epsilon^f, && \text{by error in demand of vertex } xy
\end{aligned}$$

Therefore, we can bound the error in type by

$$\epsilon_s^s \leq 2\epsilon^f.$$

Putting it all together, we have

$$\epsilon^s = \max\{\epsilon_l^s, \epsilon_u^s, \epsilon_d^s, \epsilon_s^s\} \leq O\left(|E^s|\right)\epsilon^f.$$

□

3.6.5 2CFF(A) to 2CFR(A): Dropping Fixed Flow Constraints

In this section, we show the reduction from a 2CFF instance $(G^f, F^f, \boldsymbol{u}^f, s_1, t_1, s_2, t_2)$ to a 2CFR instance $(G^r, \boldsymbol{u}^r, \bar{s}_1, \bar{t}_1, \bar{s}_2, \bar{t}_2, R_1, R_2)$. First of all, we add two new sources $\bar{s}_1, \bar{s}_2$ and two new sinks $\bar{t}_1, \bar{t}_2$. Then, for each edge $e \in E^f$, we map it to a gadget consisting edges $\{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$ in G^r, as shown in the upper part of Figure 3.7. Additionally, there is another gadget with 5 edges in G^r that connects the original sink t_i and source s_i, as shown in the lower part of Figure 3.7. Capacity of these edges is the sum capacities of all edges in G^f, i.e., $M^f = \sum_{e \in E^f} \boldsymbol{u}^f(e)$. Additionally, we set $R_1 = R_2 = 2M^f$, indicating that at least $2M^f$ unit of flow should be routed from $\bar{s}_i$ to $\bar{t}_i$, $i \in \{1, 2\}$.

The key idea to remove the fixed flow constraint is utilizing edge directions and the requirements that $2M^f$ units of the flow of commodity i to be routed from the new source $\bar{s}_i$ to the new sink $\bar{t}_i$, $i \in \{1, 2\}$. It is noticed that all edges that are incident to the new sources and sinks should be saturated to fulfill the requirements. Therefore, for a fixed flow edge e in the first gadget, the incoming flow of vertex xy' and the outgoing flow of vertex xy must be $2u$. Since the capacity of e_3 is $2u - u = u$, then the flows of e_1, e_2, e_3 are forced to be u. As such, the fixed flow constraint can be simulated. Note that instead of simply copying non-fixed flow edges to G^r, we also need to map non-fixed flow edges to the designed gadget in this step. This guarantees that the requirement on flow values can be satisfied if the 2CFF instance is feasible.

Lemma 3.6.14 (2CFF to 2CFR). *Given a 2CFF instance $(G^f, F^f, \boldsymbol{u}^f, s_1, t_1, s_2, t_2)$, we can construct, in time $O(|E^f|)$, a 2CFR instance $(G^r, \boldsymbol{u}^r, \bar{s}_1, \bar{t}_1, \bar{s}_2, \bar{t}_2, R_1, R_2)$ such that*

$$|E^r| = O\left(\left|E^f\right|\right), \quad \|\boldsymbol{u}^r\|_{\max} = M^f,$$

where $M^f = \sum_{e \in E^f} \boldsymbol{u}^f(e)$. If the 2CFF instance has a solution, then the 2CFR instance has a solution.

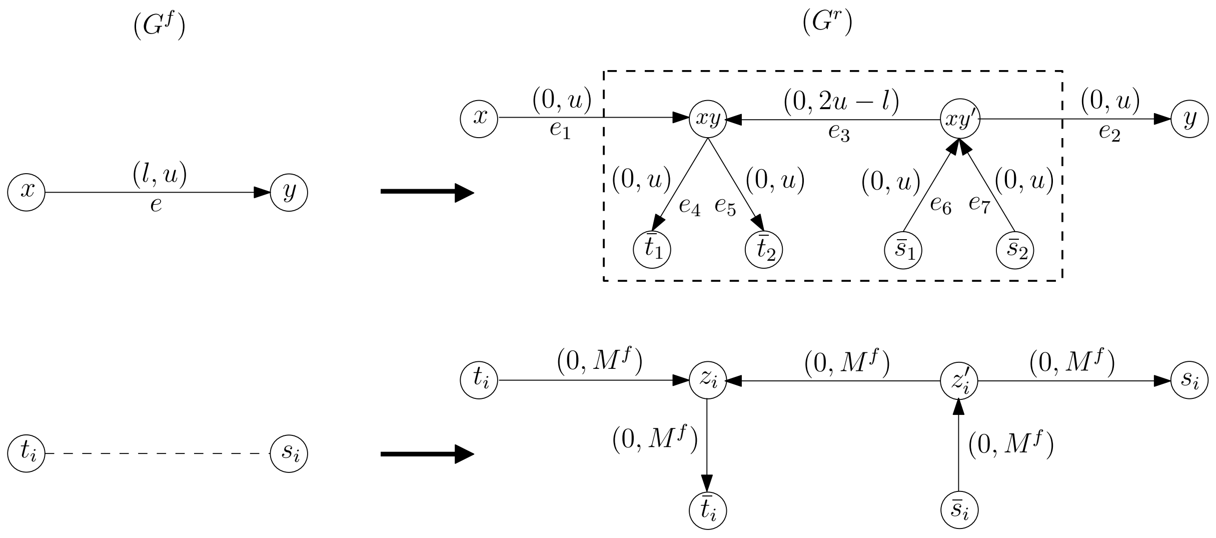

Figure 3.7: The reduction from 2CFF to 2CFR. $l = u$ if e is a fixed flow edge, and $l = 0$ if e is a non-fixed flow edge.

Proof. Suppose $\boldsymbol{f}^f$ is a feasible solution to the 2CFF instance, then we can derive a feasible solution $\boldsymbol{f}^r$ to the 2CFR instance. Concretely, we define a feasible flow $\boldsymbol{f}^r$ as follows. For any edge $e \in E^f$, we set:

$$\boldsymbol{f}_i^r(e_1) = \boldsymbol{f}_i^r(e_3) = \boldsymbol{f}_i^f(e), \qquad i \in \{1,2\},$$

$$\boldsymbol{f}_i^r(e_2) = u - \boldsymbol{f}_i^f(e), \qquad i \in \{1,2\},$$

$$\boldsymbol{f}_1^r(e_4) = \boldsymbol{f}_1^r(e_6) = u, \qquad \boldsymbol{f}_2^r(e_5) = \boldsymbol{f}_2^r(e_7) = u,$$

$$\boldsymbol{f}_2^r(e_4) = \boldsymbol{f}_2^r(e_6) = \boldsymbol{f}_1^r(e_5) = \boldsymbol{f}_1^r(e_7) = 0,$$

where $u = \boldsymbol{u}^f(e)$. And we set

$$\boldsymbol{f}_i^r(t_i, z_i) = \boldsymbol{f}_i^r(z_i', s_i) = \sum_{w:(s_i,w)\in E^f} \boldsymbol{f}_i^f(s_i, w) \leq M^f, \qquad i \in \{1,2\},$$

$$\boldsymbol{f}_i^r(z_i', z_i) = M^f - \sum_{w:(s_i,w)\in E^f} \boldsymbol{f}_i^f(s_i, w), \qquad i \in \{1,2\},$$

$$\boldsymbol{f}_i^r(\bar{s}_i, z_i') = \boldsymbol{f}_i^r(z_i, \bar{t}_i) = M^f, \qquad i \in \{1,2\},$$

$$\boldsymbol{f}_{\bar{i}}^r(t_i, z_i) = \boldsymbol{f}_{\bar{i}}^r(z_i', s_i) = \boldsymbol{f}_{\bar{i}}^r(z_i', z_i) = \boldsymbol{f}_{\bar{i}}^r(\bar{s}_i, z_i') = \boldsymbol{f}_{\bar{i}}^r(z_i, \bar{t}_i) = 0, \qquad \bar{i} \in \{1,2\}\backslash i.$$

To prove that $\boldsymbol{f}^r$ is a solution to the 2CFR instance, it is obvious that capacity constraint and flow conservation constraint are satisfied by construction. For the satisfaction of the requirement constraint $R_1 = R_2 = 2M^f$, we have, for commodity 1,

$$R_1 = \sum_{w:(\bar{s}_1,w)\in E^r} \boldsymbol{f}_1^r(\bar{s}_1, w) = \boldsymbol{f}^r(\bar{s}_1, z_1') + \sum_{e\in E^f} \boldsymbol{f}_1^r(e_6) = M^f + \sum_{e\in E^f} \boldsymbol{u}^f(e) = 2M^f,$$

$$R_1 = \sum_{u:(u,\bar{t}_1)\in E^r} \boldsymbol{f}_1^r(u, \bar{t}_1) = \boldsymbol{f}^r(z_1, \bar{t}_1) + \sum_{e\in E^f} \boldsymbol{f}_1^r(e_4) = M^f + \sum_{e\in E^f} \boldsymbol{u}^f(e) = 2M^f.$$

We can prove that the requirement $R_2 = 2M^f$ is satisfied similarly. Therefore, $\boldsymbol{f}^r$ is a feasible flow to the 2CFR instance. □

Definition 3.6.15 (2CFR Approximate Problem (2CFRA))**.** A 2CFRA instance is given by a 2CFR instance $(G, \boldsymbol{u}, s_1, t_1, s_2, t_2, R_1, R_2)$ as in Definition 3.3.8, and an error parameter $\epsilon \in [0, 1]$, which we collect in a tuple $(G, \boldsymbol{u}, s_1, t_1, s_2, t_2, R_1, R_2, \epsilon)$. We say an algorithm solves the 2CFRA problem, if, given any 2CFRA instance, it returns a pair of flows $\boldsymbol{f}_1, \boldsymbol{f}_2 \geq \boldsymbol{0}$ that satisfies

$$\boldsymbol{f}_1(e) + \boldsymbol{f}_2(e) \leq \boldsymbol{u}(e) + \epsilon, \ \forall e \in E, \qquad \text{(error in congestion)}$$

$$\left| \sum_{u:(u,v)\in E} \boldsymbol{f}_i(u,v) - \sum_{w:(v,w)\in E} \boldsymbol{f}_i(v,w) \right| \leq \epsilon, \ \forall v \in V \backslash \{s_i, t_i\}, i \in \{1,2\}, \quad \text{(error in demand)}$$

$$\left| \sum_{w:(s_i,w)\in E} \boldsymbol{f}_i(s_i,w) - R_i \right| \leq \epsilon, \qquad \left| \sum_{u:(u,t_i)\in E} \boldsymbol{f}_i(u,t_i) - R_i \right| \leq \epsilon, \ i \in \{1,2\},$$

$$\text{(error in demand)}$$

or it correctly declares that the associated 2CFR instance is infeasible.

Lemma 3.6.16 (2CFFA to 2CFRA)**.** *Given a 2CFFA instance $\left(G^f, F^f, \boldsymbol{u}^f, s_1, t_1, s_2, t_2, \epsilon^f\right)$, we can reduce it to a 2CFRA instance $(G^r, \boldsymbol{u}^r, \bar{s}_1, \bar{t}_1, \bar{s}_2, \bar{t}_2, R_1, R_2, \epsilon^r)$ by letting*

$$\epsilon^r = \Omega\left(\frac{1}{|E^f|}\right)\epsilon^f.$$

and using Lemma 3.6.14 to construct a 2CFR instance $(G^r, \boldsymbol{u}^r, \bar{s}_1, \bar{t}_1, \bar{s}_2, \bar{t}_2, R_1, R_2)$ from the SFF instance $\left(G^f, F^f, \boldsymbol{u}^f, s_1, t_1, s_2, t_2\right)$. If $\boldsymbol{f}^r$ is a solution to the 2CFRA (2CFR) instance, then in time $O(|E^f|)$, we can compute a solution $\boldsymbol{f}^f$ to the 2CFFA (2CFF, respectively) instance, where the exact case holds when $\epsilon^r = \epsilon^f = 0$.

Proof. To bound error in congestion, we have $\epsilon_u^f \leq \epsilon^r$ by Lemma 3.6.3. Regarding to ϵ_l^f, we have $\epsilon_l^f = 0$ if $e \in E^f \backslash F^f$; whereas for an arbitrary fixed flow edge $e \in F^f$, by solution mapping, we have

$$\begin{aligned}
\boldsymbol{f}^f(e) &= \boldsymbol{f}^r(e_1) = \boldsymbol{f}_1^r(e_1) + \boldsymbol{f}_2^r(e_1) \\
&\geq \sum_{i\in\{1,2\}} (\boldsymbol{f}_i^r(e_4) + \boldsymbol{f}_i^r(e_5) - \boldsymbol{f}_i^r(e_3) - \epsilon^r) \\
&\geq \boldsymbol{f}_1^r(e_4) + \boldsymbol{f}_2^r(e_5) - \boldsymbol{f}^r(e_3) - 2\epsilon^r,
\end{aligned} \tag{3.5}$$

where the last step follows from the non-negativity of $\boldsymbol{f}_1^r(e_5)$ and $\boldsymbol{f}_2^r(e_4)$.

Now, to bound $\boldsymbol{f}_1^r(e_4)$ and $\boldsymbol{f}_2^r(e_5)$, we have to make use of the requirement constraint of $\bar{t}_1, \bar{t}_2$. By definition,

$$\left| \sum_{e\in E^f} \boldsymbol{f}_1^r(e_4) + \boldsymbol{f}^r(z_1, \bar{t}_1) - 2M^f \right| \leq \epsilon^r,$$

$$\left| \sum_{e\in E^f} \boldsymbol{f}_2^r(e_5) + \boldsymbol{f}^r(z_2, \bar{t}_2) - 2M^f \right| \leq \epsilon^r.$$

Rearranging and taking the lower bound direction, we have

$$\begin{aligned}
\boldsymbol{f}_1^r(e_4) &\geq 2M^f - \epsilon^r - \sum_{\hat{e}\in E^f\setminus e} \boldsymbol{f}_1^r(e_4) - \boldsymbol{f}^r(z_1, \bar{t}_1) \\
&\geq \max\left\{0, 2M^f - \epsilon^r - \left(M^f - \boldsymbol{u}^f(e) + (|E^f|-1)\epsilon^r\right) - (M^f + \epsilon^r)\right\} \\
&= \max\left\{0, \boldsymbol{u}^f(e) - (|E^f|+1)\epsilon^r\right\}.
\end{aligned}$$

Similarly, we have $\boldsymbol{f}_2^r(e_5) \geq \max\left\{0, \boldsymbol{u}^f(e) - (|E^f|+1)\epsilon^r\right\}$.

Plugging back to Eq. (3.5),

$$\begin{aligned}
\boldsymbol{f}^f(e) &\geq \max\{0, 2\left(\boldsymbol{u}^f(e) - (|E^f|+1)\epsilon^r\right) - \boldsymbol{f}^r(e_3) - 2\epsilon^r\} \\
&\qquad\qquad \text{by lower bound of } \boldsymbol{f}_1^r(e_4) \text{ and } \boldsymbol{f}_2^r(e_5) \\
&\geq \max\{0, 2\left(\boldsymbol{u}^f(\hat{e}) - (|E^f|+1)\epsilon^r\right) - \left(\boldsymbol{u}^f(e) + \epsilon^r\right) - 2\epsilon^r\} \\
&\qquad\qquad \text{by upper bound of } \boldsymbol{f}^r(e_3) \\
&= \max\{0, \boldsymbol{u}^f(e) - (2|E^f|+5)\epsilon^r\}.
\end{aligned}$$

Hence, we can bound

$$\epsilon_l^f \leq O\left(|E^f|\right)\epsilon^r.$$

Next, we analyze error in demand. By Lemma 3.6.4 and solution mapping algorithm, we have for $i \in \{1, 2\}$,

$$\epsilon_{d,i}^f \leq \epsilon^r + \max_{v\in V^f\setminus\{s_i,t_i\}} \sum_{e=(u,v)\in E^f} \left|\boldsymbol{f}_i^r(e_1) - \boldsymbol{f}_i^r(e_2)\right|. \tag{3.6}$$

The problem is reduced to bounding $|\boldsymbol{f}_i^r(e_1) - \boldsymbol{f}_i^r(e_2)|$.

$$\begin{aligned}
&|\boldsymbol{f}_i^r(e_1) - \boldsymbol{f}_i^r(e_2)| \\
\leq\ & |\boldsymbol{f}_i^r(e_1) + \boldsymbol{f}_i^r(e_3) - \boldsymbol{f}_i^r(e_4) - \boldsymbol{f}_i^r(e_5)| + |\boldsymbol{f}_i^r(e_4) + \boldsymbol{f}_i^r(e_5) - \boldsymbol{f}_i^r(e_3) - \boldsymbol{f}_i^r(e_2)| \\
\leq\ & \epsilon^r + |\boldsymbol{f}_i^r(e_4) + \boldsymbol{f}_i^r(e_5) - \boldsymbol{f}_i^r(e_3) - \boldsymbol{f}_i^r(e_2)| \qquad \text{by error in demand of vertex } xy \\
\leq\ & \epsilon^r + |\boldsymbol{f}_i^r(e_4) + \boldsymbol{f}_i^r(e_5) - \boldsymbol{f}_i^r(e_6) - \boldsymbol{f}_i^r(e_7)| + |\boldsymbol{f}_i^r(e_6) + \boldsymbol{f}_i^r(e_7) - \boldsymbol{f}_i^r(e_3) - \boldsymbol{f}_i^r(e_2)| \\
\leq\ & 2\epsilon^r + |\boldsymbol{f}_i^r(e_4) + \boldsymbol{f}_i^r(e_5) - \boldsymbol{f}_i^r(e_6) - \boldsymbol{f}_i^r(e_7)|, \qquad \text{by error in demand of vertex } xy'
\end{aligned}$$

By error in demand of $\bar{t}_i$ and $\bar{s}_i$, we have

$$\boldsymbol{f}_1^r(e_5),\ \boldsymbol{f}_1^r(e_7) \leq \epsilon^r; \qquad \boldsymbol{f}_2^r(e_4),\ \boldsymbol{f}_2^r(e_6) \leq \epsilon^r.$$

Hence,

$$\begin{aligned}
|\boldsymbol{f}_1^r(e_1) - \boldsymbol{f}_1^r(e_2)| &\leq 4\epsilon^r + |\boldsymbol{f}_1^r(e_4) - \boldsymbol{f}_1^r(e_6)|, \\
|\boldsymbol{f}_2^r(e_1) - \boldsymbol{f}_2^r(e_2)| &\leq 4\epsilon^r + |\boldsymbol{f}_2^r(e_5) - \boldsymbol{f}_2^r(e_7)|.
\end{aligned}$$

Claim 3.6.17.

$$\sum_{e\in E^f} |\boldsymbol{f}_1^r(e_4) - \boldsymbol{f}_1^r(e_6)| \leq 6|E^f|\epsilon^r, \qquad \sum_{e\in E^f} |\boldsymbol{f}_2^r(e_5) - \boldsymbol{f}_2^r(e_7)| \leq 6|E^f|\epsilon^r.$$

Combining the above bounds and Claim 3.6.17 and plugging back to Eq. (3.6), we have

$$\epsilon_{d,1}^f,\ \epsilon_{d,2}^f \leq O\left(|E^f|\right)\epsilon^r.$$

Putting it altogether, we have

$$\epsilon^f = \max\{\epsilon_u^f, \epsilon_l^f, \epsilon_{d,1}^f, \epsilon_{d,2}^f\} \leq O\left(|E^f|\right)\epsilon^r.$$

□

Proof of Claim 3.6.17. We divide all edges in E^f into two groups:

$$E_I = \{e \in E^f \text{ s.t. } \boldsymbol{f}_1^r(e_4) \leq \boldsymbol{f}_1^r(e_6)\},$$
$$E_{II} = \{e \in E^f \text{ s.t. } \boldsymbol{f}_1^r(e_4) > \boldsymbol{f}_1^r(e_6)\}.$$

We denote

$$T_I = \sum_{e \in E_I} \boldsymbol{f}_1^r(e_4), \qquad S_I = \sum_{e \in E_I} \boldsymbol{f}_1^r(e_6);$$
$$T_{II} = \sum_{e \in E_{II}} \boldsymbol{f}_1^r(e_4), \qquad S_{II} = \sum_{e \in E_{II}} \boldsymbol{f}_1^r(e_6).$$

Then,

$$\sum_{e \in E^f} |\boldsymbol{f}_1^r(e_4) - \boldsymbol{f}_1^r(e_6)| = (S_I - T_I) + (T_{II} - S_{II}).$$

On one hand, by error in congestion, we have

$$S_I + T_{II} \leq \sum_{e \in E^f} \boldsymbol{u}^f(e) + |E^f|\epsilon^r = M^f + |E^f|\epsilon^r. \tag{3.7}$$

On the other hand, applying error in demand on vertex t_i and s_i, we have

$$T_I + T_{II} \geq (2M^f - \epsilon^r) - \boldsymbol{f}_1^r(z_1, \bar{t}_1) \geq 2M^f - \epsilon^r - M^f - \epsilon^r = M^f - 2\epsilon^r,$$
$$S_I + S_{II} \geq (2M^f - \epsilon^r) - \boldsymbol{f}_1^r(\bar{s}_1, z_1') \geq 2M^f - \epsilon^r - M^f - \epsilon^r = M^f - 2\epsilon^r.$$

Thus,

$$S_I + T_{II} = S_I + (T_I + T_{II}) - T_I \geq M^f - 2\epsilon^r + (S_I - T_I),$$
$$S_I + T_{II} = (S_I + S_{II}) + T_{II} - S_{II} \geq M^f - 2\epsilon^r + (T_{II} - S_{II}).$$

Together with Eq. (3.7), we obtain

$$S_I - T_I \leq S_I + T_{II} - M^f + 2\epsilon^r \leq M^f + |E^f|\epsilon^r - M^f + 2\epsilon^r = (|E^f| + 2)\epsilon^r,$$
$$T_{II} - S_{II} \leq S_I + T_{II} - M^f + 2\epsilon^r \leq M^f + |E^f|\epsilon^r - M^f + 2\epsilon^r = (|E^f| + 2)\epsilon^r.$$

Hence, we have

$$\sum_{e \in E^f} |\boldsymbol{f}_1^r(e_4) - \boldsymbol{f}_1^r(e_6)| = (S_I - T_I) + (T_{II} - S_{II}) \leq 2(|E^f| + 2)\epsilon^r \leq 6|E^f|\epsilon^r,$$

which finishes the proof. □

3.6.6 2CFR(A) to 2CF(A)

In this section, we show the reduction from a 2CFR instance $(G^r, \boldsymbol{u}^r, \bar{s}_1, \bar{t}_1, \bar{s}_2, \bar{t}_2, R_1, R_2)$ to a 2CF instance $(G^{2cf}, \boldsymbol{u}^{2cf}, \bar{\bar{s}}_1, \bar{t}_1, \bar{\bar{s}}_2, \bar{t}_2, R^{2cf})$. To drop the required flow constraint, we add to G^{2cf} with two new sources $\bar{\bar{s}}_1, \bar{\bar{s}}_2$ and two new edges $(\bar{\bar{s}}_1, \bar{s}_1), (\bar{\bar{s}}_2, \bar{s}_2)$ with capacity R_1, R_2, respectively. We set $R^{2cf} = R_1 + R_2$.

For solution mapping, if a 2CF solver returns $\boldsymbol{f}^{2cf}$ for the 2CF instance $(G^{2cf}, \boldsymbol{u}^{2cf}, \bar{\bar{s}}_1, \bar{t}_1, \bar{\bar{s}}_2, \bar{t}_2, R^{2cf})$, then we return $\boldsymbol{f}^r$ for the 2CFR instance $(G^r, \boldsymbol{u}^r, \bar{s}_1, \bar{t}_1, \bar{s}_2, \bar{t}_2, R_1, R_2)$ by setting $\boldsymbol{f}_i^r(e) = \boldsymbol{f}_i^{2cf}(e), \forall e \in E^r, i \in \{1, 2\}$. If the 2CF solver returns "infeasible" for the 2CF instance, then we return "infeasible" for the 2CFR instance.

Note that the required flow constraint is not an edge constraint, so conclusions from Section 3.6.1 cannot be directly applied. Instead, we analyze this step directly.

Lemma 3.6.18 (2CFR to 2CF)**.** *Given a* 2CFR *instance* $(G^r, \boldsymbol{u}^r, \bar{s}_1, \bar{t}_1, \bar{s}_2, \bar{t}_2, R_1, R_2)$*, we can construct, in time* $O(|E^r|)$*, a* 2CF *instance* $(G^{2cf}, \boldsymbol{u}^{2cf}, \bar{\bar{s}}_1, \bar{t}_1, \bar{\bar{s}}_2, \bar{t}_2, R^{2cf})$ *such that*

$$\left|E^{2cf}\right| = O\left(|E^r|\right), \quad \left\|\boldsymbol{u}^{2cf}\right\|_{\max} = \max\{R_1, R_2\}, \quad R^{2cf} = R_1 + R_2.$$

If the 2CFR *instance has a solution, then the* 2CF *instance has a solution.*

Proof. The runtime for reduction and solution mapping is trivial, as we only add two vertices and two edges. And by construction, $\left\|\boldsymbol{u}^{2cf}\right\|_{\max} = \max\{R_1, R_2\}$.

If there exists a solution $\boldsymbol{f}^r$ to the 2CFR instance such that $F_1^r \geq R_1, F_2^r \geq R_2$, then there also exists a solution $\boldsymbol{f}^{2cf}$ to the 2CF instance because $F_1^{2cf} = R_1, F_2^{2cf} = R_2$, and thus $F_1^{2cf} + F_2^{2cf} = R^{2cf}$. □

Lemma 3.6.19 (2CFRA to 2CFA)**.** *Given a* 2CFRA *instance* $(G^r, \boldsymbol{u}^r, \bar{s}_1, \bar{t}_1, \bar{s}_2, \bar{t}_2, R_1, R_2, \epsilon^r)$*, we can reduce it to a* 2CFA *instance* $(G^{2cf}, \boldsymbol{u}^{2cf}, \bar{\bar{s}}_1, \bar{t}_1, \bar{\bar{s}}_2, \bar{t}_2, R^{2cf}, \epsilon^{2cf})$ *by letting*

$$\epsilon^{2cf} = \Omega(\epsilon^r),$$

and using Lemma 3.6.18 to construct a 2CF *instance* $(G^{2cf}, \boldsymbol{u}^{2cf}, \bar{\bar{s}}_1, \bar{t}_1, \bar{\bar{s}}_2, \bar{t}_2, R^{2cf})$ *from the* 2CFR *instance* $(G^r, \boldsymbol{u}^r, \bar{s}_1, \bar{t}_1, \bar{s}_2, \bar{t}_2, R_1, R_2)$*. If* $\boldsymbol{f}^{2cf}$ *is a solution to the* 2CFA *(*2CF*) instance, then in time* $O(|E^r|)$*, we can compute a solution* $\boldsymbol{f}^r$ *to the* 2CFRA *(*2CFR*, respectively) instance, where the exact case holds when* $\epsilon^{2cf} = \epsilon^r = 0$*.*

Proof. Error in congestion does not increase since flows and edge capacities are unchanged for those edges other than $(\bar{\bar{s}}_1, \bar{s}_1), (\bar{\bar{s}}_2, \bar{s}_2)$. Hence, $\epsilon^r \leq O(\epsilon^{2cf})$.

Then for error in demand, notice that only the incoming flow of vertex $\bar{s}_1, \bar{s}_2$ changes by solution mapping. Therefore, error in demand does not increase for vertices other than $\bar{s}_1, \bar{s}_2$, thus we only need to bound the error in demand of $\bar{s}_1, \bar{s}_2$. We consider commodity $i, i \in \{1, 2\}$. We first bound the error in demand of vertex $\bar{s}_i$.

$$\begin{aligned}\left|\sum_{w:(\bar{s}_i, w)\in E} \boldsymbol{f}_i^{2cf}(\bar{s}_i, w) - R_i\right| &\leq \left|\sum_{w:(\bar{s}_i, w)\in E} \boldsymbol{f}_i^{2cf}(\bar{s}_i, w) - \boldsymbol{f}_i^{2cf}(\bar{\bar{s}}_i, \bar{s}_i)\right| + \left|\boldsymbol{f}_i^{2cf}(\bar{\bar{s}}_i, \bar{s}_i) - R_i\right| \\ &\leq \epsilon^{2cf} + \left|\boldsymbol{f}_i^{2cf}(\bar{\bar{s}}_i, \bar{s}_i) - R_i\right|, \quad \text{by error in demand of vertex } \bar{s}_i\end{aligned}$$

For $\left|\boldsymbol{f}_i^{2cf}(\bar{\bar{s}}_i, \bar{s}_i) - R_i\right|$, by error in congestion, we have

$$\boldsymbol{f}_i^{2cf}(\bar{\bar{s}}_i, \bar{s}_i) - R_i \leq \epsilon^{2cf}.$$

In the other direction, we have

$$\begin{aligned}\boldsymbol{f}_i^{2cf}(\bar{\bar{s}}_i, \bar{s}_i) - R_i &\geq F_i^{2cf} - \epsilon^{2cf} - R_i && \text{by error in demand of vertex } \bar{\bar{s}}_i \\ &= R^{2cf} - F_{\bar{i}}^{2cf} - R_i - \epsilon^{2cf} && \text{by construction} \\ &= R_{\bar{i}} - F_{\bar{i}}^{2cf} - \epsilon^{2cf} && \text{by } R_1 + R_2 = R^{2cf} \\ &\geq \boldsymbol{f}_{\bar{i}}^{2cf}(\bar{\bar{s}}_i, \bar{s}_i) - F_{\bar{i}}^{2cf} - 2\epsilon^{2cf} && \text{by error in congestion of edge } (\bar{\bar{s}}_i, \bar{s}_i) \\ &\geq -3\epsilon^{2cf}, && \text{by error in demand of vertex } \bar{\bar{s}}_{\bar{i}}\end{aligned}$$

Therefore, we have $\left|\boldsymbol{f}_i^{2cf}(\bar{\bar{s}}_i, \bar{s}_i) - R_i\right| \leq 3\epsilon^{2cf}$, and thus

$$\left|\sum_{w:(\bar{s}_i,w)\in E} \boldsymbol{f}_i^{2cf}(\bar{s}_i, w) - R_i\right| \leq 4\epsilon^{2cf}.$$

To bound the error in demand of vertex $\bar{s}_{\bar{i}}$, we have $\sum_{w:(\bar{\bar{s}}_{\bar{i}},w)\in E} \boldsymbol{f}_i^r(\bar{s}_{\bar{i}}, w) \leq 2\epsilon^{2cf}$ because error in demand is accumulated twice over two vertices $\bar{\bar{s}}_{\bar{i}}, \bar{s}_{\bar{i}}$.

To conclude,

$$\epsilon^r \leq O(\epsilon^{2cf}).$$

□

3.7 A Unified Framework for LP Transformations

In Section 3.5 and 3.6, we described a series of reduction steps in both the linear algebra space and the flow space. While we have identified certain reduction patterns in Section 3.5.1 and 3.6.1, each step uses its own gadget and error propagation arguments. We aim to provide a unified characterization of problem reduction and how the reductions can be analyzed systematically.

Note that every intermediate problem we consider (both in the algebra space and in the flow space) can be represented as a feasibility LP of the form:

$$\{\boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\},$$

where $\boldsymbol{A}$ and $\boldsymbol{b}$ define linear inequalities enforcing the constraints of that problem. Note that a linear equality constraint $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is equivalent to the pair of inequalities $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ and $-\boldsymbol{A}\boldsymbol{x} \leq -\boldsymbol{b}$. Similarly, non-negativity constraints $\boldsymbol{x} \geq \boldsymbol{0}$ can be expressed as $-\boldsymbol{I}\boldsymbol{x} \leq \boldsymbol{0}$.

This section introduces a framework that enables us to view each reduction step as an LP-to-LP transformation characterized by linear mappings on both variables and constraints. This viewpoint simplifies the correctness and error-propagation analyses, showing that each step follows the same underlying principles.

3.7.1 Characterizing LP Transformations

Suppose we have an original problem A in LP form:

$$\mathcal{F}^A = \{\boldsymbol{x}^A \in \mathbb{R}^{n^A} : \boldsymbol{A}^A\boldsymbol{x}^A \leq \boldsymbol{b}^A\}, \text{ where } \boldsymbol{A}^A \in \mathbb{R}^{m^A \times n^A}, \boldsymbol{b}^A \in \mathbb{R}^{m^A}.$$

We want to reduce it to a new problem B, also representable as an LP:

$$\mathcal{F}^B = \{\boldsymbol{x}^B \in \mathbb{R}^{n^B} : \boldsymbol{A}^B\boldsymbol{x}^B \leq \boldsymbol{b}^B\}, \text{where } \boldsymbol{A}^B \in \mathbb{R}^{m^B \times n^B}, \boldsymbol{b}^B \in \mathbb{R}^{m^B}.$$

By construction, typically $n^A \leq n^B$ and $m^A \leq m^B$, as we add auxiliary variables and constraints in the new problem.

LP transformation can be characterized with two linear operators: *variable operator* $\boldsymbol{T}$ and *constraint operator* $\boldsymbol{P}$.

Variable Operator $\boldsymbol{T}$. Each reduction step embeds the original variable vector $\boldsymbol{x}^A$ into the larger variable vector $\boldsymbol{x}^B$. Concretely, we define $\boldsymbol{T} \in \mathbb{R}^{n^A \times n^B}$ such that

$$\boldsymbol{x}^A = \boldsymbol{T}\boldsymbol{x}^B. \tag{3.8}$$

Typically, by design, each row of $\boldsymbol{T}$ typically has a single 1 and the rest zeros, so that each $\boldsymbol{x}^A(j)$ is copied from the appropriate position in $\boldsymbol{x}^B$. Equivalently, we can view $\boldsymbol{x}^B$ as

$$\boldsymbol{x}^B = \begin{bmatrix} \boldsymbol{x}^A \\ \boldsymbol{x}^{\text{aux}} \end{bmatrix}.$$

Hence, applying $\boldsymbol{T}$ just discards the auxiliary variables when extracting $\boldsymbol{x}^A$ from $\boldsymbol{x}^B$.

Constraint Operator $\boldsymbol{P}$. We regard the original system $\boldsymbol{A}^A\boldsymbol{x}^A \leq \boldsymbol{b}^A$ as being derived from $\boldsymbol{A}^B$ (and $\boldsymbol{b}^B$) by applying a linear operator $\boldsymbol{P} \in \mathbb{R}_{\geq 0}^{m^A \times m^B}$. Specifically, we require

$$\boldsymbol{A}^A\boldsymbol{T} = \boldsymbol{P}\boldsymbol{A}^B, \quad \boldsymbol{b}^A = \boldsymbol{P}\boldsymbol{b}^B. \tag{3.9}$$

That is, each constraint of problem A is obtained by taking a *non-negative* linear combination of constraints in problem B. The non-negativity of $\boldsymbol{P}$ is essential when working with inequalities, as it guarantees that feasibility is preserved: if $\boldsymbol{A}^B\boldsymbol{x}^B \leq \boldsymbol{b}^B$, then $\boldsymbol{P}(\boldsymbol{A}^B\boldsymbol{x}^B) \leq \boldsymbol{P}\boldsymbol{b}^B$, ensuring that $\boldsymbol{A}^A\boldsymbol{x}^A \leq \boldsymbol{b}^A$.

We remark that for transformation among equality constraints, the sign of $\boldsymbol{P}$ is irrelevant. In this case, if $\boldsymbol{A}^B\boldsymbol{x}^B = \boldsymbol{b}^B$, then any linear operator $\boldsymbol{P}$ can preserve feasibility $\boldsymbol{P}(\boldsymbol{A}^B\boldsymbol{x}^B) = \boldsymbol{P}\boldsymbol{b}^B$.

3.7.2 Unified Correctness Analysis

Recall that the correctness analysis of a reduction algorithm involves verifying that problem A has a feasible solution if and only if the reduced problem B also has a feasible solution. In other words, there are two main points to check:

1. Forward feasibility: From any feasible $\boldsymbol{x}^A$ for problem A, we can construct a feasible $\boldsymbol{x}^B$ for problem B. Usually, this forward-feasibility construction is direct or "gadget-based"[8].

2. Backward feasibility: Given a feasible $\boldsymbol{x}^B$ for problem B, we must show $\boldsymbol{x}^A = \boldsymbol{T}\boldsymbol{x}^B$ is feasible for problem $\boldsymbol{A}$.

Lemma 3.7.1 (Correctness of Reduction Algorithm)**.** *Let $\boldsymbol{T}$ and $\boldsymbol{P}$ satisfy the relations in Equation (3.8) and (3.9). Additionally, assume there is a forward-feasibility mapping $\boldsymbol{L} : \mathbb{R}^{n^A} \to \mathbb{R}^{n^B}$ such that $\boldsymbol{L}(\mathcal{F}^A) \subseteq \mathcal{F}^B$. Then problem A has a feasible solution if and only if the reduced problem B also has a feasible solution.*

Proof. If $\boldsymbol{x}^A$ is a feasible solution to A, by $\boldsymbol{L}(\mathcal{F}^A) \subseteq \mathcal{F}^B$, then a feasible solution to B can be constructed by applying $\boldsymbol{L}$ to $\boldsymbol{x}^A$ as $\boldsymbol{x}^B = \boldsymbol{L}(\boldsymbol{x}^A)$.

[8]In each step of Sections 3.5 and 3.6, the solution construction approach is detailed in the proofs of the lemmas for the exact cases.

It remains to show that if $\boldsymbol{x}^B$ is a feasible solution to B, then $\boldsymbol{T}\boldsymbol{x}^B$ is a feasible solution to A, i.e., $\boldsymbol{T}(\mathcal{F}^B) \subseteq \mathcal{F}^A$. For any feasible solution $\boldsymbol{x}^B \in \mathcal{F}^B$, we have

$$\begin{aligned}
\boldsymbol{A}^B\boldsymbol{x}^B &\leq \boldsymbol{b}^B \\
&\Downarrow \\
\boldsymbol{P}\boldsymbol{A}^B\boldsymbol{x}^B &\leq \boldsymbol{P}\boldsymbol{b}^B && \text{by non-negativity of } \boldsymbol{P} \\
&\Downarrow \\
\boldsymbol{A}^A\boldsymbol{T}\boldsymbol{x}^B &\leq \boldsymbol{b}^A && \text{by Eq. (3.9)} \\
&\Downarrow \\
\boldsymbol{A}^A\boldsymbol{x}^A &\leq \boldsymbol{b}^A && \text{by Eq. (3.8)}
\end{aligned}$$

□

3.7.3 Unified Error Analysis

We extend the above analysis for feasible solutions to solutions with additive approximations. We say $\boldsymbol{x}$ is an ϵ-approximate solution if[9]

$$\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \leq \epsilon\mathbf{1}.$$

Given $\boldsymbol{x}^B$ an ϵ^B-approximate solution to problem B, the goal of the error analysis is to understand how the approximation error of $\boldsymbol{x}^A = \boldsymbol{T}\boldsymbol{x}^B$ depends on ϵ^B.

Lemma 3.7.2 (Error Analysis). *Let $\boldsymbol{x}^B$ be an ϵ^B-approximate solution to problem B, then $\boldsymbol{x}^A = \boldsymbol{T}\boldsymbol{x}^B$ is an ϵ^A-approximate solution to problem A with*

$$\epsilon^A \leq \|\boldsymbol{P}\|_\infty \epsilon^B.$$

Proof. We have

$$\begin{aligned}
\boldsymbol{A}^B\boldsymbol{x}^B - \boldsymbol{b}^B &\leq \epsilon^B\mathbf{1}_{n^B} \\
&\Downarrow \\
\boldsymbol{P}(\boldsymbol{A}^B\boldsymbol{x}^B - \boldsymbol{b}^B) &\leq \epsilon^B\boldsymbol{P}\mathbf{1}_{n^B} && \text{by non-negativity of } \boldsymbol{P} \\
&\Downarrow \\
\boldsymbol{A}^A\boldsymbol{T}\boldsymbol{x}^B - \boldsymbol{b}^A &\leq \epsilon^B\|\boldsymbol{P}\|_\infty\mathbf{1}_{n^A} && \text{by Eq. (3.9)} \\
&\Downarrow \\
\boldsymbol{A}^A\boldsymbol{x}^A - \boldsymbol{b}^A &\leq \epsilon^B\|\boldsymbol{P}\|_\infty\mathbf{1}_{n^A} && \text{by Eq. (3.8)}
\end{aligned}$$

□

Therefore, bounding $\|\boldsymbol{P}\|_\infty$ by a modest polynomial factor at each step ensures the overall additive error does not grow too large throughout the entire chain of reductions. Lemma 3.7.2 unifies the more specialized error analyses found in each step of Sections 3.5 and 3.6.

[9]If we assume the non-negativity constraints are always satisfied, which is the case in our proofs in Section 3.5 and 3.6, the right-hand side vector can be further tightened.

3.7.4 Examples

As an illustration, we revisit several steps in the reduction chain and analyze them using this unified framework.

LP(A) to LEN(A)

We revisit the step that transform an LP instance $\{\boldsymbol{c}^\top \boldsymbol{x} \geq K, \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$ to an LEN instance $\{\tilde{\boldsymbol{A}}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}, \tilde{\boldsymbol{x}} \geq \boldsymbol{0}\}$. To be compatible with the form used in the framework, we rewrite the constraints in LP instance (problem A) as

$$\begin{bmatrix} -\boldsymbol{c}^\top \\ \boldsymbol{A} \\ -\boldsymbol{I} \end{bmatrix} \boldsymbol{x} \leq \begin{bmatrix} -K \\ \boldsymbol{b} \\ \boldsymbol{0} \end{bmatrix} \quad \Rightarrow \quad \boldsymbol{A}^A = \begin{bmatrix} -\boldsymbol{c}^\top \\ \boldsymbol{A} \\ -\boldsymbol{I} \end{bmatrix}, \quad \boldsymbol{b}^A = \begin{bmatrix} -K \\ \boldsymbol{b} \\ \boldsymbol{0} \end{bmatrix}.$$

The LEN instance (problem B) is defined by appending slack variables $\boldsymbol{s} \geq \boldsymbol{0}$ and $\alpha \geq 0$. We have

$$\tilde{\boldsymbol{A}} = \begin{bmatrix} -\boldsymbol{c}^\top & \boldsymbol{0} & 1 \\ \boldsymbol{A} & \boldsymbol{I} & \boldsymbol{0} \end{bmatrix}, \quad \tilde{\boldsymbol{b}} = \begin{bmatrix} -K \\ \boldsymbol{b} \end{bmatrix}.$$

Constructing $\boldsymbol{A}^B, \boldsymbol{b}^B$ from $\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{b}}$ is straightforward:

$$\boldsymbol{A}^B = \begin{bmatrix} \tilde{\boldsymbol{A}} \\ -\tilde{\boldsymbol{A}} \\ -\boldsymbol{I} \end{bmatrix}, \quad \boldsymbol{b}^B = \begin{bmatrix} \tilde{\boldsymbol{b}} \\ -\tilde{\boldsymbol{b}} \\ \boldsymbol{0} \end{bmatrix}.$$

The operator $\boldsymbol{P}$ performs row additions on $\boldsymbol{A}^B$ to match rows in $\boldsymbol{A}^A\boldsymbol{T}$. Concretely, each row of $\boldsymbol{A}^A\boldsymbol{T}$ is constructed by adding the corresponding row in $\boldsymbol{A}^B$ to a specific row of $-\boldsymbol{I}$, ensuring that any extra non-zero entry in the auxiliary variables is canceled out.

The forward-feasibility mapping $\boldsymbol{L}$ is described in the proof of Lemma 3.5.1. Thus, by Lemma 3.7.1, the correctness of the reduction can be proven. Moreover, since $\|\boldsymbol{P}\|_\infty$ is a constant here, the approximate error ϵ^A grows only by a constant factor from ϵ^B, consistent with Lemma 3.5.2.

LEN(A) to 2-LEN(A)

We use a concrete example to illustrate the construction of the constraint operator $\boldsymbol{P}$ for this step. Starting with a linear constraint $5\boldsymbol{x}_1 + 3\boldsymbol{x}_2 - 7\boldsymbol{x}_3 = -1$. In matrix form, this can be rewritten as

$$\begin{bmatrix} 5 & 3 & -7 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \end{bmatrix} = \begin{bmatrix} -1 \end{bmatrix}.$$

The reduction algorithm of bitwise decomposition (see Eq. (3.2)) leads to the following reduced problem:

$$\left[\begin{array}{ccc|cccc|c} 1 & 1 & -1 & -2 & 2 & 0 & 0 & \\ 0 & 1 & -1 & 1 & -1 & -2 & 2 & \\ 1 & 0 & -1 & 0 & 0 & 1 & -1 & \\ & & & \multicolumn{4}{c|}{\boldsymbol{I}_4} & \boldsymbol{I}_4 \end{array}\right] \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \\ \boldsymbol{c}_0 \\ \boldsymbol{d}_0 \\ \boldsymbol{c}_1 \\ \boldsymbol{d}_1 \\ \boldsymbol{s} \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 0 \\ \bar{X}\boldsymbol{1}_4 \end{bmatrix}.$$

The first three rows represent the bitwise decomposition constraints, corresponding to bit weights $2^0, 2^1$ and 2^2, respectively; and the last block row with identity matrices $\boldsymbol{I}$ corresponds to slack variables $\boldsymbol{s}$ added to carry terms $\boldsymbol{c}, \boldsymbol{d}$.

Notice that

$$\begin{bmatrix} 2^0 & 2^1 & 2^2 & \mathbf{0}_4^\top \end{bmatrix} \left[\begin{array}{ccc|cccc|c} 1 & 1 & -1 & -2 & 2 & 0 & 0 & \\ 0 & 1 & -1 & 1 & -1 & -2 & 2 & \\ 1 & 0 & -1 & 0 & 0 & 1 & -1 & \\ & & & & \boldsymbol{I}_4 & & & \boldsymbol{I}_4 \end{array}\right] = \begin{bmatrix} 5 & 3 & -7 & \mathbf{0}_4^\top & \mathbf{0}_4^\top \end{bmatrix},$$

and

$$\begin{bmatrix} 2^0 & 2^1 & 2^2 & \mathbf{0}_4^\top \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ 0 \\ \bar{X}\mathbf{1}_4 \end{bmatrix} = \begin{bmatrix} -1 \end{bmatrix}.$$

They correspond to $\boldsymbol{P}\boldsymbol{A}^B = \boldsymbol{A}^A\boldsymbol{T}$ and $\boldsymbol{P}\boldsymbol{b}^B = \boldsymbol{b}^A$, respectively. Hence, we can construct the constraint operator $\boldsymbol{P}$ (up to column rearrangements) as

$$\boldsymbol{P} = \begin{bmatrix} 2^0 & 2^1 & 2^2 & \mathbf{0}_4^\top & \\ & & & & \boldsymbol{I}_3 \end{bmatrix},$$

where the first row corresponds to the reverse of the bitwise decomposition process, summing up contributions weighted by 2^i; the last block $\boldsymbol{I}_3$ corresponds to non-negativity constraints for $\boldsymbol{x}$.

We have

$$\|\boldsymbol{P}\|_\infty = O(\bar{X}),$$

where $\bar{X}$ is the magnitude of LEN. Again, it matches the bound stated in Lemma 3.5.4.

2-LEN(A) to 1-LEN(A)

The reduction from a 2-LEN instance $(\bar{\boldsymbol{A}}, \bar{\boldsymbol{b}}, \bar{R})$ to a 1-LEN instance $(\hat{\boldsymbol{A}}, \hat{\boldsymbol{b}}, \hat{R})$ is relatively straightforward. We illustrate with a concrete example from the previous step: $\boldsymbol{x}_1 + \boldsymbol{x}_2 - \boldsymbol{x}_3 - 2(\boldsymbol{c}_0 - \boldsymbol{d}_0) = -1$. In matrix form,

$$\begin{bmatrix} 1 & 1 & -1 & -2 & -2 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \\ \boldsymbol{c}_0 \\ \boldsymbol{d}_0 \end{bmatrix} = \begin{bmatrix} -1 \end{bmatrix}.$$

Introducing auxiliary variables $\boldsymbol{c}_0'$ and $\boldsymbol{d}_0'$, we reduce it to

$$\begin{aligned} \boldsymbol{x}_1 + \boldsymbol{x}_2 - \boldsymbol{x}_3 - \boldsymbol{c}_0 - \boldsymbol{c}_0' + \boldsymbol{d}_0 + \boldsymbol{d}_0' &= -1 \\ \boldsymbol{c}_0 - \boldsymbol{c}_0' &= 0 \\ \boldsymbol{d}_0 - \boldsymbol{d}_0' &= 0, \end{aligned}$$

or in matrix form,

$$\begin{bmatrix} 1 & 1 & -1 & -1 & -1 & 1 & 1 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \\ \boldsymbol{c}_0 \\ \boldsymbol{c}'_0 \\ \boldsymbol{d}_0 \\ \boldsymbol{d}'_0 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}.$$

Notice that a linear combination of the reduced problem recovers the original problem. Specifically,

$$\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & -1 & -1 & -1 & 1 & 1 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & -1 & -2 & -2 & 0 & 0 \end{bmatrix}.$$

The constraint operator $\boldsymbol{P}$ can thus be constructed accordingly. In the general case, each coefficient of ± 2 in the 2-LEN instance leads to the introduction of two auxiliary variables and two additional equations in the 1-LEN instance. Consequently, each row of $\boldsymbol{P}$ includes a number of non-zero entries proportional to the number of variables with coefficients ± 2. Since the number of such variables is trivially bounded by the number of non-zeros $\bar{N}$ in 2-LEN, we obtain a matching bound as Lemma 3.5.6:

$$\|\boldsymbol{P}\|_\infty \leq O(\bar{N}).$$

1-LEN(A) to FHF(A)

Recall the form of 1-LEN $\{\hat{\boldsymbol{A}}\hat{\boldsymbol{x}} = \hat{\boldsymbol{b}}, \hat{\boldsymbol{x}} \geq \boldsymbol{0}\}$, where $\hat{\boldsymbol{A}} \in \mathbb{R}^{\hat{m} \times \hat{n}}$ and entries of $\hat{\boldsymbol{A}}$ are in $\{0, \pm 1\}$. For each linear equation in 1-LEN, saying

$$\sum_{j \in J^+} \hat{\boldsymbol{x}}(j) - \sum_{j \in J^-} \hat{\boldsymbol{x}}(j) = b,$$

or in matrix form,

$$\begin{bmatrix} \boldsymbol{1}_{|J^+|}^\top & -\boldsymbol{1}_{|J^-|}^\top \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{x}}(J^+) \\ \hat{\boldsymbol{x}}(J^-) \end{bmatrix} = \begin{bmatrix} b \end{bmatrix}, \quad \boldsymbol{x} \geq \boldsymbol{0}.$$

It is encoded by several flow constraints as follows:

$$\begin{aligned} \sum_{j \in J^+} \hat{\boldsymbol{x}}(j) &= \boldsymbol{f}_1 + \boldsymbol{f}_2, && \text{(flow conservation)} \\ \sum_{j \in J^-} \hat{\boldsymbol{x}}(j) &= \boldsymbol{f}_3, && \text{(flow conservation)} \\ \boldsymbol{f}_1 &= b, && \text{(fixed flow)} \\ \boldsymbol{f}_2 - \boldsymbol{f}_3 &= 0, && \text{(homologous flow)} \\ \begin{bmatrix} \hat{\boldsymbol{x}} \\ \boldsymbol{f} \end{bmatrix} &\leq \boldsymbol{u}. && \text{(capacity)} \end{aligned}$$

In matrix form, the flow constraints are

$$\begin{bmatrix} \mathbf{1}_{|J^+|}^\top & & -1 & -1 & 0 \\ & -\mathbf{1}_{|J^-|}^\top & 0 & 0 & 1 \\ & & 1 & 0 & 0 \\ & & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{x}}(J^+) \\ \hat{\boldsymbol{x}}(J^-) \\ \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \\ \boldsymbol{f}_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ b \\ 0 \end{bmatrix}, \quad \mathbf{0} \leq \begin{bmatrix} \hat{\boldsymbol{x}} \\ \boldsymbol{f} \end{bmatrix} \leq \boldsymbol{u}.$$

Observe that by taking a linear combination of the flow constraints, we can recover the original 1-LEN equation. Specifically,

$$\begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{1}_{|J^+|}^\top & & -1 & -1 & 0 \\ & -\mathbf{1}_{|J^-|}^\top & 0 & 0 & 1 \\ & & 1 & 0 & 0 \\ & & 0 & 1 & -1 \end{bmatrix} = \begin{bmatrix} \mathbf{1}_{|J^+|}^\top & -\mathbf{1}_{|J^-|}^\top & 0 & 0 & 0 \end{bmatrix}.$$

For a single equation, we can derive the above constraint operator $\boldsymbol{P}$ and $\|\boldsymbol{P}\|_\infty = \Theta(1)$. When handling multiple equations, more homologous flow constraints are introduced—each linking multiple flows corresponding to the same variable across different sections of the network. This increases the complexity of the constraint operator $\boldsymbol{P}$. However, the number of non-zero entries in each row of $\boldsymbol{P}$ remains bounded by $O(\hat{N})$, where $\hat{N}$ is the number of non-zeros in the matrix $\hat{\boldsymbol{A}}$. Thus, in the general case,

$$\|\boldsymbol{P}\|_\infty \leq O(\hat{N}),$$

which matches the bound established in Lemma 3.5.9.

Dropping Homologous Flow Constraints

We show that every homologous flow constraint can be encoded as a linear combination of a constant number of flow conservation constraints, selective flow constraints, and fixed flow constraints.

Refer to Figure 3.5. Our target is to encode $\boldsymbol{f}^p(e) = \boldsymbol{f}^p(\hat{e})$, or before solution mapping, $\boldsymbol{f}_1^s(e_1) = \boldsymbol{f}_1^s(\hat{e}_1)$.

We start by flow conservation of vertex vw, vw', yz, yz':

$$\begin{aligned} \boldsymbol{f}_1^s(e_1) + \boldsymbol{f}_1^s(e_3) = \boldsymbol{f}_1^s(e_4) = \boldsymbol{f}_1^s(e_2) + \boldsymbol{f}_1^s(e_5), \\ \boldsymbol{f}_2^s(e_1) + \boldsymbol{f}_2^s(e_3) = \boldsymbol{f}_2^s(e_4) = \boldsymbol{f}_2^s(e_2) + \boldsymbol{f}_2^s(e_5), \\ \boldsymbol{f}_1^s(\hat{e}_1) + \boldsymbol{f}_1^s(e_5) = \boldsymbol{f}_1^s(\hat{e}_4) = \boldsymbol{f}_1^s(\hat{e}_2) + \boldsymbol{f}_1^s(e_5), \\ \boldsymbol{f}_2^s(\hat{e}_1) + \boldsymbol{f}_2^s(e_5) = \boldsymbol{f}_2^s(\hat{e}_4) = \boldsymbol{f}_2^s(\hat{e}_2) + \boldsymbol{f}_2^s(e_5). \end{aligned} \qquad \text{(flow conservation)}$$

Then plugging in the selective constraints on edge $e_1, e_2, \hat{e}_1, \hat{e}_2$ (i.e., $\boldsymbol{f}_2^s(e_1) = \boldsymbol{f}_2^s(e_2) = \boldsymbol{f}_2^s(\hat{e}_1) = \boldsymbol{f}_2^s(\hat{e}_2) = 0$) and on edge $e_3, e_5, \hat{e}_5$ (i.e., $\boldsymbol{f}_1^s(e_3) = \boldsymbol{f}_1^s(e_5) = \boldsymbol{f}_1^s(\hat{e}_5) = 0$), we have

$$\begin{aligned} \boldsymbol{f}_1^s(e_1) =& \boldsymbol{f}_1^s(e_4) = \boldsymbol{f}_1^s(e_2), \\ \boldsymbol{f}_1^s(\hat{e}_1) =& \boldsymbol{f}_1^s(\hat{e}_4) = \boldsymbol{f}_1^s(\hat{e}_2), \\ \boldsymbol{f}_2^s(e_3) = \boldsymbol{f}_2^s(e_4) =& \boldsymbol{f}_2^s(e_5) = \boldsymbol{f}_2^s(\hat{e}_4) = \boldsymbol{f}_2^s(\hat{e}_5). \end{aligned} \qquad \text{(flow conservation + selective)}$$

Finally, the fixed flow constraints on edge e_4 and $\hat{e}_4$ enforce $\boldsymbol{f}_1^s(e_4) + \boldsymbol{f}_2^s(e_4) = u$ and $\boldsymbol{f}_1^s(\hat{e}_4) + \boldsymbol{f}_2^s(\hat{e}_4) = u$. Combining with the above immediately yields

$$\boldsymbol{f}_1^s(e_1) = \boldsymbol{f}_1^s(\hat{e}_1), \qquad \text{(flow conservation + selective + fixed)}$$

which encodes the homologous flow constraint. Crucially, each original homologous-flow constraint is encoded by $O(1)$ constraints in the reduced problem, so the corresponding row sum of $\boldsymbol{P}$ is bounded by a constant.

Dropping Selective Flow Constraints

Consider a selective edge e for commodity 1 (Figure 3.6). The target is to encode $\boldsymbol{f}_2^s(e) = 0$, or before solution mapping, $\boldsymbol{f}_2^f(e_1) = 0$.

By flow conservation of vertex xy and t_1, because t_1 has no outgoing edges for commodity 2, we have

$$\boldsymbol{f}_2^f(e_1) + \boldsymbol{f}_2^f(e_3) = \boldsymbol{f}_2^f(e_4) = 0. \qquad \text{(flow conservation)}$$

Since every flow is non-negative, it implies

$$\boldsymbol{f}_2^f(e_1) = 0, \qquad \text{(flow conservation + non-negativity)}$$

which encodes the selective flow constraint.

Dropping Fixed Flow Constraints

Figure 3.7 shows the gadget that replaces an edge with the fixed capacity constraint

$$\boldsymbol{f}_1^f(e) + \boldsymbol{f}_2^f(e) = u.$$

Before the solution mapping, this means enforcing

$$\boldsymbol{f}_1^r(e_1) + \boldsymbol{f}_2^r(e_1) = u.$$

By flow conservation of vertex xy and no outgoing edges for commodity 2 and 1 for $\bar{t}_1$ and $\bar{t}_2$, respectively, we have

$$\begin{aligned} \boldsymbol{f}_1^r(e_1) + \boldsymbol{f}_1^r(e_3) &= \boldsymbol{f}_1^r(e_4), \\ \boldsymbol{f}_2^r(e_1) + \boldsymbol{f}_2^r(e_3) &= \boldsymbol{f}_2^r(e_5). \end{aligned} \qquad \text{(flow conservation)}$$

Moreover, because of the required amount of flow, all incoming edges of $\bar{t}_1, \bar{t}_2$ saturate. That is, $\bar{t}_i$ receive exactly u units of commodity i:

$$\begin{aligned} \boldsymbol{f}_1^r(e_1) + \boldsymbol{f}_1^r(e_3) &= \boldsymbol{f}_1^r(e_4) = u, \\ \boldsymbol{f}_2^r(e_1) + \boldsymbol{f}_2^r(e_3) &= \boldsymbol{f}_2^r(e_5) = u. \end{aligned} \qquad \text{(flow conservation + requirement + capacity)}$$

Then combined with capacity constraint of edge e_1 and e_3 (i.e., $\boldsymbol{f}_1^r(e_1) + \boldsymbol{f}_2^r(e_1) \leq u$, $\boldsymbol{f}_1^r(e_3) + \boldsymbol{f}_2^r(e_3) \leq u$), this gives

$$\boldsymbol{f}_1^r(e_1) + \boldsymbol{f}_2^r(e_1) = u, \qquad \text{(flow conservation + requirement + capacity)}$$

which encodes the fixed flow constraint.

The examples above demonstrate that every special constraint can be encoded as a linear combination of a constant number of other constraints. We omit the detailed encoding of other ordinary constraints associated with the reduced edges, such as flow conservation and capacity constraints. Due to the constant size of the gadgets, the transformed graph G^B contains $O(\lvert E^A \rvert)$ edges, thus $O(\lvert E^A \rvert)$ constraints. Therefore, every original constraint can be recovered as the sum of at most $O(\lvert E^A \rvert)$ constraints in G^B, i.e.,

$$\lVert \boldsymbol{P} \rVert_\infty \le O(\lvert E^A \rvert),$$

yielding the matching error bounds achieved in these reduction steps:

$$\epsilon^A \le O(\lvert E^A \rvert)\epsilon^B.$$

3.7.5 Summary

Under this framework, we can treat each reduction step as a composition of two linear mappings, one on the variables ($\boldsymbol{T}$) and one on the constraints ($\boldsymbol{P}$), plus the forward-feasibility mapping $\boldsymbol{L}$. Once we construct ($\boldsymbol{T}, \boldsymbol{P}, \boldsymbol{L}$) for a reduction step, its correctness follows immediately from Lemma 3.7.1.

In addition, this framework provides a systematic approach to analyzing error propagation. Rather than redoing an error analysis from scratch for each reduction gadget, it suffices to bound the ℓ_∞ norm of the constraint operator $\lVert \boldsymbol{P} \rVert_\infty$ at each step. This immediately yields how ϵ^B translates into ϵ^A.

Intuitively, the role of $\lVert \boldsymbol{P} \rVert_\infty$ in bounding error growth corresponds to how we break down an error term into several intermediate terms in our actual proofs:

$$\left\lvert \boldsymbol{A}^A(i)\boldsymbol{x}^A - \boldsymbol{b}^A(i) \right\rvert \le \sum_{j \in J} \left\lvert \boldsymbol{A}^B(j)\boldsymbol{x}^B - \boldsymbol{b}^B(j) \right\rvert,$$

where J is the set of indices of constraints in B that are related to the ith constraint in A. Combining these terms is equivalent to forming a linear combination of constraints in problem B—an operation precisely captured by the constraint operator $\boldsymbol{P}$.

Although constructing $\boldsymbol{P}$ still requires step-specific analysis for individual reduction gadgets, the framework serves as a powerful tool both for validating existing reductions and for guiding the design and verification of new reduction steps within the chain of LP reductions.

Chapter 4

Hardness Results for Combinatorial Laplacians

This chapter is based on [Din+22]. We focus on proving Theorem 1.4.4. Specifically, we describe the reduction algorithm and analyze it for approximate solvers in Section 4.5. Then in Section 4.6, we slightly modify the reduction for approximate solvers to handle the general case, when the right-hand side vector may not be in the image of the coefficient matrix. Theorem 1.4.2 can be derived from Theorem 1.4.4 and standard arguments in linear algebra, with the proof provided in Appendix B.1.

4.1 Prior Works

Kyng and Zhang [KZ17] initiated the study of hardness results for solving structured linear equations. They showed that 2-commodity Laplacians, 2-dimensional truss stiffness matrices, and 2-total-variation matrices are all sparse-linear-equation complete. [KWZ20] considered a larger family of hardness assumptions based on linear equations. Through defining a parameterized family of hypotheses for runtime of solving sparse linear equations, they prove the 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 $\tilde{O}(\text{number of non-zeros} \times \epsilon^{-0.165})$, then one can solve a system of linear equations in time asymptotically faster than $\tilde{O}(\text{number of non-zeros} \times \text{condition number of matrix})$, which is the runtime of conjugate gradient.

4.2 Our Contributions

We prove that solving linear equations in boundary operators and combinatorial Laplacians of 2-complexes is as hard to solve as general linear equations. As discussed in Section 1.1, there are extensive applications of combinatorial Laplacians in both pure math and applied areas. In addition, 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 [MN21]. We sketch how this inverse problem arises when using an Interior Point Method in Appendix B.2. By a similar argument as Lemma 1.4.5, 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.

Our reduction is inspired by a reduction in [MN21] that proves NP-hardness of computing maximum *integral* flows in 2-complexes via a reduction from the graph 3-coloring problem. However, the correctness of their reduction heavily relies on the fact 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 [MN21], 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.

An important aspect of our contribution is that we carefully control the number of non-zeros of the boundary operator matrix that we construct, and we bound the condition number of this matrix, and how the 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 [Bla+22] 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 RealRAM 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.

4.3 Notations and Preliminaries

4.3.1 Matrix Classes

We are interested in linear equations whose coefficient matrices belong 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 non-zero entries 1 and -1;

   (b) An *average row* which has exactly three non-zero 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 that 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 a linear equation with integer entries $\widetilde{O}(1)$ bits per entry. We caution the reader that this relies on our definition of $\widetilde{O}(\cdot)$ as hiding poly-logarithmic 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.

4.3.2 Reduction Between Linear Equations

We will again follow the definition of efficient reductions in [KZ17].

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 a system 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 non-zeros and dimensions would be stronger than what we achieve.

Fact 4.3.1. *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 4.3.2 (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 4.3.3. *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, [KZ17] implicitly shows the following result.

Theorem 4.3.4 (Implicitly Stated in [KZ17])**.** LEA *over $\mathcal{DA}$ is* SLE*-complete.*

4.4 Main Results and Reduction Outline

Our main result is stated in the following theorem.

Theorem 4.4.1. LEA *over $\mathcal{B}_2$ is* SLE*-complete.*

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* $\boldsymbol{b} \in Im(\boldsymbol{A})$ (stated in Theorem 4.4.2 and 4.4.3). 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* $\boldsymbol{b} \in Im(\boldsymbol{A})$ (stated in Theorem 4.4.4). In this case, we introduce polynomially bounded edge weights for the constructed 2-complexes.

Theorem 4.4.2. *Given a linear equation instance* LE $(\boldsymbol{A}, \boldsymbol{b})$ *where* $\boldsymbol{A} \in \mathcal{DA}$ *and* $\boldsymbol{b} \in Im(\boldsymbol{A})$*, 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 4.4.3. *Given a linear equation instance* LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon^{DA})$ *where* $\boldsymbol{A} \in \mathcal{DA}$ *and* $\boldsymbol{b} \in Im(\boldsymbol{A})$*, 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 4.4.4. *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 non-negative 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*

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

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 will prove Theorem 4.4.2 and Theorem 4.4.3 in Section 4.5, and we will prove Theorem 4.4.4 in Section 4.6.

4.4.1 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 $\boldsymbol{d} \in \mathbb{R}^d$, solving $\partial_2 \boldsymbol{f} = \boldsymbol{d}$ can be interpreted as finding flows in the triangle space subject to edge demands in $\boldsymbol{d}$. We will encode difference-average linear equations as a 2-complex flow network.

Encoding a Single Equation. 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 interior edge is $\boldsymbol{f}(\Delta_1) - \boldsymbol{f}(\Delta_2)$ (see Figure 4.1 (a)); if we glue two triangles Δ_1, Δ_2 with opposite orientations, then the net flow $\partial_2 \boldsymbol{f}$ on the shared interior edge is $\boldsymbol{f}(\Delta_1)+\boldsymbol{f}(\Delta_2)$ (see Figure 4.1 (b)). Given an equation $\boldsymbol{a}^\top \boldsymbol{x} = b$ with the non-zero coefficients being ± 1, we can encode it by gluing more triangles as above and setting the demand of the shared interior 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 4.1 (c)).

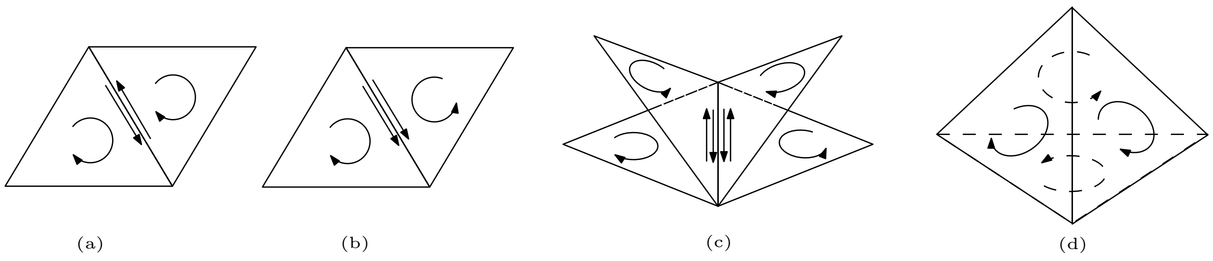

Figure 4.1: An illustration for encoding a single equation and encoding a variable.

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 4.1 (d)).

Putting It All Together. For each variable $\boldsymbol{x}(i)$ and the sphere for $\boldsymbol{x}(i)$, we create a "hole" for each equation that involves $\boldsymbol{x}(i)$, and then attach a tube. We can have an oriented triangulation of the tubes so that the triangles on the tubes have equal value as the triangles on the sphere. We then connect these tubes properly to encode each given difference and average equation.

Discussion.

- *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 $\Omega(\mathrm{nnz}(\boldsymbol{G}) \|\boldsymbol{G}\|_{\max})$. 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 $\mathrm{nnz}(\boldsymbol{A}) = O\left(\mathrm{nnz}(\boldsymbol{G}) \log \|\boldsymbol{G}\|_{\max}\right)$ (by Lemma 5.1.1). Then we can encode $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ into a 2-complex. The encoding size required to express the difference-average linear equations as a 2-complex is thus $O(\mathrm{nnz}(\boldsymbol{A}))$ (by Lemma 4.5.2). Thus, the overall encoding size required to express the original linear equation $\boldsymbol{G}\boldsymbol{y} = \boldsymbol{c}$ is now $\widetilde{O}\left(\mathrm{nnz}(\boldsymbol{G})\right)$, 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.

- *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).

4.5 Reducing $\mathcal{DA}$ to $\mathcal{B}_2$ in Feasible Case

In this section, we show that the algorithm REDUCE$\mathcal{DA}$TO$\mathcal{B}_2$ and the algorithm MAPSOLN$\mathcal{B}_2$TO$\mathcal{DA}$ reduce instances LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon)$ over $\mathcal{DA}$ to instances LEA $(\partial_2, \gamma, \epsilon')$ over $\mathcal{B}_2$, under the assumption that $\boldsymbol{b}$ is in the image of $\boldsymbol{A}$. Specifically, we show that by a proper choice of ϵ, a solution to LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon)$ can be converted to a solution to LEA $(\partial_2, \gamma, \epsilon')$ in Section 4.5.5; we upper bound the condition number of ∂_2 in Section 4.5.6.

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.

4.5.1 Reduction Algorithm

Given an instance LE $(\boldsymbol{A}, \boldsymbol{b})$ where $\boldsymbol{A}$ is a $d \times n$ matrix in $\mathcal{DA}$, the following algorithm REDUCE$\mathcal{DA}$TO$\mathcal{B}_2$ 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* α_q with a net flow demand $\boldsymbol{b}(q)$. Then,
   (a) we add a boundary component $\beta_{q,i}$ 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 4.2 for an illustration.[1]
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,

[1]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 $\mathcal{T}_{q,j}$.

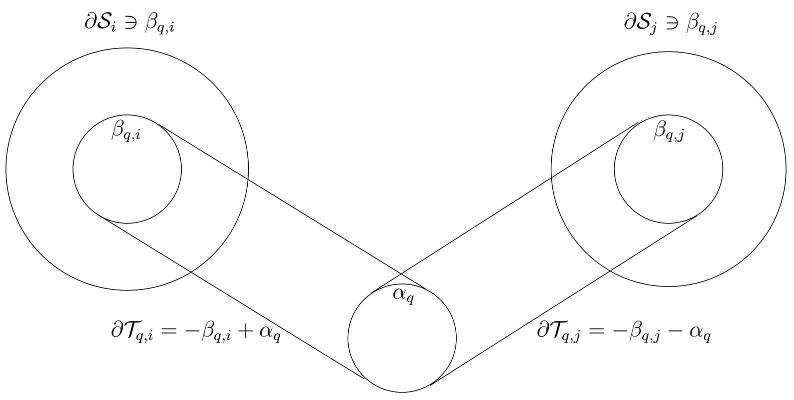
$\partial\mathcal{S}_i \ni \beta_{q,i}$
$\partial\mathcal{S}_j \ni \beta_{q,j}$
$\beta_{q,i}$
$\beta_{q,j}$
α_q
$\partial\mathcal{T}_{q,i} = -\beta_{q,i} + \alpha_q$
$\partial\mathcal{T}_{q,j} = -\beta_{q,j} - \alpha_q$

Figure 4.2: The construction for a difference equation $\boldsymbol{x}(i) - \boldsymbol{x}(j) = \boldsymbol{b}(q)$.

(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 4.3 for an illustration.[2]

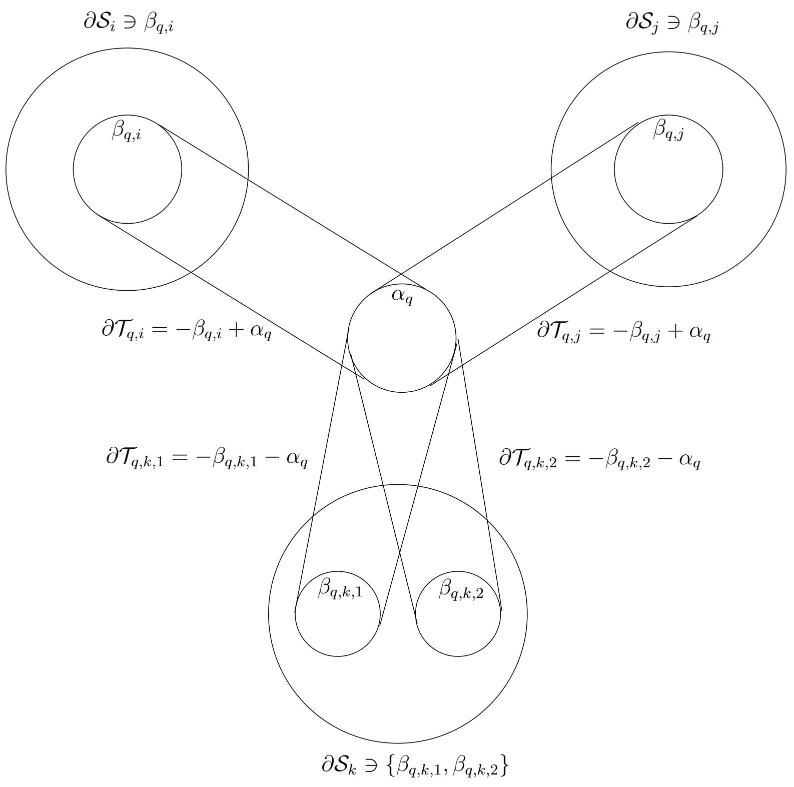
$\partial\mathcal{S}_i \ni \beta_{q,i}$
$\partial\mathcal{S}_j \ni \beta_{q,j}$
$\beta_{q,i}$
$\beta_{q,j}$
α_q
$\partial\mathcal{T}_{q,i} = -\beta_{q,i} + \alpha_q$
$\partial\mathcal{T}_{q,j} = -\beta_{q,j} + \alpha_q$
$\partial\mathcal{T}_{q,k,1} = -\beta_{q,k,1} - \alpha_q$
$\partial\mathcal{T}_{q,k,2}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

Figure 4.3: 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

[2]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.

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 4.5.1. 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 MapSolnB$_2$toDA maps a solution $\boldsymbol{f}$ to LE (∂_2, γ) to a solution $\boldsymbol{x}$ to LE $(\boldsymbol{A}, \boldsymbol{b})$.

Algorithm MapSoln$\mathcal{B}_2$to$\mathcal{DA}$

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 $\boldsymbol{A}^\top \boldsymbol{b} = \boldsymbol{0}$, 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 $\mathcal{S}_i$.

Oriented Triangulation

We provide a concrete triangulation subroutine here for the benefit of analyzing our reduction algorithm.

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 4.4 (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 4.4 (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 4.4 (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

$$\tilde{t}_i = 5b_i - 4, \qquad \tilde{m}_i = 9b_i - 6, \qquad \text{for } b_i \geq 1. \tag{4.1}$$

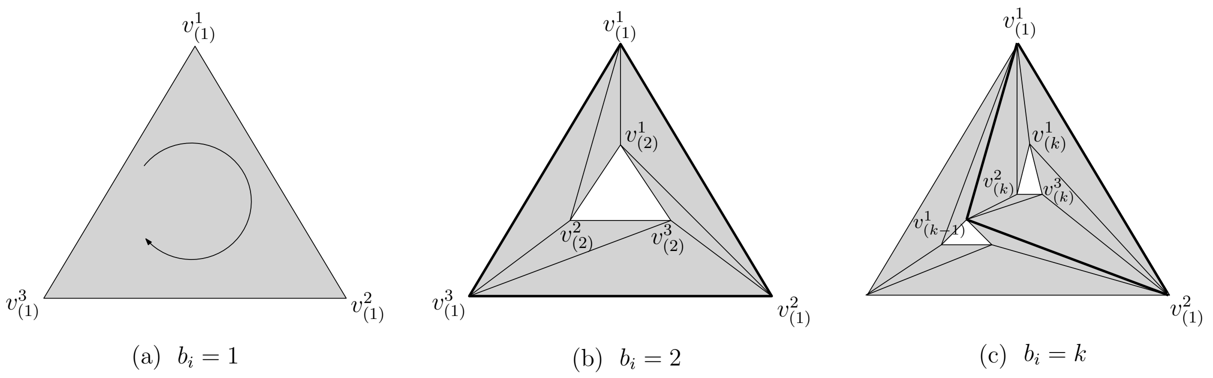

Figure 4.4: Oriented triangulation of punctured spheres. The light area represents the "holes" defined by boundary components.

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 $-\beta_{q,i,*}$,[3] 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 4.5 (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 4.5 (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.

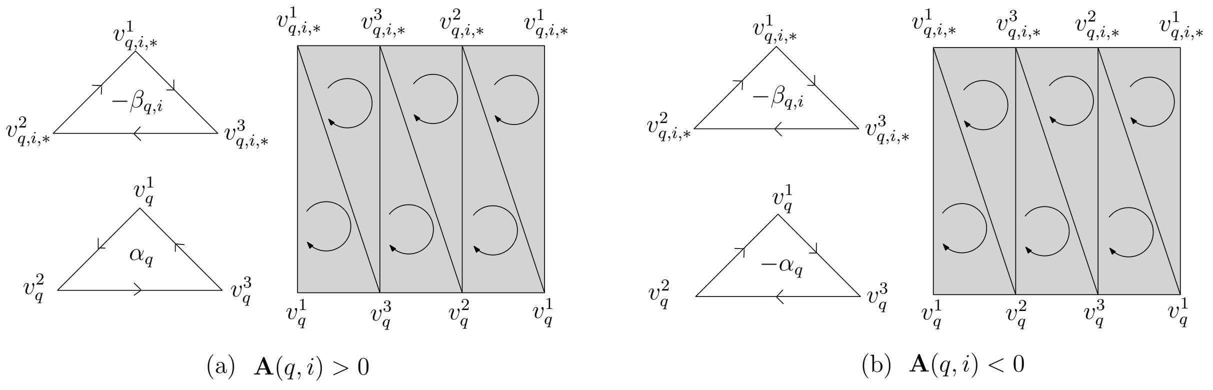
$v^1_{q,i,*}$
$v^2_{q,i,*}$
$v^3_{q,i,*}$
$-\beta_{q,i}$
v^1_q
v^2_q
v^3_q
α_q
$v^1_{q,i,*}$ $v^3_{q,i,*}$ $v^2_{q,i,*}$ $v^1_{q,i,*}$
v^1_q v^3_q v^2_q v^1_q
(a) $\mathbf{A}(q,i) > 0$
$v^1_{q,i,*}$
$v^2_{q,i,*}$
$v^3_{q,i,*}$
$-\beta_{q,i}$
v^1_q
v^2_q
v^3_q
$-\alpha_q$
$v^1_{q,i,*}$ $v^3_{q,i,*}$ $v^2_{q,i,*}$ $v^1_{q,i,*}$
v^1_q v^2_q v^3_q v^1_q
(b) $\mathbf{A}(q,i) < 0$

Figure 4.5: Oriented triangulation of tubes with opposite or identical boundary orientations.

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.

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

4.5.2 Additional Notations

To facilitate our analysis, we introduce several notions and corresponding notations about the constructed 2-complex. These notions and notations will be used in the rest of the paper.

For each $i \in [n]$, let $\mathcal{K}_i$ be the the union of the triangulated $\mathcal{S}_i$ and the triangulated tubes that are connected to $\mathcal{S}_i$, which we refer to as the ith *complex group*; let t_i be the number of triangles in $\mathcal{K}_i$; let m_i be the number of the interior edges in $\mathcal{K}_i$.

We refer to an edge on a loop α_q as a *boundary edge* and an edge not on any loop an *interior edge*. According to our triangulation, each loop α_q has three boundary edges, denoted by $\alpha_q^1 = [v_q^1, v_q^2]$, $\alpha_q^2 = [v_q^2, v_q^3]$, $\alpha_q^3 = [v_q^3, v_q^1]$. A triangle containing a boundary edge is called a *boundary triangle*. For each boundary edge α_q^r, where $q \in [d_1]$ and $r \in [3]$, corresponding to equation $\boldsymbol{x}(i) - \boldsymbol{x}(j) = \boldsymbol{b}(q)$, we denote the boundary triangles by $\Delta_{q,i,1}^r, \Delta_{q,j,1}^r$ where $\Delta_{q,i,1}^r \in \mathcal{T}_{q,i}$, $\Delta_{q,j,1}^r \in \mathcal{T}_{q,j}$; for each boundary edge α_q^r, where $q \in [d_1+1 : d_2]$ and $r \in [3]$, corresponding to equation $\boldsymbol{x}(i) + \boldsymbol{x}(j) = 2\boldsymbol{x}(k)$, we denote the boundary triangles by $\Delta_{q,i,1}^r, \Delta_{q,j,1}^r, \Delta_{q,k,1}^r, \Delta_{q,k,2}^r$ where $\Delta_{q,i,1}^r \in \mathcal{T}_{q,i}$, $\Delta_{q,j,1}^r \in \mathcal{T}_{q,j}$, $\Delta_{q,k,1}^r, \Delta_{q,k,2}^r \in \mathcal{T}_{q,k}$.

Given any two triangles $\Delta, \Delta' \in \mathcal{K}$, a *triangle path* from Δ to Δ' is an ordered collection of triangles $\mathcal{P} = [\Delta^{(0)} = \Delta, \Delta^{(1)}, \ldots, \Delta^{(l)} = \Delta']$ such that every pair of neighboring triangles shares an edge. The length of $\mathcal{P}$ is l. A triangle path can also be defined by an ordered collection of edges $\mathcal{P} = [e^{(1)}, \ldots, e^{(l)}]$, where $e^{(i)}$ denotes the edge shared by $\Delta^{(i-1)}$ and $\Delta^{(i)}$, for $i \in [l]$.

4.5.3 Algorithm Runtime and Problem Size

In this section, we firstly 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 non-zeros in the input linear equations.

Lemma 4.5.1 (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 $O\left(\|\boldsymbol{A}\|_1 + \sum_{j\in[n]} \|\boldsymbol{A}(:,j)\|_1\right) \leq O(\mathrm{nnz}(\boldsymbol{A}))$, where we use the fact $\|\boldsymbol{A}\|_{\max} = 2$.

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

Lemma 4.5.2 (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,*

- $t \leq 22\,\mathrm{nnz}(\boldsymbol{A})$;
- $m \leq 33\,\mathrm{nnz}(\boldsymbol{A})$;
- $\mathrm{nnz}(\partial_2) \leq 66\,\mathrm{nnz}(\boldsymbol{A})$.

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 (4.1), where $b_j = \sum_{i\in[d]} |\boldsymbol{A}(i,j)|$. Therefore, the number of triangles of all spheres is

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

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

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

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

$$m \le 1.5t \le 33 \operatorname{nnz}(\boldsymbol{A}).$$

Since each column of ∂_2 has exactly 3 non-zero entries, we have

$$\operatorname{nnz}(\partial_2) = 3t \le 66 \operatorname{nnz}(\boldsymbol{A}).$$

□

4.5.4 Relation Between Exact Solutions

In this section, we show that the algorithm ReduceDAToB2 and the algorithm MapSolnB2toDA reduce instances LE $(\boldsymbol{A}, \boldsymbol{b})$ over $\mathcal{DA}$ to instances LE over $\mathcal{B}_2$ by exploring the relationship between exact solutions.

Given LE $(\boldsymbol{A}, \boldsymbol{b})$ where $\boldsymbol{A}$ is a $d \times n$ matrix in $\mathcal{DA}$, let $(\partial_2, \gamma, \boldsymbol{\Delta}^c)$ be returned by ReduceDAToB2$(\boldsymbol{A}, \boldsymbol{b})$, and let $\mathcal{K}$ be the 2-complex constructed in ReduceDAToB2$(\boldsymbol{A}, \boldsymbol{b})$ and the boundary operator of $\mathcal{K}$ is ∂_2. Let $\boldsymbol{f}$ be a solution to LE $(\boldsymbol{A}, \boldsymbol{b})$.

To simplify the analysis, we reorder the columns and rows of ∂_2. The columns $[1 : t_1]$ of ∂_2 correspond to the triangles in $\mathcal{K}_1$, the columns $[t_1+1 : t_2]$ correspond to the triangles in $\mathcal{K}_2$, and so on. Then, $\boldsymbol{f}$ can be written as

$$\boldsymbol{f} = \begin{bmatrix} \boldsymbol{f}_1 \\ \vdots \\ \boldsymbol{f}_n \end{bmatrix}, \qquad \text{where } \boldsymbol{f}_i \in \mathbb{R}^{t_i},\ \forall i \in [n].$$

The rows of ∂_2 and the entries of γ are:

$$\partial_2 = \left[\begin{array}{ccc} & \boldsymbol{B}_1 & \\ & \vdots & \\ & \boldsymbol{B}_d & \\ \hdashline \boldsymbol{M}_1 & & \\ & \ddots & \\ & & \boldsymbol{M}_n \end{array}\right], \qquad \gamma = \left[\begin{array}{c} \boldsymbol{b}(1)\mathbf{1}_3 \\ \vdots \\ \boldsymbol{b}(d)\mathbf{1}_3 \\ \hdashline \mathbf{0}_{m_1} \\ \vdots \\ \mathbf{0}_{m_n} \end{array}\right], \tag{4.2}$$

Here, each submatrix $\boldsymbol{B}_q \in \{0, \pm 1\}^{3 \times t}$ corresponds to the three boundary edges $\{\alpha_q^1, \alpha_q^2, \alpha_q^3\}$; each submatrix $\boldsymbol{M}_i \in \{0, \pm 1\}^{m_i \times t_i}$ corresponds to all the interior edges in $\mathcal{K}_i$. Interior edges in $\mathcal{K}_i$ and those in $\mathcal{K}_j$ do not share endpoints if $i \neq j$. Let $\boldsymbol{M} = \mathrm{diag}(\boldsymbol{M}_1, \boldsymbol{M}_2, \ldots, \boldsymbol{M}_n)$.

Claim 4.5.3. *For each $i \in [n]$, $\boldsymbol{f}_i = \alpha \mathbf{1}_{t_i}$ for some $\alpha \in \mathbb{R}$.*

Proof. For each $i \in [n]$, we have $\boldsymbol{M}_i \boldsymbol{f}_i = \mathbf{0}$. This means that for any two triangles Δ, Δ' in $\mathcal{K}_i$ sharing an interior edge, we have $\boldsymbol{f}_i(\Delta) = \boldsymbol{f}_i(\Delta')$. By our construction of $\mathcal{K}_i$, for any two triangles Δ, Δ' in $\mathcal{K}_i$, there exists a triangle path connecting Δ and Δ'. The values of $\boldsymbol{f}_i$ at the triangles in this triangle path are equal; in particular, $\boldsymbol{f}_i(\Delta) = \boldsymbol{f}_i(\Delta')$. Thus, the values of $\boldsymbol{f}_i$ at all the triangles in $\mathcal{K}_i$ are equal, that is, $\boldsymbol{f}_i = \alpha \mathbf{1}_{t_i}$ for some $\alpha \in \mathbb{R}$. □

Lemma 4.5.4 (Exact Solvers in Feasible Case)**.** *Given a difference-average instance* LE $(\boldsymbol{A}, \boldsymbol{b})$ *where $\boldsymbol{b} \in Im(\boldsymbol{A})$, let $(\partial_2, \gamma, \boldsymbol{\Delta}^c)$ be returned by* ReduceDATo$\mathcal{B}_2$*$(\boldsymbol{A}, \boldsymbol{b})$, and let $\boldsymbol{f}$ be a solution to* LE (∂_2, γ)*. Then, $\boldsymbol{x} \leftarrow$*MapSolnB$_2$toDA*$(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{f}, \boldsymbol{\Delta}^c)$ is a solution to* LE $(\boldsymbol{A}, \boldsymbol{b})$.

Proof. By Claim 4.5.3, we can write $\boldsymbol{f}$ as

$$\boldsymbol{f} = \begin{bmatrix} \alpha_1 \mathbf{1}_{t_1} \\ \alpha_2 \mathbf{1}_{t_2} \\ \vdots \\ \alpha_n \mathbf{1}_{t_n} \end{bmatrix}, \quad \text{where } \alpha_1, \ldots, \alpha_n \in \mathbb{R}.$$

According to Algorithm MapSoln$\mathcal{B}_2$to$\mathcal{DA}$$(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{f}, \boldsymbol{\Delta}^c)$, for each $i \in [n]$, $\boldsymbol{x}(i) = \alpha_i$. Our goal is to show $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$.

- For each difference equation in LE $(\boldsymbol{A}, \boldsymbol{b})$, say $\boldsymbol{x}(i) - \boldsymbol{x}(j) = \boldsymbol{b}(q)$, we look at the equations in LE (∂_2, γ) related to $\boldsymbol{B}_q$:

$$\boldsymbol{f}_i(\Delta_{q,i}^r) - \boldsymbol{f}_j(\Delta_{q,j}^r) = \boldsymbol{b}(q), \; \forall r \in \{1, 2, 3\}.$$

  Thus $\boldsymbol{x}(i) - \boldsymbol{x}(j) = \boldsymbol{b}(q)$ holds.

- For each average equation in LE $(\boldsymbol{A}, \boldsymbol{b})$, say $\boldsymbol{x}(i) + \boldsymbol{x}(j) - 2\boldsymbol{x}(k) = \boldsymbol{b}(q) = 0$, we look at the equations in LE (∂_2, γ) related to $\boldsymbol{B}_q$:

$$\boldsymbol{f}_i(\Delta_{q,i}^r) + \boldsymbol{f}_j(\Delta_{q,j}^r) - \boldsymbol{f}_k(\Delta_{q,k,1}^r) - \boldsymbol{f}_k(\Delta_{q,k,2}^r) = 0, \; \forall r \in \{1, 2, 3\}.$$

  Thus $\boldsymbol{x}(i) + \boldsymbol{x}(j) - 2\boldsymbol{x}(k) = 0$ holds.

□

4.5.5 Relation Between Approximate Solutions

Lemma 4.5.5 (Approximate Solvers in Feasible Case)**.** *Given a difference-average instance* LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon^{DA})$ *where $\boldsymbol{b} \in Im(\boldsymbol{A})$, let $(\partial_2, \gamma, \boldsymbol{\Delta}^c)$ be returned by* ReduceDATo$\mathcal{B}_2$*$(\boldsymbol{A}, \boldsymbol{b})$. Suppose $\boldsymbol{f}$ is a solution to* LEA $(\partial_2, \gamma, \epsilon^{B_2})$ *where*

$$\epsilon^{B_2} \leq \frac{\epsilon^{DA}}{42 \, \mathrm{nnz}(\boldsymbol{A})},$$

and $\boldsymbol{x}$ is returned by MapSoln$\mathcal{B}_2$to$\mathcal{DA}$$(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{f}, \boldsymbol{\Delta}^c)$*. Then, $\boldsymbol{x}$ is a solution to* LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon^{DA})$.

Proof. Since $\boldsymbol{b} \in \operatorname{Im}(\boldsymbol{A})$, we have $\|\partial_2 \boldsymbol{f} - \gamma\|_\infty \leq \|\partial_2 \boldsymbol{f} - \gamma\|_2 \leq \epsilon^{B_2} \|\gamma\|_2$.

We claim

$$\|\boldsymbol{Ax} - \boldsymbol{b}\|_\infty \leq 24 \operatorname{nnz}(\boldsymbol{A})^{1/2} \cdot \epsilon^{B_2} \|\gamma\|_2 . \tag{4.3}$$

Then,

$$\|\boldsymbol{Ax} - \boldsymbol{b}\|_2 \leq 24 \operatorname{nnz}(\boldsymbol{A}) \cdot \epsilon^{B_2} \|\gamma\|_2 .$$

By ReduceDATo$\mathcal{B}_2(\boldsymbol{A}, \boldsymbol{b})$, $\|\gamma\|_2 \leq \sqrt{3} \|\boldsymbol{b}\|_2$. Thus,

$$\|\boldsymbol{Ax} - \boldsymbol{b}\|_2 \leq 42 \operatorname{nnz}(\boldsymbol{A})^{1/2} \cdot \epsilon^{B_2} \|\boldsymbol{b}\|_2 \leq \epsilon^{DA} \|\boldsymbol{b}\|_2 ,$$

that is, $\boldsymbol{x}$ is a solution to LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon^{DA})$.

To prove Eq. (4.3), consider an arbitrary equation in LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon^{DA})$, say $a_i \boldsymbol{x}(i) + a_j \boldsymbol{x}(j) + a_k \boldsymbol{x}(k) = \boldsymbol{b}(q)$ where $a_i, a_j, a_k \in \{-2, -1, 0, 1\}$. According to MapSoln$\mathcal{B}_2to\mathcal{DA}(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{f}, \boldsymbol{\Delta}^c)$, for each $l \in \{i, j, k\}$, $\boldsymbol{x}(l) = \boldsymbol{f}(\Delta_l)$ where $\Delta_l \in \mathcal{K}_l$ is the lth central triangle in $\boldsymbol{\Delta}^c$. Then,

$$a_i \boldsymbol{x}(i) + a_j \boldsymbol{x}(j) + a_k \boldsymbol{x}(k) = a_i \boldsymbol{f}(\Delta_i) + a_j \boldsymbol{f}(\Delta_j) + a_k \boldsymbol{f}(\Delta_k).$$

Note that the equation in LEA $(\partial_2, \gamma, \epsilon^{B_2})$ related to the boundary edge α_q^1, shared by triangles $\Delta_{q,i,1}^1, \Delta_{q,j,1}^1$ and $\Delta_{q,k,1}^1, \Delta_{q,k,2}^1$ (if the equation is an average equation), satisfies

$$\left| a_i \boldsymbol{f}(\Delta_{q,i,1}^1) + a_j \boldsymbol{f}(\Delta_{q,j,1}^1) + \frac{1}{2} a_k \left(\boldsymbol{f}(\Delta_{q,k,1}^1) + \boldsymbol{f}(\Delta_{q,k,2}^1) \right) - \boldsymbol{b}(q) \right| \leq \epsilon^{B_2} \|\gamma\|_2 .$$

For each $\Delta \in \{\Delta_{q,i,1}^1, \Delta_{q,j,1}^1, \Delta_{q,k,1}^1, \Delta_{q,k,2}^1\}$, we will replace $\boldsymbol{f}(\Delta)$ with its corresponding central triangle Δ^c. We can find a triangle path connecting Δ and Δ^c, say $\mathcal{P} = [\Delta^{(0)} = \Delta, \ldots, \Delta^{(l_q)} = \Delta^c]$, such that two adjacent triangles $\Delta^{(l)}, \Delta^{(l+1)}$ share an interior edge $e^{(l)}$. Then,

$$\begin{aligned}
|\boldsymbol{f}(\Delta) - \boldsymbol{f}(\Delta^c)| &= \left| \sum_{l=1}^{l_q} \boldsymbol{f}(\Delta^{(l-1)}) - \boldsymbol{f}(\Delta^{(l)}) \right| \leq \sum_{l=1}^{l_q} \left| \boldsymbol{f}(\Delta^{(l-1)}) - \boldsymbol{f}(\Delta^{(l)}) \right| = \sum_{l=1}^{l_q} \left| [\partial_2 \boldsymbol{f}](e^{(l)}) \right| \\
&= \sum_{l=1}^{l_q} \left| [\partial_2 \boldsymbol{f} - \gamma](e^{(l)}) \right| \qquad \text{since } \gamma(e^{(l)}) = 0 \text{ for interior edges} \\
&= \left\| [\partial_2 \boldsymbol{f} - \gamma](e^{(1)} : e^{(l_q)}) \right\|_1 \\
&\qquad \text{where the subvector corresponds to } [e^{(1)}, \ldots, e^{(l_q)}] \\
&\leq \sqrt{t_i} \left\| [\partial_2 \boldsymbol{f} - \gamma](e^{(1)} : e^{(l_q)}) \right\|_2 \qquad \text{since } l_q \leq t_i \\
&\leq \sqrt{t_i} \|\partial_2 \boldsymbol{f} - \gamma\|_2 \\
&\leq \sqrt{t_i} \cdot \epsilon^{B_2} \|\gamma\|_2 .
\end{aligned}$$

Thus,

$$\begin{aligned}
&|a_i \boldsymbol{x}(i) + a_j \boldsymbol{x}(j) + a_k \boldsymbol{x}(k) - \boldsymbol{b}(q)| \\
=\ & |a_i \boldsymbol{f}(\Delta_i) + a_j \boldsymbol{f}(\Delta_j) + a_k \boldsymbol{f}(\Delta_k) - \boldsymbol{b}(q)| \\
\le\ & \underbrace{\left| a_i \boldsymbol{f}(\Delta^1_{q,i,1}) + a_j \boldsymbol{f}(\Delta^1_{q,j,1}) + \frac{1}{2} a_k \left(\boldsymbol{f}(\Delta^1_{q,k,1}) + \boldsymbol{f}(\Delta^1_{q,k,2}) \right) - \boldsymbol{b}(q) \right|}_{\le \epsilon^{B_2} \|\gamma\|_2} \\
& + \underbrace{\left| \boldsymbol{f}(\Delta^1_{q,i,1}) - \boldsymbol{f}(\Delta_i) \right|}_{\le \sqrt{t_i} \epsilon^{B_2} \|\gamma\|_2} + \underbrace{\left| \boldsymbol{f}(\Delta^1_{q,j,1}) - \boldsymbol{f}(\Delta_j) \right|}_{\le \sqrt{t_j} \epsilon^{B_2} \|\gamma\|_2} + \underbrace{\left| \boldsymbol{f}(\Delta^1_{q,k,1}) - \boldsymbol{f}(\Delta_k) \right|}_{\le \sqrt{t_k} \epsilon^{B_2} \|\gamma\|_2} + \underbrace{\left| \boldsymbol{f}(\Delta^1_{q,k,2}) - \boldsymbol{f}(\Delta_k) \right|}_{\le \sqrt{t_k} \epsilon^{B_2} \|\gamma\|_2} \\
\le\ & 5\sqrt{t} \cdot \epsilon^{B_2} \|\gamma\|_2 \\
\le\ & 24 \, \mathrm{nnz}(\boldsymbol{A})^{1/2} \cdot \epsilon^{B_2} \|\gamma\|_2 , \qquad \text{by Lemma 4.5.2}
\end{aligned}$$

That is, $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_\infty \le 24 \, \mathrm{nnz}(\boldsymbol{A})^{1/2} \cdot \epsilon^{B_2} \|\gamma\|_2$. □

4.5.6 Bounding the Condition Number of the New Matrix

In this section, we show that the condition number of ∂_2 is upper bounded by a polynomial of $\mathrm{nnz}(\boldsymbol{A}), \kappa(\boldsymbol{A})$.

Lemma 4.5.6 (Condition Number of ∂_2). *Given a difference-average instance* LEA$(\boldsymbol{A}, \boldsymbol{b}, \epsilon^{DA})$ *where* $\boldsymbol{b} \in Im(\boldsymbol{A})$, *let* $(\partial_2, \gamma, \boldsymbol{\Delta}^c)$ *be returned by* ReduceDAToB$_2$$(\boldsymbol{A}, \boldsymbol{b})$. *Then,*

$$\kappa(\partial_2) \le 10^9 \, \mathrm{nnz}(\boldsymbol{A})^{9/2} \kappa(\boldsymbol{A})^2.$$

Note that

$$\kappa^2(\partial_2) = \kappa(\partial_2^\top \partial_2) = \frac{\lambda_{\max}(\partial_2^\top \partial_2)}{\lambda_{\min}(\partial_2^\top \partial_2)}.$$

We will upper bound $\lambda_{\max}(\partial_2^\top \partial_2)$ and lower bound $\lambda_{\min}(\partial_2^\top \partial_2)$. Our proof will heavily rely on the Courant-Fischer theorem.

Theorem 4.5.7 (The Courant-Fischer Theorem). *Let* $\boldsymbol{M}$ *be a symmetric matrix in* $\mathbb{R}^{n \times n}$ *where* $\lambda_{\max}, \lambda_{\min}$ *are its maximum and minimum non-zero eigenvalue, respectively. Then*

$$\lambda_{\max} = \max_{\boldsymbol{x} \neq \boldsymbol{0}} \frac{\boldsymbol{x}^\top \boldsymbol{M} \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{x}}, \qquad \lambda_{\min} = \min_{\boldsymbol{x} \perp Ker(\boldsymbol{M}), \boldsymbol{x} \neq \boldsymbol{0}} \frac{\boldsymbol{x}^\top \boldsymbol{M} \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{x}}.$$

The Maximum Eigenvalue

Lemma 4.5.8 (Maximum Eigenvalue). $\lambda_{\max}(\partial_2^\top \partial_2) \le 12$.

Proof. By the Courant-Fischer theorem,

$$\lambda_{\max}(\partial_2^\top \partial_2) = \max_{\boldsymbol{f} : \|\boldsymbol{f}\|_2 = 1} \boldsymbol{f}^\top \partial_2^\top \partial_2 \boldsymbol{f}.$$

Then for any $\boldsymbol{f}$ with $\|\boldsymbol{f}\|_2 = 1$,

$$\boldsymbol{f}^\top \partial_2^\top \partial_2 \boldsymbol{f} = \sum_{i=1}^{m} \left(\sum_{\Delta : e_i \in \Delta} \partial_2(e_i, \Delta) \boldsymbol{f}(\Delta) \right)^2 \leq 4 \sum_{i=1}^{m} \sum_{\Delta : e_i \in \Delta} \boldsymbol{f}^2(\Delta) = 12 \|\boldsymbol{f}\|_2^2 = 12.$$

where the inequality is by the Cauchy-Schwarz inequality.

□

The Minimum Non-Zero Eigenvalue

We start with proving a relation between the null space of $\boldsymbol{A}$ and that of ∂_2.

Lemma 4.5.9. *Let*

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{1}_{t_1} & & \\ & \vdots & \\ & & \boldsymbol{1}_{t_n} \end{bmatrix} \in \mathbb{R}^{t \times n}.$$

Then, $\boldsymbol{H}$ is a bijection from Ker($\boldsymbol{A}$) to Ker(∂_2).

Proof. Let $\boldsymbol{x} \in \mathrm{Ker}(\boldsymbol{A})$. By our construction of ∂_2, we have $\boldsymbol{H}\boldsymbol{x} \in \mathrm{Ker}(\partial_2)$. For any $\boldsymbol{f} \in \mathrm{Ker}(\partial_2)$, by the proof of Lemma 4.5.4, $\boldsymbol{f} = \boldsymbol{H}\boldsymbol{x}$ for some $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}$. □

Lemma 4.5.10 (Minimium Non-Zero Eigenvalue)**.**

$$\lambda_{\min}(\partial_2^\top \partial_2) \geq \frac{\min\{\lambda_{\min}(\boldsymbol{A}^\top \boldsymbol{A})^2, 1\}}{10^{16} d^7}. \tag{4.4}$$

Proof. By the Courant-Fischer theorem,

$$\lambda_{\min}(\partial_2^\top \partial_2) = \min_{\substack{\boldsymbol{f} \in \mathbb{R}^t : \boldsymbol{f} \perp \mathrm{Ker}(\partial_2) \\ \|\boldsymbol{f}\|_2 = 1}} \boldsymbol{f}^\top \partial_2^\top \partial_2 \boldsymbol{f}.$$

Let

$$C = \frac{\min\{\lambda_{\min}(\boldsymbol{A}^\top \boldsymbol{A}), 1\}}{10^6 d^{2.5}}.$$

We will exhaust all the vectors in $\{\boldsymbol{f} : \boldsymbol{f} \perp \mathrm{Ker}(\partial_2), \|\boldsymbol{f}\|_2 = 1\}$ by the following two cases.

Case 1. Suppose there exists $i \in [n]$ such that $\mathcal{K}_i$ contains two triangles Δ, Δ' satisfying $|\boldsymbol{f}(\Delta) - \boldsymbol{f}(\Delta')| \geq C$. Consider a triangle path in $\mathcal{K}_i$ connecting Δ and Δ', say $[\Delta^{(0)} = \Delta, \Delta^{(1)}, \ldots, \Delta^{(l)} = \Delta']$ where $l \leq 22d$. There must exists $i^* \in [l]$ such that

$$\left|\boldsymbol{f}(\Delta^{(i^*-1)}) - \boldsymbol{f}(\Delta^{(i^*)})\right| \geq \frac{C}{l}.$$

Note that

$$\boldsymbol{f}^\top \partial_2^\top \partial_2 \boldsymbol{f} \geq \left(\boldsymbol{f}(\Delta^{(i^*-1)}) - \boldsymbol{f}(\Delta^{(i^*)})\right)^2 \geq \left(\frac{C}{l}\right)^2 \geq \frac{\min\{\lambda_{\min}(\boldsymbol{A}^\top \boldsymbol{A})^2, 1\}}{10^{16} d^7}.$$

Case 2. Suppose for every $i \in [n]$ and every two $\Delta, \Delta' \in \mathcal{K}_i$, we have $|\boldsymbol{f}(\Delta) - \boldsymbol{f}(\Delta')| < C$. We write $\boldsymbol{f}$ as

$$\boldsymbol{f} = \tilde{\boldsymbol{f}} + \boldsymbol{\epsilon} = \begin{bmatrix} \alpha_1 \mathbf{1}_{t_1} \\ \alpha_2 \mathbf{1}_{t_2} \\ \vdots \\ \alpha_n \mathbf{1}_{t_n} \end{bmatrix} + \boldsymbol{\epsilon},$$

where α_i is the value of $\boldsymbol{f}$ at the central triangle of $\mathcal{K}_i$. Then,

$$\begin{aligned} \|\boldsymbol{\epsilon}\|_2 &< \sqrt{t}C \le \sqrt{22d}C = o(1), \\ \left\|\tilde{\boldsymbol{f}}\right\|_2 &= \|\boldsymbol{f} - \boldsymbol{\epsilon}\|_2 \in \left(\frac{1}{2}, 2\right). \end{aligned}$$

We lower bound the quadratic value:

$$\begin{aligned} \boldsymbol{f}^\top \partial_2^\top \partial_2 \boldsymbol{f} &= \tilde{\boldsymbol{f}}^\top \partial_2^\top \partial_2 \tilde{\boldsymbol{f}} + 2\tilde{\boldsymbol{f}}^\top \partial_2^\top \partial_2 \boldsymbol{\epsilon} + \boldsymbol{\epsilon}^\top \partial_2^\top \partial_2 \boldsymbol{\epsilon} \\ &\ge \tilde{\boldsymbol{f}}^\top \partial_2^\top \partial_2 \tilde{\boldsymbol{f}} - 2\left\|\partial_2^\top \partial_2 \tilde{\boldsymbol{f}}\right\|_2 \|\boldsymbol{\epsilon}\|_2 \\ &\ge \tilde{\boldsymbol{f}}^\top \partial_2^\top \partial_2 \tilde{\boldsymbol{f}} - 4\left\|\partial_2^\top \partial_2\right\|_2 \|\boldsymbol{\epsilon}\|_2 \\ &\ge \tilde{\boldsymbol{f}}^\top \partial_2^\top \partial_2 \tilde{\boldsymbol{f}} - 48\sqrt{22d}C, && \text{by Lemma 4.5.8} \end{aligned}$$

Note that

$$\tilde{\boldsymbol{f}}^\top \partial_2^\top \partial_2 \tilde{\boldsymbol{f}} = 3\|\boldsymbol{A}\boldsymbol{\alpha}\|_2^2,$$

where $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_n)^\top$. Write $\boldsymbol{\alpha} = \boldsymbol{\alpha}_\perp + \boldsymbol{\alpha}_0$, where $\boldsymbol{\alpha}_\perp$ is orthogonal to the null space of $\boldsymbol{A}$ and $\boldsymbol{\alpha}_0$ is in the null space of $\boldsymbol{A}$. Then,

$$\tilde{\boldsymbol{f}}^\top \partial_2^\top \partial_2 \tilde{\boldsymbol{f}} \ge 3\lambda_{\min}(\boldsymbol{A}^\top \boldsymbol{A}) \|\boldsymbol{\alpha}_\perp\|_2^2. \tag{4.5}$$

It remains to lower bound $\|\boldsymbol{\alpha}_\perp\|_2$. We can write $\tilde{\boldsymbol{f}} = \boldsymbol{H}\boldsymbol{\alpha}_\perp + \boldsymbol{H}\boldsymbol{\alpha}_0$. By Lemma 4.5.9, $\boldsymbol{H}\boldsymbol{\alpha}_0 \in \mathrm{Ker}(\partial_2)$ and $\boldsymbol{H}\boldsymbol{\alpha}_\perp \perp \mathrm{Ker}(\partial_2)$. On the other hand, $\tilde{\boldsymbol{f}} = \boldsymbol{f} - \boldsymbol{\epsilon}$ and $\boldsymbol{f} \perp \mathrm{Ker}(\partial_2)$. We know

$$\|\boldsymbol{H}\boldsymbol{\alpha}_\perp\|_2 \ge \|\boldsymbol{f}\|_2 - \|\boldsymbol{\epsilon}\|_2,$$

and thus

$$\|\boldsymbol{\alpha}_\perp\|_2 \ge \frac{1}{\sqrt{t}} \|\boldsymbol{H}\boldsymbol{\alpha}_\perp\|_2 \ge \frac{1}{\sqrt{t}} - C > \frac{1}{\sqrt{22d}}.$$

Together with Eq. (4.5),

$$\tilde{\boldsymbol{f}}^\top \partial_2^\top \partial_2 \tilde{\boldsymbol{f}} \ge \frac{3\lambda_{\min}(\boldsymbol{A}^\top \boldsymbol{A})}{22d}.$$

This completes the proof. □

Proof of Lemma 4.5.6. The proof follows by combining Lemma 4.5.8 and 4.5.10. □

4.6 Reducing $\mathcal{DA}$ to $\mathcal{B}_2$ in General Case

In this section, we show how to reduce LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon^{DA})$ to instances LEA over $\mathcal{B}_2$ without requiring the assumption that $\boldsymbol{b}$ is in the image of $\boldsymbol{A}$ as we did in earlier sections. In this more general case, LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon^{DA})$ aims to compute an *approximate* solution to $\arg\min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2 = \arg\min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{b}\|_2$.

We remark that our reduction for this general case does not work for LE $(\boldsymbol{A}, \boldsymbol{b})$, which computes an *exact* solution to $\min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2$. Approximate linear equation solvers, however, can be more interesting in practice, since numerical errors occur unavoidably during data collection and computation, and approximate solutions may be computed much faster than exact solutions.

4.6.1 Warm-Up: Reweighting Infeasible Equations to Preserve Solutions

There is a crucial difference between the reductions we use for LE $(\boldsymbol{A}, \boldsymbol{b})$ and LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon^{DA})$ when $\boldsymbol{b} \in \mathrm{Im}(\boldsymbol{A})$ and in the general case when we may have $\boldsymbol{b} \notin \mathrm{Im}(\boldsymbol{A})$.

To understand this, consider the following feasible system of just two linear equations in two variables x and y.

$$\begin{aligned} x - y &= 1, \\ -x + y &= -1. \end{aligned}$$

A feasible solution is $x = 1$ and $y = 0$. Now, suppose we add another linear equation that is satisfied by all solutions to the previous equations, for example, we can simply repeat the first constraint $x - y = 1$. It remains true that the existing solutions are feasible.

Now, in contrast, consider an infeasible system of two linear equations in variables x and y.

$$\begin{aligned} x - y &= 1, \\ -x + y &= 0. \end{aligned}$$

This linear equation is *not* feasible. In particular, we can consider the associated minimization problem $\arg\min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2$ with $\boldsymbol{A} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$ and $\boldsymbol{b} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, for which one minimizing solution is $\boldsymbol{x} = \begin{bmatrix} 1/2 \\ 0 \end{bmatrix}$. Notice that if we add a row which simply repeats the first constraint, i.e. $x - y = 1$, then the resulting minimization problem has

$$\boldsymbol{A} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \\ 1 & -1 \end{bmatrix} \text{ and } \boldsymbol{b} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix},$$

and now $\boldsymbol{x} = \begin{bmatrix} 1/2 \\ 0 \end{bmatrix}$ is no longer a minimizing solution to $\arg\min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2$. In particular, $\boldsymbol{x} = \begin{bmatrix} 1/2 \\ 0 \end{bmatrix}$ achieves a value of $\sqrt{3}/2$, while $\boldsymbol{x}' = \begin{bmatrix} 2/3 \\ 0 \end{bmatrix}$ achives the smaller value $\sqrt{6}/3$.

However, if we reweigh the first and last row of our system of inequalities by a factor $1/\sqrt{2}$, so that

$$\boldsymbol{A} = \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ -1 & 1 \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix} \text{ and } \boldsymbol{b} = \begin{bmatrix} 1/\sqrt{2} \\ 0 \\ 1/\sqrt{2} \end{bmatrix},$$

then we in fact maintain that the original solution $\boldsymbol{x} = \begin{bmatrix} 1/2 \\ 0 \end{bmatrix}$ stays optimal. Thus, when we modify an infeasible system of linear equations, we have to be very careful about the weight we assign to different constraints if we are to (approximately) preserve the correspondence between the optimal solutions of the original and the final problems. In the following section, we describe a reweighting scheme which can be combined with our existing reductions to ensure that optimal solutions are (approximately) preserved, even when the original problem is infeasible.

4.6.2 Reduction Algorithm

Our reduction is almost the same as the algorithm REDUCE$\mathcal{DA}$TO$\mathcal{B}_2$, except that we assign edge weights for the constructed 2-complex.

Suppose we are given an instance LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon^{DA})$ where $\boldsymbol{A}$ is a $d \times n$ matrix in $\mathcal{DA}$. Let $(\partial_2, \gamma, \boldsymbol{\Delta}^c) \leftarrow$ REDUCE$\mathcal{DA}$TO$\mathcal{B}_2(\boldsymbol{A}, \boldsymbol{b})$. Let $\mathcal{K}$ be the 2-complex whose boundary operator is ∂_2. We compute the edge weights of $\mathcal{K}$ as follows.

1. For each boundary triangle $\Delta^1_{q,i,*}$ where $q \in [d], i \in [n]$ and $* \in \{1, 2\}$, we find a minimal triangle path $\mathcal{P}^1_{q,i,*}$ in $\mathcal{K}_i$ from the central triangle Δ_i to $\Delta^1_{q,i,*}$. Let $E^1_{q,i,*}$ be the set of edges shared by neighboring triangles in $\mathcal{P}^1_{q,i,*}$.[4]

2. For each interior edge e and equation q, let $k_{q,e}$ be the number of triangle paths indexed by equation q that contain e. For each equation q, let l_q be the sum of the lengths of all the triangle paths indexed by equation q. Then, we set the weight for edge e to be
$$w_e = \begin{cases} 1, & \text{if } e \text{ is a boundary edge} \\ \alpha \sum_{q \in [d]} k_{q,e} l_q, & \text{if } e \text{ is an interior edge} \end{cases}, \tag{4.6}$$
where
$$\alpha = \frac{2}{(\epsilon^{DA})^2}.$$

Let $\boldsymbol{W}$ be a diagonal matrix whose diagonals are the edge weights. We return $(\boldsymbol{W}, \partial_2, \gamma, \boldsymbol{\Delta}^c)$.

Note that an edge in $\mathcal{K}$ may have weight 0. If we want to make all the edge weights positive, we can impose polynomially small edge weights for these 0-weight edges, which only affects the error propagation and condition number up to polynomial factors.

We refer to the above algorithm as REDUCEREG$\mathcal{DA}$TO$\mathcal{B}_2$. We use the algorithm MAPSOLN$\mathcal{B}_2$TO$\mathcal{DA}$ to map a solution to LEA $(\boldsymbol{W}^{1/2}\partial_2, \boldsymbol{W}^{1/2}\gamma, \epsilon^{B_2})$ back to a solution to LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon^{DA})$, where we choose

$$\epsilon^{B_2} \le \frac{\epsilon^{DA}}{\sqrt{3\left(1 + \frac{1}{\alpha}\left\|\boldsymbol{b}\right\|_2^2 \operatorname{nnz}(\boldsymbol{A}) \left\|\boldsymbol{A}\right\|_{\max}^2\right)}}.$$

[4]Note that any two neighboring triangles in $\mathcal{P}^1_{q,i,*}$ share exactly one edge. So, the length of $\mathcal{P}^1_{q,i,*}$ equals $|E^1_{q,i,*}|$.

Computing the Edge Weights in Linear Time. Since the weight of an interior edge e only depends on $\mathcal{K}_i$ that contains edge e, we will compute the edge weights for the edges in each $\mathcal{K}_i$ separately.

For each $i \in [n]$, consider a graph G_i whose vertices are the triangles in $\mathcal{K}_i$ and the two vertices are adjacent if and only if the two corresponding triangles share an edge in $\mathcal{K}_i$. We run the breadth-first-search (BFS) to construct a shortest-path tree T_i of G_i rooted at Δ_i. For each boundary triangle $\Delta^1_{q,i,*}$, we choose the triangle path $\mathcal{P}^1_{q,i,*}$ to be the triangle path induced by T_i from the root Δ_i to the node corresponding to $\Delta^1_{q,i,*}$, whose length is the height of the node $\Delta^1_{q,i,*}$. Since we are only interested in these triangle paths, for simplicity, we remove a node from T_i if it is not a boundary triangle $\Delta^1_{q,i,*}$ for some q and none of its descendant is a boundary triangle $\Delta^1_{q,i,*}$ for some q. After this operation, every leaf of T_i is a boundary triangle $\Delta^1_{q,i,*}$ for some q.

Our goal is to count $\sum_{q\in[d]} k_{q,e} l_q$ for each edge in the tree T_i. We observe that an edge e appears in a triangle path from the root Δ_i to a boundary triangle $\Delta^1_{q,i,*}$ if and only if $\Delta^1_{q,i,*}$ is a descendant of an end-node of e with the higher height. First, we BFS traverse the tree T_i to store the height of each boundary triangle node. Then, we traverse the tree T_i to store l_q at each boundary triangle node $\Delta^1_{q,i,*}$ for each $q \in [d]$. Next, we traverse the tree T_i from the leaf nodes to the root to count $\sum_{q\in[d]} k_{q,e} l_q$ for each edge e. The total runtime is linear in the number of triangles in $\mathcal{K}_j$.

4.6.3 Relation Between Exact Solutions

In this section, we show that by reweighting all the edges in $\mathcal{K}$, an exact solution to $(\boldsymbol{W}^{1/2}\partial_2, \boldsymbol{W}^{1/2}\gamma)$ is close to an exact solution to $(\boldsymbol{A}, \boldsymbol{b})$, stated in Claim 4.6.1. Claim 4.6.1 plays a key role in analyzing approximate solutions and condition numbers.

Claim 4.6.1. *For any $\boldsymbol{f} \in \mathbb{R}^t$ and $\boldsymbol{x} \leftarrow$ $\textsc{MapSoln}\mathcal{B}_2\textsc{to}\mathcal{DA}(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{f}, \boldsymbol{\Delta}^c)$,*

$$\frac{\alpha}{\alpha+1}\left\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\right\|_2^2 \le \left\|\boldsymbol{W}^{1/2}\partial_2\boldsymbol{f} - \boldsymbol{W}^{1/2}\gamma\right\|_2^2.$$

Note that if $\boldsymbol{f}$ satisfies $\partial_2\boldsymbol{f} = \gamma$, then $\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_2^2 = \|\partial_2\boldsymbol{f}-\gamma\|_2^2$. Claim 4.6.1 states that by our choice of weights in $\boldsymbol{W}$, we can generalize this relation to more general $\boldsymbol{f}$.

Proof. For the convenience of analysis, we construct an auxiliary boundary matrix $\hat{\partial}_2$. For each interior edge e, let k_e be the number of all the triangle paths containing e, and we split the row $\partial_2(e)$ into k_e copies. For each copy related to equation q, we assign its weight to be $\alpha \cdot l_q$. Let $\hat{\boldsymbol{W}}$ be the auxiliary weight matrix, and let $\hat{\gamma}$ be the auxiliary demand vector. We can check

$$\left\|\boldsymbol{W}^{1/2}\partial_2\boldsymbol{f} - \boldsymbol{W}^{1/2}\gamma\right\|_2^2 = \left\|\hat{\boldsymbol{W}}^{1/2}\hat{\partial}_2\boldsymbol{f} - \hat{\boldsymbol{W}}^{1/2}\hat{\gamma}\right\|_2^2. \tag{4.7}$$

We reorder rows of the matrices $\hat{\boldsymbol{W}}, \hat{\partial}_2$ and the vector $\hat{\gamma}$ in the following way. For each $q \in [d]$, let E_q be a (multi)set that is the union of the shared edges in the triangle paths indexed by q [5]. Then, we reorder rows of $\hat{\boldsymbol{W}}, \hat{\partial}_2, \hat{\gamma}$ by grouping those corresponding

[5]If the qth equation is a difference equation, then every edge appears in E_q at most once; if the qth equation is an average equation, then some edges may appear twice.

to the edges in $E_q \cup \{\alpha_q^1\}$:

$$\hat{\partial}_2 = \begin{bmatrix} \boldsymbol{G}_1 \\ \vdots \\ \boldsymbol{G}_d \end{bmatrix}, \qquad \text{where} \quad \boldsymbol{G}_q = \begin{bmatrix} \boldsymbol{B}_q \\ \boldsymbol{M}_q \end{bmatrix}. \tag{4.8}$$

where $\boldsymbol{B}_q$ corresponds to the boundary edge α_q^1 and $\boldsymbol{M}_q$ is the submatrix corresponding to all the interior edges in E_q. Correspondingly, we write

$$\hat{\boldsymbol{W}} = \begin{bmatrix} \hat{\boldsymbol{W}}_1 & & \\ & \ddots & \\ & & \hat{\boldsymbol{W}}_d \end{bmatrix} \quad \text{and} \quad \hat{\gamma} = \begin{bmatrix} \hat{\gamma}_1 \\ \vdots \\ \hat{\gamma}_d \end{bmatrix}, \qquad \text{where} \quad \hat{\gamma}_q = \begin{bmatrix} \boldsymbol{b}(q) \\ \boldsymbol{0}_{|E_q|} \end{bmatrix}. \tag{4.9}$$

Let

$$\hat{w}_{q,e} \stackrel{\text{def}}{=} \hat{\boldsymbol{W}}_q(e,e) = \begin{cases} 1, & \text{if } e \text{ is a boundary edge} \\ \alpha \cdot l_q, & \text{if } e \text{ is an interior edge} \end{cases}.$$

For each $q \in [d]$, we define

$$\epsilon_q = \boldsymbol{A}(q)\boldsymbol{x} - \boldsymbol{b}(q), \quad \xi_q = \boldsymbol{B}_q \boldsymbol{f} - \boldsymbol{b}(q), \quad \delta_e = \boldsymbol{M}_q(e)\boldsymbol{f}.$$

We can check

$$\epsilon_q = \boldsymbol{1}^\top \left(\boldsymbol{G}_q \boldsymbol{f} - \hat{\gamma}_q\right) = \xi_q + \sum_{e \in E_q} \delta_e.$$

By the Cauchy-Schwarz inequality,

$$\epsilon_q^2 = \left(\xi_q + \sum_{e\in E_q} \delta_e\right)^2 \le \left(1 + \sum_{e \in E_q} \frac{1}{\hat{w}_{q,e}}\right)\left(\xi_q^2 + \sum_{e\in E_q} \hat{w}_{q,e}\delta_e^2\right) = \left(1 + \frac{1}{\alpha}\right)\left(\xi_q^2 + \sum_{e \in E_q} \hat{w}_{q,e}\delta_e^2\right).$$

That is,

$$\|\boldsymbol{A}(q)\boldsymbol{x} - \boldsymbol{b}(q)\|_2^2 \le \left(1 + \frac{1}{\alpha}\right) \left\| \hat{\boldsymbol{W}}_q^{1/2} \boldsymbol{G}_q \boldsymbol{f} - \hat{\boldsymbol{W}}_q^{1/2} \hat{\gamma}_q \right\|_2^2.$$

Summing over all d equations, we get

$$\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 \le \left(1 + \frac{1}{\alpha}\right) \left\| \hat{\boldsymbol{W}}^{1/2} \hat{\partial}_2 \boldsymbol{f} - \hat{\boldsymbol{W}}^{1/2} \hat{\gamma} \right\|_2^2.$$

By Eq. (4.7), we get the inequality in the statement. $\square$

Claim 4.6.1 implies the following relation between the exact solutions to $(\boldsymbol{A}, \boldsymbol{b})$ and those to $(\boldsymbol{W}^{1/2}\partial_2, \boldsymbol{W}^{1/2}\gamma)$.

Lemma 4.6.2. *Given any $d \times n$ matrix $\boldsymbol{A} \in \mathcal{DA}$ and vector $\boldsymbol{b} \in \mathbb{R}^d$, let $(\boldsymbol{W}, \partial_2, \gamma, \boldsymbol{\Delta}^c) \leftarrow$ ReduceReg$\mathcal{DA}$To$\mathcal{B}_2(\boldsymbol{A}, \boldsymbol{b}, \epsilon^{DA})$. Then,*

$$\frac{\alpha}{\alpha + 1} \min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 \le \min_{\boldsymbol{f}} \left\| \boldsymbol{W}^{1/2}\partial_2 \boldsymbol{f} - \boldsymbol{W}^{1/2}\gamma \right\|_2^2 \le \min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2.$$

Remark that Lemma 4.6.2 can also be stated as

$$\frac{\alpha}{\alpha+1}\left\|(\boldsymbol{I}-\boldsymbol{\Pi}_{\boldsymbol{A}})\boldsymbol{b}\right\|_2^2 \leq \left\|(\boldsymbol{I}-\boldsymbol{\Pi}_{\boldsymbol{W}^{1/2}\partial_2})\,\boldsymbol{W}^{1/2}\gamma\right\|_2^2 \leq \left\|(\boldsymbol{I}-\boldsymbol{\Pi}_{\boldsymbol{A}})\boldsymbol{b}\right\|_2^2.$$

We do not prove equalities. But as $\alpha \to \infty$, the leftmost hand side and the rightmost hand side are equal.

Proof. Let $\boldsymbol{f}^* \in \arg\min_{\boldsymbol{f}} \left\|\boldsymbol{W}^{1/2}\partial_2\boldsymbol{f} - \boldsymbol{W}^{1/2}\gamma\right\|_2$ and $\boldsymbol{x}^* \in \arg\min_{\boldsymbol{x}} \left\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\right\|_2$. Let $\boldsymbol{x} \leftarrow$ MapSolnB$_2$toDA$(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{W}^{1/2}\partial_2, \boldsymbol{f}^*)$. By Claim 4.6.1

$$\frac{\alpha}{\alpha+1}\left\|\boldsymbol{A}\boldsymbol{x}^*-\boldsymbol{b}\right\|_2^2 \leq \frac{\alpha}{\alpha+1}\left\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\right\|_2^2 \leq \left\|\boldsymbol{W}^{1/2}\partial_2\boldsymbol{f}^* - \boldsymbol{W}^{1/2}\gamma\right\|_2^2,$$

which is the first inequality in the lemma statement. Let

$$\boldsymbol{f} = \begin{bmatrix} \boldsymbol{x}^*(1)\boldsymbol{1}_{t_1} \\ \vdots \\ \boldsymbol{x}^*(n)\boldsymbol{1}_{t_n} \end{bmatrix}.$$

Then,

$$\left\|\boldsymbol{W}^{1/2}\partial_2\boldsymbol{f}^* - \boldsymbol{W}^{1/2}\gamma\right\|_2^2 \leq \left\|\boldsymbol{W}^{1/2}\partial_2\boldsymbol{f} - \boldsymbol{W}^{1/2}\gamma\right\|_2^2 = \left\|\boldsymbol{A}\boldsymbol{x}^*-\boldsymbol{b}\right\|_2^2,$$

which is the second inequality in the lemma statement. □

4.6.4 Relation Between Approximate Solutions

Linear equation problem LE $(\boldsymbol{A}, \boldsymbol{b})$ aims to find a vector $\boldsymbol{x}$ such that $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{b}$. In our setting, both $\boldsymbol{A}$ and $\boldsymbol{b}$ have integer entries. We will need the following claim to lower bound $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{b}$.

Claim 4.6.3. *Let $\boldsymbol{A} \in \mathbb{R}^{d\times n}$ and $\boldsymbol{b} \in \mathbb{R}^d$ such that $\left\|\boldsymbol{A}^\top\boldsymbol{b}\right\|_2 \geq 1$. Then,*

$$\left\|\boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{b}\right\|_2^2 \geq \frac{1}{\lambda_{\max}\left(\boldsymbol{A}^\top\boldsymbol{A}\right)}.$$

Proof. Note that

$$\left\|\boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{b}\right\|_2^2 = \boldsymbol{b}^\top\boldsymbol{A}(\boldsymbol{A}^\top\boldsymbol{A})^\dagger\boldsymbol{A}^\top\boldsymbol{b} \geq \lambda_{\min}((\boldsymbol{A}^\top\boldsymbol{A})^\dagger)\left\|\boldsymbol{A}^\top\boldsymbol{b}\right\|_2^2 \geq \frac{1}{\lambda_{\max}\left(\boldsymbol{A}^\top\boldsymbol{A}\right)}.$$

□

Claim 4.6.3 and Lemma 4.6.2 enable us to relate $\left\|\boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{b}\right\|_2$ with $\left\|\boldsymbol{\Pi}_{\boldsymbol{W}^{1/2}\partial_2}\boldsymbol{W}^{1/2}\gamma\right\|_2$.

Claim 4.6.4.

$$\left\|\boldsymbol{\Pi}_{\boldsymbol{W}^{1/2}\partial_2}\boldsymbol{W}^{1/2}\gamma\right\|_2^2 \leq \left(1+\frac{1}{\alpha}\lambda_{\max}(\boldsymbol{A}^\top\boldsymbol{A})\left\|\boldsymbol{b}\right\|_2^2\right)\left\|\boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{b}\right\|_2^2.$$

Proof. We apply Lemma 4.6.2.

$$\begin{aligned}
\|\boldsymbol{\Pi_A b}\|_2^2 &= \|\boldsymbol{b}\|_2^2 - \|(\boldsymbol{I} - \boldsymbol{\Pi_A})\boldsymbol{b}\|_2^2 \\
&\geq \left\|\boldsymbol{W}^{1/2}\gamma\right\|_2^2 - \frac{\alpha+1}{\alpha}\left\|(\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{W}^{1/2}\partial_2})\boldsymbol{W}^{1/2}\gamma\right\|_2^2 \\
&= \left\|\boldsymbol{\Pi}_{\boldsymbol{W}^{1/2}\partial_2}\boldsymbol{W}^{1/2}\gamma\right\|_2^2 - \frac{1}{\alpha}\left\|(\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{W}^{1/2}\partial_2})\boldsymbol{W}^{1/2}\gamma\right\|_2^2 \\
&\geq \left\|\boldsymbol{\Pi}_{\boldsymbol{W}^{1/2}\partial_2}\boldsymbol{W}^{1/2}\gamma\right\|_2^2 - \frac{1}{\alpha}\|(\boldsymbol{I} - \boldsymbol{\Pi_A})\boldsymbol{b}\|_2^2.
\end{aligned}$$

Since $\boldsymbol{A}, \boldsymbol{b}$ have integer entries, by Claim 4.6.3,

$$\|(\boldsymbol{I} - \boldsymbol{\Pi_A})\boldsymbol{b}\|_2^2 \leq \|\boldsymbol{b}\|_2^2 \leq \|\boldsymbol{b}\|_2^2 \cdot \lambda_{\max}(\boldsymbol{A}^\top\boldsymbol{A})\,\|\boldsymbol{\Pi_A b}\|_2^2.$$

Thus,

$$\left\|\boldsymbol{\Pi}_{\boldsymbol{W}^{1/2}\partial_2}\boldsymbol{W}^{1/2}\gamma\right\|_2^2 \leq \left(1 + \frac{1}{\alpha}\lambda_{\max}(\boldsymbol{A}^\top\boldsymbol{A})\,\|\boldsymbol{b}\|_2^2\right)\|\boldsymbol{\Pi_A b}\|_2^2.$$

□

Now, we apply Claim 4.6.1 and Lemma 4.6.4 to prove a relation between approximate solutions.

Lemma 4.6.5 (Approximate Solvers in General Case)**.** *Given a difference-average instance* LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon^{DA})$*, let* $(\boldsymbol{W}, \partial_2, \gamma, \boldsymbol{\Delta}^c) \leftarrow$ ReduceReg$\mathcal{DA}$To$\mathcal{B}_2(\boldsymbol{A}, \boldsymbol{b}, \alpha)$*. Suppose* $\boldsymbol{f}$ *is a solution to* LEA $(\boldsymbol{W}^{1/2}\partial_2, \boldsymbol{W}^{1/2}\gamma, \epsilon^{B_2})$*, where* $\epsilon^{B_2} \leq \frac{\epsilon^{DA}}{10}$*. Let* $\boldsymbol{x} \leftarrow$ MapSoln$\mathcal{B}_2$to$\mathcal{DA}(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{f}, \boldsymbol{\Delta}^c)$*. Then,* $\boldsymbol{x}$ *is a solution to* LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon^{DA})$*.*

Proof. By Claim 4.6.1, we have

$$\|\boldsymbol{Ax} - \boldsymbol{b}\|_2^2 \leq \frac{\alpha+1}{\alpha}\left\|\boldsymbol{W}^{1/2}\partial_2\boldsymbol{f} - \boldsymbol{W}^{1/2}\gamma\right\|_2^2. \tag{4.10}$$

Also note that

$$\begin{aligned}
\|\boldsymbol{Ax} - \boldsymbol{b}\|_2^2 &= \|\boldsymbol{Ax} - \boldsymbol{\Pi_A b}\|_2^2 + \|(\boldsymbol{I} - \boldsymbol{\Pi_A})\boldsymbol{b}\|_2^2, \\
\left\|\boldsymbol{W}^{1/2}\partial_2\boldsymbol{f} - \boldsymbol{W}^{1/2}\gamma\right\|_2^2 &= \left\|\boldsymbol{W}^{1/2}\partial_2\boldsymbol{f} - \boldsymbol{\Pi}_{\boldsymbol{W}^{1/2}\partial_2}\boldsymbol{W}^{1/2}\gamma\right\|_2^2 + \left\|(\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{W}^{1/2}\partial_2})\boldsymbol{W}^{1/2}\gamma\right\|_2^2.
\end{aligned}$$

Plugging these into Eq. (4.10) and apply Lemma 4.6.2,

$$\begin{aligned}
\|\boldsymbol{Ax} - \boldsymbol{\Pi_A b}\|_2^2 &\leq \frac{\alpha+1}{\alpha}\left(\left\|\boldsymbol{W}^{1/2}\partial_2\boldsymbol{f} - \boldsymbol{\Pi}_{\boldsymbol{W}^{1/2}\partial_2}\boldsymbol{W}^{1/2}\gamma\right\|_2^2 + \left\|(\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{W}^{1/2}\partial_2})\boldsymbol{W}^{1/2}\gamma\right\|_2^2\right) \\
&\quad - \|(\boldsymbol{I} - \boldsymbol{\Pi_A})\boldsymbol{b}\|_2^2 \\
&\leq \frac{\alpha+1}{\alpha}\left\|\boldsymbol{W}^{1/2}\partial_2\boldsymbol{f} - \boldsymbol{\Pi}_{\boldsymbol{W}^{1/2}\partial_2}\boldsymbol{W}^{1/2}\gamma\right\|_2^2 + \frac{1}{\alpha}\|(\boldsymbol{I} - \boldsymbol{\Pi_A})\boldsymbol{b}\|_2^2.
\end{aligned}$$

By the fact that $\boldsymbol{f}$ is a solution to LEA $(\boldsymbol{W}^{1/2}\partial_2, \boldsymbol{W}^{1/2}\gamma, \epsilon^{B_2})$ and by Claim 4.6.4,

$$\left\|\boldsymbol{W}^{1/2}\partial_2\boldsymbol{f} - \boldsymbol{\Pi}_{\boldsymbol{W}^{1/2}\partial_2}\boldsymbol{W}^{1/2}\gamma\right\|_2^2 \leq \left(\epsilon^{B_2}\right)^2\left\|\boldsymbol{\Pi}_{\boldsymbol{W}^{1/2}\partial_2}\boldsymbol{W}^{1/2}\gamma\right\|_2^2 \leq \frac{\left(\epsilon^{DA}\right)^2}{3}\|\boldsymbol{\Pi_A b}\|_2^2.$$

In addition,

$$\frac{1}{\alpha}\|(\boldsymbol{I} - \boldsymbol{\Pi_A})\boldsymbol{b}\|_2^2 \leq \frac{1}{\alpha}\|\boldsymbol{b}\|_2^2 \leq \frac{\left(\epsilon^{DA}\right)^2}{2}\|\boldsymbol{\Pi_A b}\|_2^2.$$

Thus,

$$\|\boldsymbol{Ax} - \boldsymbol{\Pi_A b}\|_2^2 \leq \left(\epsilon^{DA}\right)^2\|\boldsymbol{\Pi_A b}\|_2^2,$$

that is, $\boldsymbol{x}$ is a solution to LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon^{DA})$. □

4.6.5 Bounding the Condition Number of the New Matrix

We will upper bound the maximum eigenvalue of $\partial_2^\top \boldsymbol{W} \partial_2$ and lower bound its minimum non-zero eigenvalue. The proofs are similar to those in Section 4.5.6, which bound the eigenvalues of $\partial_2^\top \partial_2$.

Lemma 4.6.6. $\kappa(\partial_2^\top \boldsymbol{W} \partial_2) = O\left((\epsilon^{DA})^{-2} \operatorname{nnz}(\boldsymbol{A})^{15/2} \kappa(\boldsymbol{A}^\top \boldsymbol{A})\right)$.

Proof. The proof follows the same proof line in Section 4.5.6 for $\boldsymbol{W} = \boldsymbol{I}$. Here, we lose a factor $w_{\max}$ when we upper bound $\lambda_{\max}(\partial_2^\top \boldsymbol{W} \partial_2)$, and we lose a factor $\frac{w_{\max}}{w_{\min}}$ when we lower bound $\lambda_{\max}(\partial_2^\top \boldsymbol{W} \partial_2)$, where $w_{\max}$ is the maximum diagonal in $\boldsymbol{W}$ and $w_{\min}$ is the minimum non-zero diagonal in $\boldsymbol{W}$. By our setting, $w_{\max} = O(\alpha \operatorname{nnz}(\boldsymbol{A})^2)$ and $w_{\min} = \alpha$, where $\alpha = 2(\epsilon^{DA})^{-2}$. $\square$

Chapter 5

Hardness Results for More Structured Problems

This chapter extends the reduction from Chapter 4 to derive additional hardness results.

We begin by presenting the reduction algorithm from general linear equations to difference-average equations for completeness, and prove Theorem 1.4.3: LEA over $\mathcal{DA}$ is sparse-linear-equation complete. Next, using matrices in $\mathcal{DA}$ as our starting point, we introduce an even simpler form called *One-Or-Three* matrices, in which each row has either one or three non-zero entries (all of which are ± 1). We denote this matrix class by $\mathcal{OT}$. Building on this simplification, we extend the reduction of linear equations to the reduction of linear programs, leading to what we call *1-or-3* linear programs. We believe these programs serve as a strong foundation for further reductions to more structured LP problems. We remark that this result is unpublished.

For simplicity, throughout this chapter we focus on the reduction algorithms. Analyses such as error propagation and condition number are omitted.

5.1 Reducing General LE to Difference-Average LE

In this section, we prove Theorem 4.3.4: LEA over $\mathcal{DA}$ is sparse-linear-equation complete.

We show a nearly-linear time reduction from linear equation problems over matrices in $\mathcal{G}$ to linear equation problems over matrices in $\mathcal{DA}$. This reduction is implicit in Section 4 of [KZ17], as an intermediate step to reduce linear equation problems over matrices in $\mathcal{G}$ to matrices in a slight generalization of Laplacians. We explicitly separate the reduction step and simplify the proofs in [KZ17], which might be of independent interest.

Recall that a matrix in $\mathcal{G}$ has polynomially bounded integer entries and polynomially bounded condition numbers, and they do not have all-0 rows or all-0 columns; a matrix $\boldsymbol{A} \in \mathcal{DA}$ only has two types of rows such that if we multiply $\boldsymbol{A}$ to a vector $\boldsymbol{x}$, then the entries of $\boldsymbol{Ax}$ are in the form of either $\boldsymbol{x}(i) - \boldsymbol{x}(j)$ or $\boldsymbol{x}(i) + \boldsymbol{x}(j) - 2\boldsymbol{x}(k)$.

The reduction in [KZ17] has three steps:

1. Reduce linear equation problems over matrices in $\mathcal{G}$ to matrices in $\mathcal{G}_z$, a subset of $\mathcal{G}$ containing matrices with row sum 0. Given an instance $(\boldsymbol{G}, \boldsymbol{b})$ where $\boldsymbol{G} \in \mathcal{G}$, we construct a new instance $(\boldsymbol{G}', \boldsymbol{b})$ where $\boldsymbol{G}' = \begin{bmatrix} \boldsymbol{G} & -\boldsymbol{G}\boldsymbol{1} \end{bmatrix} \in \mathcal{G}_z$.

2. Reduce linear equation problems over matrices in $\mathcal{G}_z$ to matrices in $\mathcal{G}_{z,2}$, a subset of $\mathcal{G}_z$ containing matrices such that the sum of positive entries in each row is a

power of 2. Given an instance $(\boldsymbol{G}', \boldsymbol{b})$ where $\boldsymbol{G}' \in \mathcal{G}_z$, we construct a new instance $(\boldsymbol{G}'', \boldsymbol{b}'')$ where $\boldsymbol{G}'' = \begin{bmatrix} \boldsymbol{G}' & \boldsymbol{g} & -\boldsymbol{g} \\ \boldsymbol{0} & 1 & -1 \end{bmatrix} \in \mathcal{G}_{z,2}$ and $\boldsymbol{b}'' = \begin{bmatrix} \boldsymbol{b} \\ 0 \end{bmatrix}$.

3. Reduce linear equation problems over matrices in $\mathcal{G}_{z,2}$ to matrices in $\mathcal{DA}$.

The first two steps are proved in Section 7 of [KZ17], by standard tricks. We will focus on the third step.

In the rest of this section, to be consistent with the notations used in [KZ17], we use subscripts to denote entries of a matrix or a vector. Let $\boldsymbol{A}_i$ denote the ith row of a matrix $\boldsymbol{A}$, and $\boldsymbol{A}_{ij}$ denote the (i,j)th entry of $\boldsymbol{A}$. Let $\boldsymbol{x}_i$ denote the ith entry of a vector $\boldsymbol{x}$, and $\boldsymbol{x}_{i:j}$ denote the subvector of entries $\boldsymbol{x}_i, \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_j$. Moreover, we let $\|\boldsymbol{A}\|_{\max}$ be the maximum magnitude of the entries of $\boldsymbol{A}$.

Given an instance of linear equation problems over $\mathcal{G}_{z,2}$, our construction of an instance over $\mathcal{DA}$ does not depend on the error parameter. Therefore, we will describe our construction without the error parameters; then we will explain how to set the error parameter for the construction instance over $\mathcal{DA}$.

Let $(\boldsymbol{A}, \boldsymbol{c}^{\mathrm{A}})$ be an instance over $\mathcal{G}_{z,2}$. The idea is to write each equation in $(\boldsymbol{A}, \boldsymbol{c}^{\mathrm{A}})$ as a set of difference equations and average equations, via bitwise decomposition. Same as [KZ17], we first explain the idea by an example:

$$3\boldsymbol{x}_1 + 5\boldsymbol{x}_2 - \boldsymbol{x}_3 - 7\boldsymbol{x}_4 = 1. \tag{5.1}$$

We will manipulate the variables with positive coefficients and the variables with negative coefficients separately. Let us take the positive coefficients as an example. We pair all the variables with the odd positive coefficients and replace each pair with a new variable via an average equation. In this example, we pair $\boldsymbol{x}_1, \boldsymbol{x}_2$; we then create a new variable $\boldsymbol{x}_{12}$ and a new average equation

$$\boldsymbol{x}_1 + \boldsymbol{x}_2 - 2\boldsymbol{x}_{12} = 0.$$

Plugging this into Eq. (5.1), we get

$$2\boldsymbol{x}_1 + 4\boldsymbol{x}_2 + 2\boldsymbol{x}_{12} - \boldsymbol{x}_3 - 7\boldsymbol{x}_4 = 1.$$

We pull out a factor 2:

$$2(\boldsymbol{x}_1 + 2\boldsymbol{x}_2 + \boldsymbol{x}_{12}) - \boldsymbol{x}_3 - 7\boldsymbol{x}_4 = 1.$$

We repeat the pair-and-replace process to pair $\boldsymbol{x}_1, \boldsymbol{x}_{12}$ with a new variables $\boldsymbol{x}_{1,12}$ and a new average equation $\boldsymbol{x}_1 + \boldsymbol{x}_{12} - 2\boldsymbol{x}_{1,12} = 0$:

$$2(2\boldsymbol{x}_{1,12} + 2\boldsymbol{x}_2) - \boldsymbol{x}_3 - 7\boldsymbol{x}_4 = 1.$$

Pull out a factor 2:

$$4(\boldsymbol{x}_{1,12} + \boldsymbol{x}_2) - \boldsymbol{x}_3 - 7\boldsymbol{x}_4 = 1.$$

Repeat pair-and-replace with $\boldsymbol{x}_{1,12} + \boldsymbol{x}_2 - 2\boldsymbol{x}_{1,12,2} = 0$:

$$8\boldsymbol{x}_{1,12,2} - \boldsymbol{x}_3 - 7\boldsymbol{x}_4 = 1.$$

Similarly, we can use a sequence of average equations for the variables with odd coefficients, and the above equation becomes:

$$8\boldsymbol{x}_{1,12,2} - 8\boldsymbol{x}_{34} = 1.$$

where $\boldsymbol{x}_{34}$ is a new variable. The final equation is a difference equation.

The above reduction relies on the fact that the sum of all the coefficients is 0 and the sum of all the positive coefficients is a power of 2.

Pseudo-code of the reduction from linear equation problems over $\mathcal{G}_{z,2}$ to $\mathcal{DA}$ is shown in Algorithm 1.

Lemma 5.1.1. *Let $\boldsymbol{A} \in \mathcal{G}_{z,2}$, and let $\boldsymbol{B}$ be returned by running Algorithm* ReduceG$_{z,2}$toDA *on $\boldsymbol{A}$. Then,*

- $\mathrm{nnz}(\boldsymbol{B}) = O(\mathrm{nnz}(\boldsymbol{A}) \log \|\boldsymbol{A}\|_{\max})$;
- *both the dimensions of $\boldsymbol{B}$ are $O(\mathrm{nnz}(\boldsymbol{A}) \log \|\boldsymbol{A}\|_{\max})$.*

Proof. The second statement is implied by the first one.

To prove the first statement, we focus on Algorithm ReduceG$_{z,2}$toDA for a single equation in $\boldsymbol{A}_i$. The number of the while-iterations from line 16 to 27 is at most $\log \|\boldsymbol{A}\|_{\max}$. We observe that (1) in each iteration, the number of newly created variables and equations is at most the value of $\mathrm{nnz}(\mathcal{A}_i)$ at the beginning of the iteration; (2) once an auxiliary variable is added to $\mathcal{A}_i$, it must be replaced with a new auxiliary variable in the next iteration (if exists). Therefore, $\mathrm{nnz}(\mathcal{A}_i) = O(\mathrm{nnz}(\boldsymbol{A}_i))$ for every iteration. So, $\mathrm{nnz}(\boldsymbol{B}) = O(\mathrm{nnz}(\boldsymbol{A}) \log \|\boldsymbol{A}\|_{\max})$. □

Knowing that the first two steps of reduction only increase the problem size by a constant factor, we obtain the following result directly.

Corollary 5.1.2. *Let $\boldsymbol{A} \in \mathcal{G}$ and we reduce a polynomially bounded general* LE *$(\boldsymbol{A}, \boldsymbol{b})$ to a difference-average* LE *$(\boldsymbol{B}, \boldsymbol{c})$, where $\boldsymbol{B} \in \mathcal{DA}$. Then,*

$$\mathrm{nnz}(\boldsymbol{B}) = O(\mathrm{nnz}(\boldsymbol{A}) \log \|\boldsymbol{A}\|_{\max}).$$

5.2 Reducing Difference-Average LE to 1-or-3 LE

Recall that a matrix in $\mathcal{DA}$ only has two types of rows such that for any qth equation in $\boldsymbol{Ax} = \boldsymbol{b}$, it takes one of the following forms:

1. Average equation: $\boldsymbol{x}(q_1) + \boldsymbol{x}(q_2) = 2\boldsymbol{x}(q_3)$;
2. Difference equation: $\boldsymbol{x}(q_1) - \boldsymbol{x}(q_2) = b_q$.

In this section, we present a reduction algorithm to reduce $\mathcal{DA}$ to $\mathcal{OT}$, a class of matrices with a further simplified structure. Specifically, a matrix $\boldsymbol{A} \in \mathcal{OT}$ only has two types of rows such that for any qth equation in $\boldsymbol{Ax} = \boldsymbol{b}$, it has one of the following forms:

[1]Note that given two singleton multi-sets each containing a single equation, e.g. $\{\boldsymbol{a}_1^\top \boldsymbol{x} = c_1\}$ and $\{\boldsymbol{a}_2^\top \boldsymbol{x} = c_2\}$ where $\boldsymbol{a}_1, \boldsymbol{a}_2$ are vectors, we define $\{\boldsymbol{a}_1^\top \boldsymbol{x} = c_1\} + \{\boldsymbol{a}_2^\top \boldsymbol{x} = c_2\} = \{\boldsymbol{a}_1^\top \boldsymbol{x} + \boldsymbol{a}_2^\top \boldsymbol{x} = c_1 + c_2\}$ and we define $\{\boldsymbol{a}_1^\top \boldsymbol{x} = c_1\} \cup \{\boldsymbol{a}_2^\top \boldsymbol{x} = c_2\} = \{\boldsymbol{a}_1^\top \boldsymbol{x} = c_1, \boldsymbol{a}_2^\top \boldsymbol{x} = c_2\}$.

Algorithm 1: Reduce$\mathcal{G}_{z,2}$to$\mathcal{DA}$

```
Input: $(\boldsymbol{A}, \boldsymbol{c}^{A})$ where $\boldsymbol{A} \in \mathcal{G}_{z,2}$ is an $m \times n$ matrix, $\boldsymbol{c}^{A} \in \mathbb{R}^n$, and $\alpha > 0$.
Output: $(\boldsymbol{B}, \boldsymbol{c}^{B})$ where $\boldsymbol{B} \in \mathcal{DA}$ is an $m \times n$ matrix, and $\boldsymbol{c}^{B} \in \mathbb{R}^n$.
1  $\mathcal{X} \leftarrow \{\boldsymbol{u}_1, \dots, \boldsymbol{u}_n\}$ ;        // $\mathcal{DA}$ variables and index of new variables
2  Let $\boldsymbol{x}$ be the vector of variables corresponding to the set of variables $\mathcal{X}$
3  $t \leftarrow n + 1$
4  $\mathcal{A} \leftarrow \emptyset$, $\mathcal{B} \leftarrow \emptyset$ ;      // Multisets of main and $\mathcal{DA}$ auxiliary equations
     respectively
5  for each equation $1 \le i \le m$ in $\boldsymbol{A}$ do
6      if $\boldsymbol{A}_i$ is already a difference equation or an average equation then
7          $\mathcal{A} \leftarrow \mathcal{A} \cup \{\boldsymbol{A}_{ij_+}\boldsymbol{u}_{j_+} - \boldsymbol{A}_{ij_+}\boldsymbol{u}_{j_-} = \boldsymbol{c}_i\}$
8          $w_i \leftarrow \frac{\alpha}{\alpha+1}$
9      end
10     else
11         Let $\mathcal{I}^{+1} \leftarrow \{j : \boldsymbol{A}_{ij} > 0\}, \mathcal{I}^{-1} \leftarrow \{j : \boldsymbol{A}_{ij} < 0\}$
12         $\mathcal{A}_i \leftarrow \{\boldsymbol{A}_i\boldsymbol{u} = \boldsymbol{c}_i\}$
13         $\mathcal{B}_i \leftarrow \emptyset$
14         for $s = -1, +1$ do
15             $r \leftarrow 0$
16             while $\mathcal{A}_i$ is neither a difference equation nor an average equation do
17                 For each $j$, let $\widehat{\boldsymbol{A}}_{ij}$ be the coefficient of $\boldsymbol{u}_j$ in $\mathcal{A}_i$.
18                 $\mathcal{I}^s_{odd} \leftarrow \{j \in \mathcal{I}^s : \lfloor |\widehat{\boldsymbol{A}}_{ij}|/2^r \rfloor$ is odd$\}$
19                 Pair the indices of $\mathcal{I}^s_{odd}$ into disjoint pairs $(j_1, l_1), (j_2, l_2), \dots$
20                 for each pair of indices $(j_k, l_k)$ do
21                     $\mathcal{A}_i \leftarrow \mathcal{A}_i + s \cdot 2^r \{(2\boldsymbol{u}_t - (\boldsymbol{u}_{j_k} + \boldsymbol{u}_{l_k})) = 0\}$ [1]
22                     $\mathcal{B}_i \leftarrow \mathcal{B}_i \cup s \cdot 2^r \{(2\boldsymbol{u}_t - (\boldsymbol{u}_{j_k} + \boldsymbol{u}_{l_k})) = 0\}$
23                     $\mathcal{X} \leftarrow \mathcal{X} \cup \{\boldsymbol{u}_t\}$, update $\boldsymbol{x}$ accordingly
24                     $t \leftarrow t + 1$
25                     $r \leftarrow r + 1$
26                 end
27             end
28         end
29         $w_i \leftarrow \alpha\,|\mathcal{B}_i|$
30         $\mathcal{B} \leftarrow \mathcal{B} \cup w_i^{1/2} \cdot \mathcal{B}_i$
31         $\mathcal{A} \leftarrow \mathcal{A} \cup \mathcal{A}_i$.
32     end
33 end
34 return $\boldsymbol{B}, \boldsymbol{c}$ s.t. $\boldsymbol{Bx} = \boldsymbol{c}$ corresponds to the equations in $\mathcal{A} \cup \mathcal{B}$, on the variable
     set $\mathcal{X}$.
```

1. Three-variable equation: $\boldsymbol{x}(q_1) + \boldsymbol{x}(q_2) = \boldsymbol{x}(q_3)$;

2. Fixed-value equation: $\boldsymbol{x}(q) = 1$.

We independently reduce each average or difference equation into the $\mathcal{OT}$ form. The reduction process is presented step by step.

5.2.1 Reducing Average Equations

For each average equation $\boldsymbol{x}(q_1) + \boldsymbol{x}(q_2) = 2\boldsymbol{x}(q_3)$, we first introduce auxiliary variables $\boldsymbol{s}^{\text{avg}}$ and replace it with a constant number of equations:

$$\begin{aligned} \boldsymbol{x}(q_1) + \boldsymbol{x}(q_2) &= \boldsymbol{s}^{\text{avg}}(q_1), \\ \boldsymbol{x}(q_3) + \boldsymbol{s}^{\text{avg}}(q_2) &= \boldsymbol{s}^{\text{avg}}(q_1), \\ \boldsymbol{x}(q_3) + \boldsymbol{s}^{\text{avg}}(q_3) &= \boldsymbol{s}^{\text{avg}}(q_2), \\ \boldsymbol{s}^{\text{avg}}(q_3) &= 0. \end{aligned} \tag{Average reduction}$$

Compared to the $\mathcal{OT}$ form, it remains to reduce the last equation $\boldsymbol{s}^{\text{avg}}(q_3) = 0$ because its right-hand-side (RHS) is not 1. For this, we reduce it to

$$\begin{aligned} \boldsymbol{s}^{\text{avg}}(q_3) + \boldsymbol{s}^{\text{avg}}(q_4) &= \boldsymbol{s}^{\text{avg}}(q_5), \\ \boldsymbol{s}^{\text{avg}}(q_4) &= 1, \\ \boldsymbol{s}^{\text{avg}}(q_5) &= 1. \end{aligned} \tag{RHS reduction from 0 to 1}$$

Now, the reduced equations are in the form of $\mathcal{OT}$. And we can check that this reduction preserves equivalence with the original average equation.

5.2.2 Reducing Difference Equations

For each difference equation $\boldsymbol{x}(q_1) - \boldsymbol{x}(q_2) = b_q$, the major challenge lies in the reduction of the RHS b_q to 1. Without loss of generality, we can assume $b_q \geq 0$ because we can switch $\boldsymbol{x}(q_1), \boldsymbol{x}(q_2)$ otherwise. Moreover, we assume that entries of $\boldsymbol{b}$ are integers and can be represented with polylogarithmic bits.

We present the reduction in several steps.

1. First, we reduce the difference equation to the following two equations:

$$\begin{aligned} \boldsymbol{x}(q_1) + \boldsymbol{s}^{\text{diff}}(q) &= \boldsymbol{x}(q_2), \\ \boldsymbol{s}^{\text{diff}}(q) &= b_q. \end{aligned}$$

2. We then conduct bitwise decomposition of b_q:

$$b_q = \sum_{i=0}^{K} r_q^{(i)} 2^i, \quad \text{where } r_q^{(i)} \in \{0,1\} \text{ and } K = \log b_q.$$

   Using more auxiliary variables, we break down $\boldsymbol{s}^{\text{diff}}(q) = b_q$ into K equations:

$$\begin{aligned} \boldsymbol{s}^{\text{diff}}(q_0) &= r_q^{(0)} \in \{0,1\}, \\ \boldsymbol{s}^{\text{diff}}(q_i) &= \boldsymbol{s}^{\text{diff}}(q_{i-1}) + \boldsymbol{y}_q(i), \quad \forall i \in \{1,\ldots,K\}, \\ \boldsymbol{y}_q(i) &= r_q^{(i)} 2^i, \quad \forall i \in \{1,\ldots,K\}. \end{aligned}$$

3. In this step, we reduce $\boldsymbol{y}_q(i) = r_q^{(i)} 2^i$. If $r_q^{(i)} = 0$, then we skip. In the case of $r_q^{(i)} = 1$, we reduce it to a series of i equations:

$$\begin{aligned} &\boldsymbol{y}_q(i_0) = 1, \\ &\boldsymbol{y}_q(i_j) = 2\boldsymbol{y}_q(i_{j-1}), \quad \forall j \in \{1,\ldots,i\}. \end{aligned}$$

4. For each equation $\boldsymbol{y}_q(i_j) = 2\boldsymbol{y}_q(i_{j-1})$, we reduce it to
$$\begin{aligned}\boldsymbol{y}_q(i_{j-1,1}) + \boldsymbol{y}_q(i_{j-1,2}) &= \boldsymbol{y}_q(i_j),\\ \boldsymbol{y}_q(i_{j-1,1}) - \boldsymbol{y}_q(i_{j-1,2}) &= 0.\end{aligned}$$
Notice that the second equation is a difference equation with RHS being 0, applying the reduction in step 1 gives
$$\begin{aligned}\boldsymbol{y}_q(i_{j-1,1}) + \boldsymbol{y}_q(i_{j-1,3}) &= \boldsymbol{y}_q(i_{j-1,2}),\\ \boldsymbol{y}_q(i_{j-1,3}) &= 0.\end{aligned}$$

5. Finally, for all equation with only one variable and 0 RHS, we apply the reduction in (RHS reduction from 0 to 1).

Now, all the equations are reduced to the two types of equations in $\mathcal{OT}$: $\{\boldsymbol{x}(q_1) + \boldsymbol{x}(q_2) = \boldsymbol{x}(q_3), \boldsymbol{x}(q) = 1\}$.

Lemma 5.2.1. *Let $\boldsymbol{A} \in \mathcal{DA}$ and we reduce a difference-average* LE $(\boldsymbol{A}, \boldsymbol{b})$ *to a 1-or-3* LE $(\boldsymbol{B}, \boldsymbol{c})$*, where $\boldsymbol{B} \in \mathcal{OT}$ by applying the above reduction algorithm. Then,*
$$\mathrm{nnz}(\boldsymbol{B}) = O(\mathrm{nnz}(\boldsymbol{A}) \log^2 \|\boldsymbol{b}\|_{\max}).$$

Proof. We first have that each average equation is reduced to a constant number of equations. For each difference equation with RHS b_q, step 1, 4, and 5 only increase the number of equations by a constant factor, while step 2 and 3 increase the reduced number of equations by a factor of $O(\log b_q)$, respectively. Thus, each difference equation is reduced to $O(\log^2 b_q)$ equations.

To conclude, $\mathrm{nnz}(\boldsymbol{B}) = O(\mathrm{nnz}(\boldsymbol{A}) \log^2 \|\boldsymbol{b}\|_{\max})$. □

The following corollary is obtained by combining Corollary 5.1.2 and Lemma 5.2.1.

Corollary 5.2.2. *Let $\boldsymbol{A} \in \mathcal{G}$ and we reduce a polynomially bounded general* LE $(\boldsymbol{A}, \boldsymbol{b})$ *to a 1-or-3* LE $(\boldsymbol{B}, \boldsymbol{c})$*, where $\boldsymbol{B} \in \mathcal{OT}$ and entries of $\boldsymbol{c}$ are in $\{0, 1\}$. Then,*
$$\mathrm{nnz}(\boldsymbol{B}) = O(\mathrm{nnz}(\boldsymbol{A}) \log \|\boldsymbol{A}\|_{\max} \log^2 \|\boldsymbol{b}\|_{\max}).$$

5.3 Reducing General LP to (Simplified) 1-or-3 LP

In this section, we present a reduction algorithm from general linear programs to (simplified) 1-or-3 linear programs.

We start with an arbitrary polynomially-bounded LP
$$\{\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \quad \boldsymbol{x} \geq \boldsymbol{0}\}, \tag{General LP}$$
where entries of $\boldsymbol{A}, \boldsymbol{b}$ are polynomially bounded integers and $\|\boldsymbol{x}\|_1$ is also polynomially bounded. Our goal is to reduce it to 1-or-3 LP, which is in the form of
$$\{\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \quad \boldsymbol{0} \leq \boldsymbol{x} \leq \boldsymbol{1}\}, \quad \text{where } \boldsymbol{A} \in \mathcal{OT}. \tag{1-or-3 LP}$$

Compared to the reduction algorithms for general LE described in Section 5.1 and 5.2, reducing general linear programs (LPs) requires extra caution to preserve non-negativity constraints. Moreover, to reduce to 1-or-3 LP, we have to reduce and maintain the scale of $\boldsymbol{x}$ to be in the range of $[0,1]$.

Interestingly, the upper bound of variables $\boldsymbol{x} \leq \mathbf{1}$ in a 1-or-3 LP can be relaxed, leading to a further simplified problem:

$$\{\boldsymbol{A}\boldsymbol{x} = \mathbf{1}, \quad \boldsymbol{x} \geq \mathbf{0}\}, \tag{Simplified 1-or-3 LP}$$

The reduction algorithm for transforming a 1-or-3 LP into its simplified version is detailed in Section 5.3.4. The reduction algorithm from (General LP) to (1-or-3 LP) is presented in Section 5.3.1, 5.3.2 and 5.3.3, following a chain of reductions through several intermediate problems:

$$\text{General LP} \rightarrow \text{Scaled LP} \rightarrow \text{Difference-Average LP} \rightarrow \text{1-or-3 LP},$$

where a scaled LP takes the form

$$\{\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq \mathbf{0}, \|\boldsymbol{x}\|_{\max} \leq c\}, \quad \text{where } c \leq \frac{1}{2} \text{ is a constant,} \tag{Scaled LP}$$

and a difference-average LP is defined as

$$\{\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq \mathbf{0}, \|\boldsymbol{x}\|_{\max} \leq \frac{1}{2}\}, \quad \text{where } \boldsymbol{A} \in \mathcal{DA}. \tag{Difference-Average LP}$$

5.3.1 Reducing General LP to Scaled LP

Since (General LP) is polynomially bounded, let

$$B = \|\boldsymbol{x}\|_{\max} \leq \|\boldsymbol{x}\|_1 = \text{poly}(n).$$

And set k and C be integers such that

$$k = \lceil \log_2 \|\boldsymbol{b}\|_{\max} \rceil, \qquad C = 2^{\max\{k,\ 1+\lceil \log_2 B \rceil\}}.$$

Then we set

$$\widetilde{\boldsymbol{A}} = \frac{C\boldsymbol{A}}{2^k}, \quad \tilde{\boldsymbol{b}} = \frac{\boldsymbol{b}}{2^k}, \quad \tilde{\boldsymbol{x}} = \frac{\boldsymbol{x}}{C},$$

and obtain a scaled LP

$$\left\{\widetilde{\boldsymbol{A}}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}, \quad \tilde{\boldsymbol{x}} \geq \mathbf{0}, \quad \|\tilde{\boldsymbol{x}}\|_{\max} \leq \frac{B}{C}\right\}, \tag{5.2}$$

where we can notice that

- $\widetilde{\boldsymbol{A}} = \frac{C\boldsymbol{A}}{2^k}$ is a matrix with integer entries since $\frac{C}{2^k}$ is an integer;
- $\left\|\tilde{\boldsymbol{b}}\right\|_{\max} \leq 1$;
- $\|\tilde{\boldsymbol{x}}\|_{\max} \leq \frac{B}{C} \leq \frac{1}{2}$.

5.3.2 Reducing Scaled LP to Difference-Average LP

We apply the reduction algorithm described in Section 5.1. Recall the three steps to reduce a general LE to $\mathcal{DA}$, and we closely keep track of the non-negativity and the infinity norm of variables.

1. To reduce $\widetilde{\boldsymbol{A}}$ to a matrix with row sum 0, we set
$$\widetilde{\boldsymbol{A}}' = \begin{bmatrix} \widetilde{\boldsymbol{A}} & -\widetilde{\boldsymbol{A}}\mathbf{1} \end{bmatrix}, \quad \tilde{\boldsymbol{b}}' = \tilde{\boldsymbol{b}}, \quad \tilde{\boldsymbol{x}}' = \begin{bmatrix} \tilde{\boldsymbol{x}} \\ x_1^{\text{aux}} \end{bmatrix},$$
and we add an additional equality constraint
$$x_1^{\text{aux}} = 0.$$
Then we can check that $\tilde{\boldsymbol{x}}' \geq \mathbf{0}$ and $\|\tilde{\boldsymbol{x}}'\|_{\max} \leq 1/2$.

2. To reduce $\widetilde{\boldsymbol{A}}'$ to a matrix such that the sum of positive entries in each row is a power of 2, we set
$$\widetilde{\boldsymbol{A}}'' = \begin{bmatrix} \widetilde{\boldsymbol{A}}' & \boldsymbol{g} & -\boldsymbol{g} \\ \mathbf{0} & 1 & -1 \end{bmatrix}, \quad \tilde{\boldsymbol{b}}'' = \begin{bmatrix} \tilde{\boldsymbol{b}}' \\ 0 \end{bmatrix}, \quad \tilde{\boldsymbol{x}}'' = \begin{bmatrix} \tilde{\boldsymbol{x}}' \\ x_2^{\text{aux}} \\ x_3^{\text{aux}} \end{bmatrix},$$
and we add additional equality constraints
$$x_2^{\text{aux}} = 0, \quad x_3^{\text{aux}} = 0.$$
Again, we can check that $\tilde{\boldsymbol{x}}'' \geq \mathbf{0}$ and $\|\tilde{\boldsymbol{x}}''\|_{\max} \leq 1/2$.

3. To reduce $\widetilde{\boldsymbol{A}}''$ to a difference-average matrix, notice that each time we introduce a new variable, its value is the average of two old variables. Hence, new variables are also non-negative, and the infinity norm of the auxiliary variables will not exceed that of the original variables. The resulting problem is a difference-average LP, denoted as
$$\left\{ \bar{\boldsymbol{A}}\bar{\boldsymbol{x}} = \bar{\boldsymbol{b}}, \quad \bar{\boldsymbol{x}} \geq \mathbf{0}, \quad \|\bar{\boldsymbol{x}}\|_{\max} \leq \frac{1}{2} \right\}, \quad \text{where } \bar{\boldsymbol{A}} \in \mathcal{DA}. \tag{5.3}$$
In particular, $\bar{\boldsymbol{b}} = \begin{bmatrix} \tilde{\boldsymbol{b}}'' \\ \mathbf{0} \end{bmatrix}$, thus $\|\bar{\boldsymbol{b}}\|_{\max} \leq 1$.

5.3.3 Reducing Difference-Average LP to 1-or-3 LP

In this step, we extend the reduction algorithm from $\mathcal{DA}$ to $\mathcal{OT}$, as described in Section 5.2. To reduce average equations as shown in (Average reduction), we have
$$\boldsymbol{s}^{\text{avg}} \geq \mathbf{0}, \quad \|\boldsymbol{s}^{\text{avg}}\|_{\max} \leq 2\,\|\bar{\boldsymbol{x}}\|_{\max} \leq 1.$$

The reduction for difference equations needs to be slightly adjusted. The reason is that in Section 5.2, we assume that entries of the RHS vector are polynomially bounded integers, while here we have the RHS $\|\bar{\boldsymbol{b}}\|_{\max} \leq 1$. Therefore, for each difference equation $\bar{\boldsymbol{x}}(q_1) - \bar{\boldsymbol{x}}(q_2) = \bar{b}_q$, conducting bitwise decomposition of $\bar{b}_q$ gives

$$\bar{b}_q = \sum_{i=0}^{k} r_q^{(i)} 2^{-i}, \quad \text{where } r_q^{(i)} \in \{0,1\} \text{ and } k = \lceil \log \|\boldsymbol{b}\|_{\max} \rceil.$$

And to reduce $\boldsymbol{y}_q(i) = r_q^{(i)} 2^{-i}$ in the case of $r_q^{(i)} = 1$ and for $i \in \{1, \dots, k\}$, we reduce it to the following series of i equations:

$$\begin{aligned} &\boldsymbol{y}_q(i_0) = 1, \\ &\boldsymbol{y}_q(i_j) = \boldsymbol{y}_q(i_{j-1})/2, \quad \forall j \in \{1, \dots, i\}. \end{aligned}$$

And for each equations $\boldsymbol{y}_q(i_j) = \boldsymbol{y}_q(i_{j-1})/2$, we reduce it to

$$\begin{aligned} \boldsymbol{y}_q(i_{j-1,1}) + \boldsymbol{y}_q(i_{j-1,2}) &= 2\boldsymbol{y}_q(i_j), \\ \boldsymbol{y}_q(i_{j-1,2}) &= 0. \end{aligned}$$

Notice that the first equation is an average equation, which can be further reduced by applying (Average reduction). As the last step, we reduce one-variable equations with 0 right-hand side by applying (RHS reduction from 0 to 1). Again, we have the newly introduced variables to satisfy

$$\boldsymbol{s}^{\text{diff}}, \boldsymbol{y} \geq \mathbf{0} \quad \text{and} \quad \left\|\boldsymbol{s}^{\text{diff}}\right\|_{\max}, \|\boldsymbol{y}\|_{\max} \leq 1.$$

Putting it altogether, we reduce the difference-average LP (5.3) to a 1-or-3 LP in the form of (1-or-3 LP).

Lemma 5.3.1. *Let $\boldsymbol{A} \in \mathcal{G}$ be the coefficient matrix in* (General LP) *and let $\boldsymbol{B} \in \mathcal{OT}$ be the coefficient matrix in* (1-or-3 LP). *Then,*

$$\text{nnz}(\boldsymbol{B}) = O\left(\text{nnz}(\boldsymbol{A}) \log \|\boldsymbol{A}\|_{\max} \log^2 \|\boldsymbol{b}\|_{\max}\right).$$

Proof. It follows directly from Corollary 5.2.2. □

5.3.4 Reducing 1-or-3 LP to Simplified 1-or-3 LP

Actually, (1-or-3 LP) can be further simplified by dropping the upper bound of variables $\boldsymbol{x} \leq \mathbf{1}$, and achieving the following form:

$$\{\boldsymbol{A}\boldsymbol{x} = \mathbf{1}, \quad \boldsymbol{x} \geq \mathbf{0}\},$$

where equality constraints have one of the following forms:

1. $\boldsymbol{x}(q_1) + \boldsymbol{x}(q_2) + \boldsymbol{x}(q_3) = 1$;

2. $\boldsymbol{x}(q) = 1$.

For this, because of $\boldsymbol{x} \leq \mathbf{1}$, we can replace each equation in the form of $\boldsymbol{x}(q_1)+\boldsymbol{x}(q_2) = \boldsymbol{x}(q_3)$ with the following equations:

$$\begin{aligned} \boldsymbol{x}(q_1) + \boldsymbol{x}(q_2) + \boldsymbol{s}(q_1) &= 1, \\ \boldsymbol{x}(q_3) + \boldsymbol{s}(q_1) + \boldsymbol{s}(q_2) &= 1, \\ \boldsymbol{s}(q_1) &\geq 0, \\ \boldsymbol{s}(q_2) &= 0. \end{aligned} \qquad \text{(Dropping upper bound)}$$

And the equation $\boldsymbol{s}(q_2) = 0$ can be further reduced to

$$\begin{aligned} \boldsymbol{s}(q_2) + \boldsymbol{s}(q_3) + \boldsymbol{s}(q_4) &= 1, \\ \boldsymbol{s}(q_3) &= 1, \\ \boldsymbol{s}(q_4) &\geq 0. \end{aligned} \qquad \text{(Another RHS reduction from 0 to 1)}$$

The simplified 1-or-3 LP has a much more direct and intuitive structure, making it easier to relate to 3-SAT. Like 3-SAT, it serves as a promising starting point for proving the hardness of other problems, and for ultimately establishing a *sparse-linear-program complete* problem class, similar to the NP-complete problem class. This perspective aligns with the findings of [PW17], which also reduced general LPs to the simplified 1-or-3 LP but used a different reduction algorithm. Moreover, [PW17] extended this work by reducing the simplified 1-or-3 LP to LP relaxations of several combinatorial optimization problems, such as Set Cover, Set Packing, Maximum Satisfiability, and Maximum Independent Set. These results concluded that the LP relaxations of these combinatorial problems are sparse-linear-program complete.

Chapter 6

1-Laplacian Solver for Well-Shaped Simplicial Complexes

This chapter is based on [DZ23b]. We will prove Theorem 1.4.6 and 1.4.7. In the published version of [DZ23b], we further developed additional technical components that are not included in this thesis but may be of independent interest (e.g., sufficient conditions for 3-complexes that enable efficient r-hollowing computation; a nearly-linear time hollowing algorithm; a faster solver for up-Laplacian).

6.1 Prior Works

Cohen, Fasy, Miller, Nayyeri, Peng, and Walkington [Coh+14a] initiated the study of fast approximate solvers for 1-Laplacian linear equations. They designed a nearly-linear time approximate solver for simplicial complexes *with zero Betti numbers and known collapsing sequences*. Later, Black, Maxwell, Nayyeri, and Winkelman [Bla+22], and Black and Nayyeri [BN22] generalized this algorithm to subcomplexes of such a complex *with bounded first Betti numbers*. One concrete example studied in these papers is convex simplicial complexes that piecewise linearly triangulate a convex ball in $\mathbb{R}^3$, for which a collapsing sequence exists and can be computed in linear time [Chi67; Chi80]. However, deciding whether a simplicial complex has a collapsing sequence is NP-hard in general [Tan16]; computing the Betti numbers is as hard as computing the ranks of general $\{0, 1\}$ matrices [EP14]. In addition, 1-Laplacian systems for general simplicial complexes embedded in $\mathbb{R}^4$ are as hard to solve as general sparse linear equations [Din+22], for which the best-known algorithms need super-quadratic time [PV21; Nie22].

In addition to the specialized solvers for 1-Laplacian systems mentioned above, *Nested Dissection* can solve 1-Laplacian systems in quadratic time for simplicial complexes in $\mathbb{R}^3$ with additional geometric structures [Geo73; LRT79; MT90] such as bounded aspect ratios of individual tetrahedra. Furthermore, iterative methods such as *Preconditioned Conjugate Gradient* approximately solve 1-Laplacian systems in time $\widetilde{O}(n\sqrt{\kappa})$,[1] where n is the number of simplexes and κ is the condition number of the coefficient matrix.

Inspired by solvers that leverage both geometric structures and spectral properties, we develop efficient 1-Laplacian approximate solvers for well-shaped simplicial complexes embedded in $\mathbb{R}^3$ *without known collapsing sequences and with arbitrary Betti num-*

[1]We use $\widetilde{O}(\cdot)$ to hide polylog factors on the number of simplexes, the condition number, and the inverse of the error parameter.

bers. Our solver adapts the Incomplete Nested Dissection algorithm, proposed by Kyng, Peng, Schwieterman, and Zhang [Kyn+18] for solving linear equations in well-shaped 3-dimensional truss stiffness matrices. These matrices represent another generalization of graph Laplacians; however, they differ quite from the 1-Laplacians studied in this paper. A primary distinction is that the kernel of a truss stiffness matrix has an explicit and well-understood form, while computing a 1-Laplacian's kernel is as hard as that for a general matrix.

6.2 Motivations and Applications

In the past decades, combinatorial Laplacians have played a crucial role in the development of computational topology and topological data analysis in various domains, such as statistics [Jia+11; ODO13], graphics and imaging [Ma+11b; Ton+03b], brain networks [Lee+19], deep learning [Bro+17], signal processing [BS20], and cryo-electron microscope [YL17b]. We recommend readers consult accessible surveys [Ghr08b; Car09; EH10; Lim20] for more information.

Combinatorial Laplacians have their roots in the study of discrete Hodge decomposition [Eck44], which states that the kernel of the i-Laplacian $\boldsymbol{L}_i$ is isomorphic to the ith homology group of the simplicial complex. Among the many applications of combinatorial Laplacians, a central problem is determining the Betti numbers – the ranks of the homology groups – which are important topological invariants. Additionally, discrete Hodge decomposition allows for the extraction of meaningful information from data by decomposing them into three mutually orthogonal components: gradient (in the image of $\partial_i^\top$), curl (in the image of ∂_{i+1}), and harmonic (in the kernel of $\boldsymbol{L}_i$) components. For instance, the three components of edge flows in a graph capture the global trends, local circulations, and "noise".

The computation of both Betti numbers and discrete Hodge decomposition of higher-order flows can be achieved by solving systems of linear equations in combinatorial Laplacians [Fri96; Lim20]. The rank of a matrix $\boldsymbol{L}_i$ can be determined by solving a logarithmic number of linear equation systems in $\boldsymbol{L}_i$ [BV21]. The discrete Hodge decomposition can be calculated by solving least square problems involving boundary operators or combinatorial Laplacians, which in turn reduces to solving linear equations in these matrices.

Furthermore, an important question in numerical linear algebra concerns whether the nearly-linear time solvers for graph Laplacian linear equations can be generalized to larger classes of linear equations. Researchers have achieved success with elliptic finite element systems [BHV08b], connection Laplacians [Kyn+16], directed Laplacians [Coh+17a; Coh+18], well-shaped truss stiffness matrices [DS07; ST08; Kyn+18]. It would be intriguing to determine what structures of linear equations facilitate faster solvers. Another theoretically compelling reason for developing efficient solvers for 1-Laplacians stems from the "equivalence" of time complexity between solving 1-Laplacian systems and general sparse systems of linear equations [Din+22]. If one can solve all 1-Laplacian systems in time $\widetilde{O}((\text{number of simplexes})^c)$ where $c \geq 1$ is a constant, then one can solve all general systems of linear equations in time $\widetilde{O}((\text{number of non-zeros})^c)$.

6.3 Notations and Preliminaries

Aspect Ratio. The *aspect ratio* of a set $S \subset \mathbb{R}^3$ is the radius of the smallest ball containing S divided by the radius of the largest ball contained in S. The aspect ratio of S is always greater than or equal to 1. We say a simplex σ is *stable* if it has $O(1)$ aspect ratio and $\Theta(1)$ weight. Miller and Thurston proved the following lemma. As a corollary, the numbers of the vertices, the edges, the triangles, and the tetrahedra of a 3-complex $\mathcal{K}$ that is composed of stable tetrahedra are all equal up to a constant factor.

Lemma 6.3.1 (Lemma 4.1 of [MT90])**.** *Let $\mathcal{K}$ be a 3-complex in $\mathbb{R}^3$ in which each tetrahedron has $O(1)$ aspect ratio. Then, each vertex of $\mathcal{K}$ is contained in at most $O(1)$ tetrahedra.*

r-Hollowings. Let $\mathcal{K}$ be a pure 3-complex with n simplexes. A set of triangles $\hat{\mathbf{\Delta}}_1, \ldots, \hat{\mathbf{\Delta}}_k$ form a *triangle path* of length $k-1$ if for any $1 \le i \le k-1$, $\hat{\mathbf{\Delta}}_i$ and $\hat{\mathbf{\Delta}}_{i+1}$ share an edge. The *triangle distance* between two triangles $\mathbf{\Delta}_1$ and $\mathbf{\Delta}_2$ is the shortest triangle path length between $\mathbf{\Delta}_1$ and $\mathbf{\Delta}_2$. The *triangle diameter* of $\mathcal{K}$ is the longest triangle distance between any two triangles. A *spherical shell* is $\{\boldsymbol{x} \in \mathbb{R}^3 : R_1 \le \|\boldsymbol{x}\|_2 \le R_2\}$ where $R_1 < R_2$. If $\mathcal{K}$ triangulates a spherical shell, we define the *shell width* to be the shortest triangle distance between any two triangles where one is on the outer sphere and one is on the inner sphere.

Definition 6.3.2 (r-Hollowing)**.** Let $\mathcal{K}$ be a 3-complex with n simplexes, and let $r = o(n)$ be a positive number. We divide $\mathcal{K}$ into $O(n/r)$ *regions* each of $O(r)$ simplexes, $O(r^{2/3})$ *boundary simplexes*, and $O(r^{2/3})$ exterior simplexes. Non-boundary simplexes are called *interior simplexes*.[2] Interior simplexes from different regions do not share any subsimplexes. The boundary of each region triangulates a spherical shell in $\mathbb{R}^3$ and has triangle diameter $O(r^{1/3})$ and shell width at least 5. The union of all boundary simplexes of each region is referred to as an *r-hollowing* of $\mathcal{K}$.

Figure 6.1 illustrates an example of r-hollowing by showing a cross-section of a 3-complex. The left figure presents a cross-section of a 3-complex $\mathcal{K}$ with two holes inside, depicted as two empty discs. The outlines in black represent the "exterior simplexes" of $\mathcal{K}$. On the right-hand side of Figure 6.1, the gray area represents an r-hollowing of $\mathcal{K}$. It divides $\mathcal{K}$ into 4 balanced regions $\{(1), (2), (3), (4)\}$, each indicated by the area inside the red, blue, green, or orange outline, respectively. Region (1) includes the smaller hole, while the larger hole "intersects" regions (3) and (4). The unshaded area inside the squares corresponds to "interior simplexes," which encompasses the exterior simplexes of the smaller hole and the left half of the larger hole. In contrast, the gray area represents "boundary simplexes," including the exterior simplexes of the right half of the larger hole. The shell width is indicated by arrows.

In the full version, [DZ23b] examines sufficient conditions for 3-complexes that enable us to compute an r-hollowing in nearly-linear time. Here, we focus on the case where $\mathcal{K}$ is already equipped with a known r-hollowing.

[2] We would like to emphasize that "exterior simplex" is defined for any 3-complex, while "boundary simplex" and "interior simplex" are defined for r-hollowing. Although boundary and interior simplexes are mutually exclusive, an exterior simplex can be either a boundary or an interior simplex for a region of an r-hollowing.

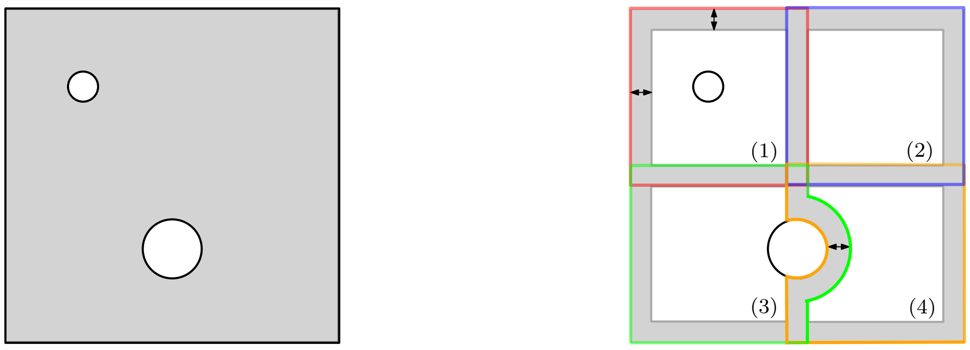

Figure 6.1: A cross-sectional illustration of an r-hollowing.

Algorithms for Solving Linear Equations. Our algorithm combines two of the most important tools for solving linear systems: Nested Dissection and preconditioning.

Theorem 6.3.3 (Nested Dissection [MT90])**.** *Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ be a symmetric matrix defined on a simple tetrahedral mesh. A Cholesky factorization $\boldsymbol{A} = \boldsymbol{P}\boldsymbol{L}\boldsymbol{L}^\top\boldsymbol{P}^\top$ can be computed in time $O(n^2)$, in which $\boldsymbol{P}$ is a permutation matrix and $\boldsymbol{L}$ is a lower triangular matrix with $O(n^{4/3})$ non-zero entries. As a result, a linear system in $\boldsymbol{A}$ can be solved in time $O(n^2)$ by Gaussian Elimination.*

Theorem 6.3.4 (Preconditioned Conjugate Gradient [Axe85])**.** *Let $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$ be two symmetric PSD matrices, and let $\boldsymbol{b} \in \mathbb{R}^n$. Each iteration of Preconditioned Conjugate Gradient multiplies one vector with $\boldsymbol{A}$, solves one system of linear equations in $\boldsymbol{B}$, and performs a constant number of vector operations. For any $\epsilon > 0$, the algorithm outputs an $\boldsymbol{x}$ satisfying $\left\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{\Pi}_{Im(\boldsymbol{A})}\boldsymbol{b}\right\|_2 \le \epsilon \left\|\boldsymbol{\Pi}_{Im(\boldsymbol{A})}\boldsymbol{b}\right\|_2$ in $O(\sqrt{\kappa}\log(\kappa/\epsilon))$ such iterations, where $\boldsymbol{\Pi}_{Im(\boldsymbol{A})}$ is the orthogonal projection matrix onto the image of $\boldsymbol{A}$ and $\kappa = \kappa(\boldsymbol{A}, \boldsymbol{B})$.*

6.4 Main Results

We formally state our main results as follows.

Theorem 6.4.1 (A Pure 3-Complex)**.** *Let $\mathcal{K}$ be a pure 3-complex embedded in $\mathbb{R}^3$ consisting of n stable simplexes and with a known r-hollowing. Let $\boldsymbol{L}_1$ be the 1-Laplacian operator of $\mathcal{K}$, and let $\boldsymbol{\Pi}_1$ be the orthogonal projection matrix onto the image of $\boldsymbol{L}_1$. For any vector $\boldsymbol{b}$ and $\epsilon > 0$, we can find a solution $\tilde{\boldsymbol{x}}$ such that $\|\boldsymbol{L}_1\tilde{\boldsymbol{x}} - \boldsymbol{\Pi}_1\boldsymbol{b}\|_2 \le \epsilon \|\boldsymbol{\Pi}_1\boldsymbol{b}\|_2$ in time*

$$O\left(nr + n^{4/3}r^{5/18}\log(n/\epsilon) + n^2 r^{-2/3}\right).$$

We will overview our algorithm for Theorem 6.4.1 in Section 6.5 and provide detailed proofs in Section 6.6, 6.7, 6.8, and 6.9.

Theorem 6.4.2 (A Union of Pure 3-Complexes)**.** *Let $\mathcal{U}$ be a union of pure 3-complexes glued together by identifying certain subsets of their exterior simplexes. Assume that each 3-complex chunk is embedded in $\mathbb{R}^3$, contains n_i stable simplexes, and has a known $\Theta(n_i^{3/5})$-hollowing. Let $\boldsymbol{L}_1$ be the 1-Laplacian operator of $\mathcal{U}$, and let $\boldsymbol{\Pi}_1$ be the orthogonal projection matrix onto the image of $\boldsymbol{L}_1$. For any vector $\boldsymbol{b}$ and $\epsilon > 0$, we can find a solution $\tilde{\boldsymbol{x}}$ such that $\|\boldsymbol{L}_1\tilde{\boldsymbol{x}} - \boldsymbol{\Pi}_1\boldsymbol{b}\|_2 \le \epsilon \|\boldsymbol{\Pi}_1\boldsymbol{b}\|_2$ in time*

$$\widetilde{O}(n^{8/5} + n^{3/10}k^2 + k^3),$$

where n is the number of simplexes in $\mathcal{U}$ and k is the number of exterior simplexes shared by more than one chunk.

We will defer our proof of Theorem 6.4.2 to Section 6.10.

6.5 Algorithm Overview

Cohen, Fasy, Miller, Nayyeri, Peng, and Walkington [Coh+14a] observed that

$$\boldsymbol{L}_1^\dagger = \left(\boldsymbol{L}_1^{\text{down}}\right)^\dagger + \left(\boldsymbol{L}_1^{\text{up}}\right)^\dagger,$$

where $\boldsymbol{L}_1^{\text{down}} = \partial_1^\top \boldsymbol{W}_0 \partial_1$ is the down-Laplacian and $\boldsymbol{L}_1^{\text{up}} = \partial_2 \boldsymbol{W}_2 \partial_2^\top$ is the up-Laplacian.

The orthogonal projection matrices onto $\text{Im}(\partial_1^\top)$ and $\text{Im}(\partial_2)$ are:

$$\boldsymbol{\Pi}_1^{\text{down}} \stackrel{\text{def}}{=} \partial_1^\top (\partial_1 \partial_1^\top)^\dagger \partial_1, \qquad \boldsymbol{\Pi}_1^{\text{up}} \stackrel{\text{def}}{=} \partial_2 (\partial_2^\top \partial_2)^\dagger \partial_2^\top.$$

Lemma 6.5.1 (Lemma 4.1 of [Coh+14a])**.** *Let $\boldsymbol{b}$ be a vector. Consider the systems of linear equations: $\boldsymbol{L}_1 \boldsymbol{x} = \boldsymbol{\Pi}_1 \boldsymbol{b}, \boldsymbol{L}_1^{up} \boldsymbol{x}^{up} = \boldsymbol{\Pi}_1^{up} \boldsymbol{b}, \boldsymbol{L}_1^{down} \boldsymbol{x}^{down} = \boldsymbol{\Pi}_1^{down} \boldsymbol{b}$. Then, $\boldsymbol{x} = \boldsymbol{\Pi}_1^{up} \boldsymbol{x}^{up} + \boldsymbol{\Pi}_1^{down} \boldsymbol{x}^{down}$.*

Lemma 6.5.1 implies that four operators are needed to approximate $\boldsymbol{L}_1^\dagger$: (1) an approximate projection operator $\widetilde{\boldsymbol{\Pi}}_1^{\text{down}} \approx \boldsymbol{\Pi}_1^{\text{down}}$, (2) an approximate projection operator $\widetilde{\boldsymbol{\Pi}}_1^{\text{up}} \approx \boldsymbol{\Pi}_1^{\text{up}}$, (3) a down-Laplacian solver $\boldsymbol{Z}_1^{\text{down}}$ such that $\boldsymbol{L}_1^{\text{down}} \boldsymbol{Z}_1^{\text{down}} \boldsymbol{b} \approx \boldsymbol{b}$ for any $\boldsymbol{b} \in \text{Im}(\boldsymbol{L}_1^{\text{up}})$, and (4) an up-Laplacian solver $\boldsymbol{Z}_1^{\text{up}}$ such that $\boldsymbol{L}_1^{\text{up}} \boldsymbol{Z}_1^{\text{up}} \boldsymbol{b} \approx \boldsymbol{b}$ for any $\boldsymbol{b} \in \text{Im}(\boldsymbol{L}_1^{\text{up}})$.

6.5.1 Down-Laplacian

We will apply the same approximate orthogonal projection $\widetilde{\boldsymbol{\Pi}}_1^{\text{down}}$ given in [Coh+14a], which does not depend on Betti numbers. Our solver for the first down-Laplacian is a slight modification of the one in [Coh+14a] to incorporate simplex weights. We state the two lemmas but defer their proofs to Section 6.6.

Lemma 6.5.2 (Down-Projection Operator, Lemma 3.2 of [Coh+14a])**.** *Let $\mathcal{K}$ be a 3-complex with n simplexes. For any $\epsilon > 0$, there exists a linear operator $\widetilde{\boldsymbol{\Pi}}_1^{down}$, computable in nearly-linear time, such that*

$$(1-\epsilon)\boldsymbol{\Pi}_1^{down} \preccurlyeq \widetilde{\boldsymbol{\Pi}}_1^{down}(\epsilon) \preccurlyeq \boldsymbol{\Pi}_1^{down}.$$

Lemma 6.5.3 (Down-Laplacian Solver)**.** *Let $\mathcal{K}$ be a weighted simplicial complex, and let $\boldsymbol{b} \in Im(\boldsymbol{L}_1^{down})$. There exists an operator $\boldsymbol{Z}_1^{down}$ such that $\boldsymbol{L}_1^{down} \boldsymbol{Z}_1^{down} \boldsymbol{b} = \boldsymbol{b}$. In addition, we can compute $\boldsymbol{Z}_1^{down} \boldsymbol{b}$ in linear time.*

6.5.2 Solver for Up-Laplacian

One of our primary technical contributions is the development of an efficient solver for the up-Laplacian system, stated in Lemma 6.5.4. We will describe the key idea behind our solver in this section.

Lemma 6.5.4 (Up-Laplacian Solver). *Let $\mathcal{K}$ be a pure 3-complex embedded in $\mathbb{R}^3$ and composed of n stable simplexes. Suppose we are given an r-hollowing for $\mathcal{K}$ where $r = o(n)$. Then for any $\epsilon > 0$, there exists an operator $\boldsymbol{Z}_1^{up}$ such that*

$$\forall \boldsymbol{b} \in Im(\boldsymbol{L}_1^{up}), \quad \left\|\boldsymbol{L}_1^{up}\boldsymbol{Z}_1^{up}\boldsymbol{b} - \boldsymbol{b}\right\|_2 \leq \epsilon \left\|\boldsymbol{b}\right\|_2.$$

In addition, $\boldsymbol{Z}_1^{up}\boldsymbol{b}$ can be computed in time $O\left(nr + n^{4/3}r^{5/18}\log(n/\epsilon) + n^2r^{-2/3}\right)$.

We remark that Lemma 6.5.4 can be improved to $\widetilde{O}(n^{3/2})$ by using a slightly different r-hollowing (proved in Section 10 of [DZ23b]), which might be of independent interest. Since the bottleneck of our solver for 1-Laplacians is from the projection for up 1-Laplaicans, we use the same r-hollowing here.

The given r-hollowing suggests a partition of the edges in $\mathcal{K}$ into $F \cup C$. We will explain the concrete partition shortly. We have the following matrix identity:

$$\boldsymbol{L}_1^{\text{up}} = \begin{bmatrix} \boldsymbol{I} & \\ \boldsymbol{L}_1^{\text{up}}(C,F)\boldsymbol{L}_1^{\text{up}}(F,F)^\dagger & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{L}_1^{\text{up}}(F,F) & \\ & \text{Sc}(\boldsymbol{L}_1^{\text{up}})_C \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{L}_1^{\text{up}}(F,F)^\dagger \boldsymbol{L}_1^{\text{up}}(F,C) \\ & \boldsymbol{I} \end{bmatrix},$$

where

$$\text{Sc}(\boldsymbol{L}_1^{\text{up}})_C = \boldsymbol{L}_1^{\text{up}}(C,C) - \boldsymbol{L}_1^{\text{up}}(C,F)\boldsymbol{L}_1^{\text{up}}(F,F)^\dagger \boldsymbol{L}_1^{\text{up}}(F,C).$$

The following Lemma 6.5.5 reduces (approximately) solving a system in $\boldsymbol{L}_1^{\text{up}}$ to (approximately) solving two systems in $\boldsymbol{L}_1^{\text{up}}(F,F)$ and one system in $\text{Sc}(\boldsymbol{L}_1^{\text{up}})_C$, which is proved in Appendix C. It is worth noting that Lemma 6.5.5 holds if we replace $\boldsymbol{L}_1^{\text{up}}$ with an arbitrary symmetric PSD matrix, and we will apply it or its variants for different PSD matrices in our solvers. To avoid introducing additional notations, we state the lemma below in terms of $\boldsymbol{L}_1^{\text{up}}$.

Lemma 6.5.5. *Suppose we have two operators:*

1. UpLapFSolver$(\cdot)$ *such that given any $\boldsymbol{b} \in Im(\boldsymbol{L}_1^{up}(F,F))$,* UpLapFSolver$(\boldsymbol{b})$ *returns a vector $\boldsymbol{x}$ satisfying $\boldsymbol{L}_1^{up}(F,F)\boldsymbol{x} = \boldsymbol{b}$;*

2. SchurSolver$(\cdot,\cdot)$ *such that for any $\boldsymbol{h} \in Im(Sc(\boldsymbol{L}_1^{up})_C)$ and $\delta > 0$,* SchurSolver$(\boldsymbol{h},\delta)$ *returns $\tilde{\boldsymbol{x}}$ satisfying $\left\|Sc(\boldsymbol{L}_1^{up})_C\tilde{\boldsymbol{x}} - \boldsymbol{h}\right\|_2 \leq \delta \left\|\boldsymbol{h}\right\|_2$.*

Given any $\boldsymbol{b} = \begin{bmatrix} \boldsymbol{b}_F \\ \boldsymbol{b}_C \end{bmatrix} \in Im(\boldsymbol{L}_1^{up})$ and any $\epsilon > 0$, let

$$\begin{aligned} \boldsymbol{h} &= \boldsymbol{b}_C - \boldsymbol{L}_1^{up}(F,F) \cdot \text{UpLapFSolver}(\boldsymbol{b}_F), \\ \tilde{\boldsymbol{x}}_C &= \text{SchurSolver}(\boldsymbol{h},\delta), \\ \tilde{\boldsymbol{x}}_F &= \text{UpLapFSolver}\left(\boldsymbol{b}_F - \boldsymbol{L}_1^{up}(F,C)\tilde{\boldsymbol{x}}_C\right), \end{aligned} \tag{6.1}$$

where $\delta \leq \frac{\epsilon}{1+\left\|\boldsymbol{L}_1^{up}(F,F)\boldsymbol{L}_1^{up}(F,F)^\dagger\right\|_2}$. Then,

$$\left\|\boldsymbol{L}_1^{up}\tilde{\boldsymbol{x}} - \boldsymbol{b}\right\|_2 \leq \epsilon \left\|\boldsymbol{b}\right\|_2,$$

where $\tilde{\boldsymbol{x}} = \begin{bmatrix} \tilde{\boldsymbol{x}}_F \\ \tilde{\boldsymbol{x}}_C \end{bmatrix}$. Let $m_F = |F|$ and $m_C = |C|$, and let UpLapFSolver *have runtime $t_1(m_F)$ and* SchurSolver *have runtime $t_2(m_C)$. Then, we can compute $\tilde{\boldsymbol{x}}$ in time $O(t_1(m_F) + t_2(m_C) + m_F + m_C)$.*

The proof of Lemma 6.5.5 can be found in Appendix C.

Partitioning the Edges

As suggested by Lemma 6.5.5, we want to partition the edges of $\mathcal{K}$ into $F \cup C$ so that both systems in $\boldsymbol{L}_1^{\text{up}}(F,F)$ and the Schur complement $\text{Sc}(\boldsymbol{L}_1^{\text{up}})_C$ can be efficiently solved. The given $O(n^{3/5})$-hollowing divides $\mathcal{K}$ into *"disjoint" and balanced* regions with *small* boundary. Let F be the set of the interior edges of the regions and C be the set of the boundary edges.

We first show that the interiors of different regions are "disjoint" in the sense that $\boldsymbol{L}_1^{\text{up}}(F,F)$ is a block diagonal matrix where each diagonal block corresponds to the interior of a region. We can write $\boldsymbol{L}_1^{\text{up}}$ as the sum of rank-1 matrices that each corresponds to a triangle in $\mathcal{K}$:

$$\boldsymbol{L}_1^{\text{up}} = \partial_2 \boldsymbol{W}_2 \partial_2^\top = \sum_{\sigma:\text{triangle in } \mathcal{K}} \boldsymbol{W}_2(\sigma,\sigma) \cdot \partial_2(:,\sigma)\partial_2(:,\sigma)^\top.$$

For any two edges e_1, e_2, $\boldsymbol{L}_1^{\text{up}}(e_1,e_2) = 0$ if and only if no triangle in $\mathcal{K}$ contains both e_1, e_2. By our definition of r-hollowing in Definition 6.3.2, for different regions R_1, R_2 of $\mathcal{K}$ w.r.t. an r-hollowing, no triangle contains both an edge from R_1 and an edge from R_2.

In addition, the following lemma shows that the boundaries of the regions well approximate the Schur complement onto the boundaries. The proof is in Section 6.7.2.

Lemma 6.5.6 (Spectral Bounds for r-Hollowing)**.** *Let $\mathcal{K}$ be a pure 3-complex embedded in $\mathbb{R}^3$ composed of stable simplexes. Let $\mathcal{T}$ be an r-hollowing of $\mathcal{K}$, and let C be the edges of $\mathcal{T}$. Then,*

$$\boldsymbol{L}_{1,\mathcal{T}}^{up} \preccurlyeq Sc(\boldsymbol{L}_1^{up})_C \preccurlyeq O(r)\boldsymbol{L}_{1,\mathcal{T}}^{up}.$$

Proof of Lemma 6.5.4 for Up-Laplacian Solver

Algorithm 2 sketches a pseudo-code for our up-Laplacian solver.

```
Algorithm 2: UpLapSolver($\mathcal{K}, \mathcal{T}, \boldsymbol{b}, \epsilon$)
  Data: A pure 3-complex $\mathcal{K}$ of $n$ stable simplexes with up-Laplacian $\boldsymbol{L}_1^{\text{up}}$, an
        $r$-hollowing $\mathcal{T}$, a vector $\boldsymbol{b} \in \text{Im}(\boldsymbol{L}_1^{\text{up}})$, an error parameter $\epsilon > 0$
  Result: An approximate solution $\tilde{\boldsymbol{x}}$ such that $\|\boldsymbol{L}_1^{\text{up}}\tilde{\boldsymbol{x}} - \boldsymbol{b}\|_2 \le \epsilon \|\boldsymbol{b}\|_2$
1 $F \leftarrow$ the interior edges of regions of $\mathcal{K}$ w.r.t. $\mathcal{T}$;   $C \leftarrow$ the boundary edges of
   regions.
2 UpLapFSolver($\cdot$)$\leftarrow$ a solver by Nested Dissection that satisfies the
   requirement in Lemma 6.5.5.
3 SchurSolver($\cdot,\cdot$)$\leftarrow$ a solver by Preconditioned Conjugate Gradient with the
   preconditioner being the up-Laplacian of $\mathcal{T}$ that satisfies the requirement in
   Lemma 6.5.5.
4 $\tilde{\boldsymbol{x}} \leftarrow$ computed by Equation (6.1)
5 return solution $\tilde{\boldsymbol{x}}$
```

By Lemma 6.5.5, the $\tilde{\boldsymbol{x}}$ returned by Algorithm 2 satisfies $\|\boldsymbol{L}_1^{\text{up}}\tilde{\boldsymbol{x}} - \boldsymbol{b}\|_2 \le \epsilon \|\boldsymbol{b}\|_2$. To bound the runtime of Algorithm 2, we need the following lemmas for lines 2 and 3.

Lemma 6.5.7 (Solver for the "F" Part)**.** *Let $\mathcal{K}$ be a pure 3-complex embedded in $\mathbb{R}^3$ and composed of n stable simplexes. Let $\mathcal{T}$ be an r-hollowing of $\mathcal{K}$, and let F be the set of interior edges in each region of $\mathcal{K}$ w.r.t. $\mathcal{T}$. Then, with a pre-processing time $O(nr)$, there exists a solver* UpLapFSolver$(\cdot)$ *such that given any $\boldsymbol{b}_F \in im(\boldsymbol{L}_1^{up}(F,F))$,* UpLapFSolver$(\boldsymbol{b}_F)$ *returns an $\boldsymbol{x}_F$ such that $\boldsymbol{L}_1^{up}(F,F)\boldsymbol{x}_F = \boldsymbol{b}_F$ in time $O(nr^{1/3})$.*

By our choice of F, the matrix $\boldsymbol{L}_1^{\mathrm{up}}(F,F)$ can be written as a block diagonal matrix where each block corresponds to a region of $\mathcal{K}$ w.r.t. the r-hollowing $\mathcal{T}$. Since each region is a 3-complex in which every tetrahedron has an aspect ratio $O(1)$, we can construct the solver UpLapFSolver by Nested Dissection [MT90]. However, since each row or column of $\boldsymbol{L}_1^{\mathrm{up}}(F,F)$ corresponds to an edge in $\mathcal{K}$, we need to turn the good *vertex separators* in [MT90] into good *edge separators* for regions of $\mathcal{K}$. We prove Lemma 6.5.7 in Section 6.7.1.

Lemma 6.5.8 (Solver for the Schur Complement)**.** *Let $\mathcal{K}$ be a pure 3-complex embedded in $\mathbb{R}^3$ and composed of n stable simplexes. Let $\mathcal{T}$ be an r-hollowing of $\mathcal{K}$, and let C be the set of boundary edges of each region of $\mathcal{K}$ w.r.t. $\mathcal{T}$. Then, with a pre-processing time $O(nr + n^2r^{-2/3})$ there exists a solver* SchurSolver$(\cdot,\cdot)$ *such that for any $\boldsymbol{h} \in Im(Sc(\boldsymbol{L}_1^{up})_C)$ and $\delta > 0$,* SchurSolver$(\boldsymbol{h},\delta)$ *returns an $\tilde{\boldsymbol{x}}_C$ such that $\|Sc(\boldsymbol{L}_1^{up})_C\tilde{\boldsymbol{x}}_C - \boldsymbol{h}\|_2 \le \delta\|\boldsymbol{h}\|_2$ in time $\widetilde{O}\left(nr^{5/6} + n^{4/3}r^{5/18}\right)$.*

Our solver SchurSolver is based on the Preconditioned Conjugate Gradient (PCG) with the preconditioner $\boldsymbol{L}_{1,\mathcal{T}}^{\mathrm{up}}$, the up-Laplacian operator of $\mathcal{T}$. By Theorem 6.3.4 and Lemma 6.5.6, the number of PCG iterations is $\widetilde{O}(\sqrt{r})$. In each PCG iteration, we solve the system in $\boldsymbol{L}_{1,\mathcal{T}}^{\mathrm{up}}$ via Nested Dissection. We prove Lemma 6.5.8 in Section 6.7.2.

Given the above lemmas, we prove Lemma 6.5.4.

Proof of Lemma 6.5.4. The correctness of Algorithm 2 is by Lemma 6.5.5. By Lemma 6.5.7 and 6.5.8, the total runtime of the algorithm is

$$\widetilde{O}\left(nr^{5/6} + n^{4/3}r^{5/18} + nr + n^2r^{-2/3}\right).$$

□

6.5.3 Projection onto the Image of Up-Laplacian

As the first Betti number of $\mathcal{K}$ can be arbitrary, the approximate projection operators for the up 1-Laplacian provided in [Coh+14a; Bla+22; BN22] are not applicable here. Our approximate projection operator follows a similar approach to our up-Laplacian solver, which is based on an incomplete Nested Dissection for *triangles*, instead of edges.

Lemma 6.5.9 (Up-Projection Operator)**.** *Let $\mathcal{K}$ be a pure 3-complex embedded in $\mathbb{R}^3$ and composed of n stable simplexes. Suppose we are given an r-hollowing for $\mathcal{K}$. Then, for any $\epsilon > 0$, there exists an operator $\widetilde{\boldsymbol{\Pi}}_1^{up}$ such that for any $\boldsymbol{b}$, $\widetilde{\boldsymbol{\Pi}}_1^{up}\boldsymbol{b}$ is in the image of $\boldsymbol{L}_1^{up}$, and*

$$\left\|\widetilde{\boldsymbol{\Pi}}_1^{up}\boldsymbol{b} - \boldsymbol{\Pi}_1^{up}\boldsymbol{b}\right\|_2 \le \epsilon\left\|\boldsymbol{\Pi}_1^{up}\boldsymbol{b}\right\|_2.$$

In addition, $\widetilde{\boldsymbol{\Pi}}_1^{up}\boldsymbol{b}$ can be computed in time $O\left(nr + n^{4/3}r^{5/18}\log(n/\epsilon) + n^2r^{-2/3}\right)$.

A detailed proof of Lemma 6.5.9 is deferred to Section 6.8. The subsequent Lemma offers a helpful formula of $\boldsymbol{\Pi}_1^{\mathrm{up}}$, the orthogonal projection matrix onto the image of $\boldsymbol{L}_1^{\mathrm{up}}$.

Lemma 6.5.10. *Let $\mathcal{K}$ be a simplicial complex with boundary operator ∂_2. For any partition $F \cup C$ of the 2-simplexes of $\mathcal{K}$, the orthogonal projection $\mathbf{\Pi}_1^{up}$ for $\mathcal{K}$ can be decomposed as*

$$\mathbf{\Pi}_1^{up} = \mathbf{\Pi}_{Im(\partial_2(:,F))} + \mathbf{\Pi}_{Ker(\partial_2^\top(F,:))}\partial_2(:,C)(Sc(\boldsymbol{L}_2^{down})_C)^\dagger \partial_2^\top(C,:)\mathbf{\Pi}_{Ker(\partial_2^\top(F,:))},$$

where $\boldsymbol{L}_2^{down}$ is the down 2-Laplacian.

Once more, an r-hollowing offers a natural partition of the triangles within $\mathcal{K}$. We assign all the interior triangles to F and all the boundary triangles to C. As such, Nested Dissection can be utilized to compute $\mathbf{\Pi}_{\mathrm{Im}(\partial_2(:,F))}$ and $\mathbf{\Pi}_{\mathrm{Ker}(\partial_2^\top(F,:))}$, and PCG to solve the system in the Schur complement. The primary technical challenge arises when bounding the relative condition number of the Schur complement and the preconditioner, which requires a different approach.

6.6 Solver for Down-Laplacian

This section shows how to solve $\boldsymbol{L}_1^{\mathrm{down}}\boldsymbol{x}^{\mathrm{down}} = \boldsymbol{b}$ for any $\boldsymbol{b} \in \mathrm{Im}(\boldsymbol{L}_1^{\mathrm{down}})$ in linear time and proves Lemma 6.5.3. Recall that $\boldsymbol{L}_1^{\mathrm{down}} = \partial_1^\top \boldsymbol{W}_0 \partial_1$. Our down-Laplacian solver works for *any* simplicial complexes and returns a solution *without error*. Our approach is a slight modification of the down-Laplacian solver in [Coh+14a] to incorporate the vertex weights $\boldsymbol{W}_0$. Specifically, we compute $\boldsymbol{x}^{\mathrm{down}}$ by three steps: solve $\partial_1^\top \boldsymbol{W}_0^{1/2}\boldsymbol{y} = \boldsymbol{b}$, project $\boldsymbol{y}$ onto $\mathrm{Im}(\boldsymbol{W}_0^{1/2}\partial_1)$ and get a new vector $\boldsymbol{y}'$, then solve $\boldsymbol{W}_0^{1/2}\partial_1\boldsymbol{x}^{\mathrm{down}} = \boldsymbol{y}'$. The first and the last steps can be solved by the approach in Lemma 4.2 of [Coh+14a] (stated below), and the second step can be explicitly solved by utilizing that ∂_1 is a vertex-edge incidence matrix of an oriented graph.

Lemma 6.6.1 (Restatement of Lemma 4.2 of [Coh+14a])**.** *Given any $\boldsymbol{b}_1 \in Im(\partial_1^\top)$ (respectively, $\boldsymbol{b}_0 \in Im(\partial_1)$), there is a linear operator $\partial_1^{+\top}$ such that $\partial_1^\top\partial_1^{+\top}\boldsymbol{b}_1 = \boldsymbol{b}_1$ (respectively, $\partial_1^+ = (\partial_1^{+\top})^\top$ such that $\partial_1\partial_1^+\boldsymbol{b}_0 = \boldsymbol{b}_0$). In addition, we can compute $\partial_1^{+\top}\boldsymbol{b}_1$ (respectively, $\partial_1^+\boldsymbol{b}_0$) in linear time.*

We remark that the operators ∂_1^+ and $\partial_1^{+\top}$ in Lemma 6.6.1 are not necessary to be the pseudo-inverses of ∂_1 and $\partial_1^\top$.

Claim 6.6.2 explicitly characterizes the image of $\boldsymbol{W}_0^{1/2}\partial_1$.

Claim 6.6.2. *Let $\mathcal{K}$ be a simplicial complex whose 1-skeleton is connected, and let v be the number of vertices in $\mathcal{K}$. Let $\boldsymbol{W}_0 = \mathrm{diag}\,(w_1, \dots, w_v)$ where $w_1, \dots, w_v > 0$. Then,*

$$Ker(\partial_1^\top \boldsymbol{W}_0^{1/2}) = \mathrm{span}\,\{\boldsymbol{u}\}\,, \text{ where } \boldsymbol{u} = \left(\frac{1}{\sqrt{w_1}}, \dots, \frac{1}{\sqrt{w_v}}\right)^\top.$$

Proof. Let $\dim(\mathcal{V})$ be the dimension of a space $\mathcal{V}$. Since all the diagonals of $\boldsymbol{W}_0$ are positive,

$$\dim(\mathrm{Ker}(\partial_1^\top \boldsymbol{W}_0^{1/2})) = \dim(\mathrm{Ker}(\partial_1^\top)) = v - \dim(\mathrm{Im}(\partial_1)) = 1,$$

where the last equality holds since the 1-skeleton of $\mathcal{K}$ is connected. We can check that

$$\partial_1^\top \boldsymbol{W}_0^{1/2}\boldsymbol{u} = \partial_1^\top \mathbf{1} = \mathbf{0}.$$

Here, $\mathbf{1}$ is the all-one vector, and the second equality holds since ∂_1 is the vertex-edge incidence matrix of an oriented (weakly) connected graph. Thus, the statement holds. □

We construct our down-Laplacian solver by combining Lemma 6.6.1 and Claim 6.6.2. We remark that this down-Laplacian solver applies to any simplicial complex.

Proof of Lemma 6.5.3. Without loss of generality, we assume the 1-skeleton of $\mathcal{K}$ is connected. Otherwise, we can write $\boldsymbol{L}_1^{\text{down}}$ as a block diagonal matrix, each corresponding to a connected component of the 1-skeleton, and we reduce solving a system in $\boldsymbol{L}_1^{\text{down}}$ into solving several smaller down-Laplacian systems.

Let $\partial_1^+, \partial_1^{+\top}$ be the linear operators in Lemma 6.6.1 for $\mathcal{K}$, and let $\boldsymbol{u}$ be the vector in Claim 6.6.2. We define

$$\boldsymbol{Z}_1^{\text{down}} \stackrel{\text{def}}{=} \partial_1^+ \boldsymbol{W}_0^{-1/2} \left(\boldsymbol{I} - \frac{\boldsymbol{u}\boldsymbol{u}^\top}{\|\boldsymbol{u}\|_2^2} \right) \boldsymbol{W}_0^{-1/2} \partial_1^{+\top}.$$

We can compute $\boldsymbol{Z}_1^{\text{down}} \boldsymbol{b}$ in linear time. In addition, we define

$$\boldsymbol{y} = \partial_1^{+\top} \boldsymbol{b}, \ \boldsymbol{z} = \boldsymbol{W}_0^{-1/2} \boldsymbol{y}, \ \boldsymbol{z}_1 = \left(\boldsymbol{I} - \frac{\boldsymbol{u}\boldsymbol{u}^\top}{\|\boldsymbol{u}\|_2^2} \right) \boldsymbol{z}, \text{ and } \boldsymbol{x} = \partial_1^+ \boldsymbol{W}_0^{-1/2} \boldsymbol{z}_1.$$

Then, $\boldsymbol{z}_1 \in \operatorname{Im}(\boldsymbol{W}_0^{1/2} \partial_1)$ and $\boldsymbol{z} - \boldsymbol{z}_1 \in \operatorname{Ker}(\partial_1^\top \boldsymbol{W}_0^{1/2})$.

$$\begin{aligned}
\boldsymbol{L}_1^{\text{down}} \boldsymbol{Z}_1^{\text{down}} \boldsymbol{b} &= \partial_1^\top \boldsymbol{W}_0 \partial_1 \boldsymbol{x} = \partial_1^\top \boldsymbol{W}_0 \partial_1 \partial_1^+ \boldsymbol{W}_0^{-1/2} \boldsymbol{z}_1 \\
&= \partial_1^\top \boldsymbol{W}_0 \boldsymbol{W}_0^{-1/2} \boldsymbol{z}_1 && \text{since } \boldsymbol{W}_0^{-1/2} \boldsymbol{z}_1 \in \operatorname{Im}(\partial_1) \\
&= \partial_1^\top \boldsymbol{W}_0^{1/2} \boldsymbol{z} && \text{since } \boldsymbol{z} - \boldsymbol{z}_1 \in \operatorname{Ker}(\partial_1^\top \boldsymbol{W}_0^{1/2}) \\
&= \partial_1^\top \boldsymbol{y} = \boldsymbol{b}.
\end{aligned}$$

□

6.7 Solver for Up-Laplacian

In this section, we will prove the two key lemmas, Lemma 6.5.7 and Lemma 6.5.8, for building the first up-Laplacian solver.

6.7.1 Solver for $\mathbf{L}_1^{\text{up}}(F, F)$

In this section, we construct an efficient solver UpLapFSolver($\boldsymbol{b}$) that returns an $\boldsymbol{x}$ such that $\boldsymbol{L}_1^{\text{up}}(F, F)\boldsymbol{x} = \boldsymbol{b}$ for any $\boldsymbol{b} \in \operatorname{Im}(\boldsymbol{L}_1^{\text{up}}(F, F))$ and prove Lemma 6.5.7.

We first show that the interiors of different regions are "disjoint" in the sense that $\boldsymbol{L}_1^{\text{up}}(F, F)$ can be written as a block diagonal matrix where each diagonal block corresponds to the interior of a region after proper row and column permutation. By definition, we can write $\boldsymbol{L}_1^{\text{up}}$ as the sum of rank-1 matrices that each corresponds to a triangle in $\mathcal{K}$:

$$\boldsymbol{L}_1^{\text{up}} = \partial_2 \boldsymbol{W}_2 \partial_2^\top = \sum_{\sigma:\text{triangle in } \mathcal{K}} \boldsymbol{W}_2(\sigma, \sigma) \cdot \partial_2(:, \sigma) \partial_2(:, \sigma)^\top.$$

For any two edges e_1, e_2, $\boldsymbol{L}_1^{\text{up}}(e_1, e_2) = 0$ if and only if no triangle in $\mathcal{K}$ contains both e_1, e_2. We say such e_1 and e_2 are $\boldsymbol{\Delta}$-*disjoint*. The following claim shows that interior edges from different regions are $\boldsymbol{\Delta}$-disjoint.

Claim 6.7.1. *Let $\mathcal{K}$ be a stable 3-complex and $\mathcal{T}$ be an r-hollowing of $\mathcal{K}$. Let R_1, R_2 be two different regions of $\mathcal{K}$ w.r.t. $\mathcal{T}$, and let e_1 be an interior edge in R_1 and e_2 an interior edge in R_2. Then, e_1 and e_2 are $\boldsymbol{\Delta}$-disjoint.*

Proof. Assume by contradiction there exists a triangle $\sigma \in \mathcal{K}$ that contains both e_1, e_2. Let e_3 be the third edge in σ. Since e_1, e_2 are interior edges of different regions R_1, R_2, e_3 must cross the boundary of R_1, R_2. This contradicts the fact that regions can only intersect on their boundaries. Thus, e_1, e_2 must be $\boldsymbol{\Delta}$-disjoint. □

By the above Claim 6.7.1, computing an $\boldsymbol{x}$ such that $\boldsymbol{L}_1^{\mathrm{up}}(F,F)\boldsymbol{x} = \boldsymbol{b}$ reduces to computing a solution for each diagonal block submatrix per region. For this reason, Lemma 6.5.7 is a corollary of the following Lemma 6.7.2.

Lemma 6.7.2. *Let $\mathcal{X}$ be a 3-complex with $O(r)$ simplexes embedded in $\mathbb{R}^3$ such that (1) each tetrahedron of $\mathcal{X}$ has $O(1)$ aspect ratio, and (2) $\mathcal{X}$ has $O(r^{2/3})$ exterior simplexes. Let F be the set of interior edges of $\mathcal{X}$, and let $\boldsymbol{L}_{1,\mathcal{X}}^{up}$ be the up-Laplacian of $\mathcal{X}$ and $\boldsymbol{M} \stackrel{\mathrm{def}}{=} \boldsymbol{L}_{1,\mathcal{X}}^{up}(F,F)$. Then, there is a permutation matrix $\boldsymbol{P}$ and a lower triangular matrix $\boldsymbol{L}$ with $O(r^{4/3})$ non-zeros such that*

$$\boldsymbol{M} = \boldsymbol{P}\boldsymbol{L}\boldsymbol{L}^\top\boldsymbol{P}^\top. \tag{6.2}$$

In addition, we can find such $\boldsymbol{P}$ and $\boldsymbol{L}$ in time $O(r^2)$. Given the above factorization, for any $\boldsymbol{b} \in Im(\boldsymbol{M})$, we can compute an $\boldsymbol{x}$ such that $\boldsymbol{M}\boldsymbol{x} = \boldsymbol{b}$ in $O(r^{4/3})$ time.

The factorization in Equation (6.2) is called *Cholesky factorization.* We will utilize the geometric structures of $\mathcal{X}$ to find a sparse Cholesky factorization efficiently. Miller and Thurston [MT90] studied vertex separators of the 1-skeleton of a 3-complex $\mathcal{X}$ in which each tetrahedron has $O(1)$ aspect ratio. A subset of vertices S of a graph over v vertices *δ-separates* if the remaining vertices can be partitioned into two sets A, B such that there are no edges between A and B, and $|A|, |B| \leq \delta v$. The set S is an *$f(v)$-separator* if there exists a constant $\delta < 1$ such that S δ-separates and $|S| \leq f(v)$. These separators can be incorporated with Nested Dissection to efficiently compute a sparse Cholesky factorization of a matrix in which the non-zero structure encodes the 1-skeleton of $\mathcal{X}$.

Theorem 6.7.3 (Vertex Separator for a 3-Complex, Theorem 1.5 of [MT90] and [Mil+98])**.** *Let $\mathcal{X}$ be a 3-complex in $\mathbb{R}^3$ in which each tetrahedron has $O(1)$ aspect ratio, and suppose $\mathcal{X}$ has t tetrahedra and $\bar{v}$ exterior vertices. Then, the 1-skeleton of $\mathcal{X}$ has a $O(t^{2/3} + \bar{v})$-separator that $4/5$-separates $\mathcal{X}$. In addition, such a separator can be found in linear time.*

We need a slightly modified version of Theorem 6.7.3 to apply to our matrix $\boldsymbol{M}$.

Corollary 6.7.4 (Edge Separator for a 3-Complex)**.** *Let $\mathcal{X}$ be a 3-complex that satisfies the requirements in Theorem 6.7.3. In addition, the 1-skeleton of $\mathcal{X}$ is connected. Let m be the number of edges in $\mathcal{X}$. Then, there exists an algorithm that removes $O(t^{2/3} + \bar{v})$ edges in linear time so that the remaining edges can be partitioned into two $\boldsymbol{\Delta}$-disjoint sets, each of size at most cm for some constant $c < 1$.*

Proof. Let $v = O(r)$ be the number of vertices in $\mathcal{X}$. By Lemma 6.3.1, $v = \Theta(m)$. Let S be the set of $O(t^{2/3} + \bar{v})$ vertices that $4/5$-separates $\mathcal{X}$ (by Theorem 6.7.3), and let A, B

be two disjoint sets of the remaining vertices after removing S such that $|A|, |B| \leq 4v/5$. Let E be the set of edges in $\mathcal{X}$ incident to some vertex in S. Since each vertex has $O(1)$ degree by Lemma 6.3.1, $|E| = O(|S|) = O(t^{2/3} + \bar{v})$. Now, we remove edges in E from the edges of $\mathcal{X}$. Let E_A be the set of the remaining edges incident to a vertex in A, and let E_B be the set of the remaining edges incident to a vertex in B. Since A, B are disjoint, E_A and E_B are $\boldsymbol{\Delta}$-disjoint. We then show that $|E_A|, |E_B| \leq cm$ for some constant $c < 1$. Let E'_A be the set of edges in $\mathcal{X}$ that are incident to some vertex in A. Since the 1-skeleton of $\mathcal{X}$ is a connected graph, $|E'_A| \geq |A| \geq (1 - o(1))\frac{v}{5}$. Besides, $|E'_A| = |E_A| + |E''_A|$, where E''_A contains edges with one endpoint in A and the other in S. Since $|E''_A| = O(t^{2/3} + \bar{v})$, we know $|E_A| \geq (1 - o(1))\frac{v}{5} > c'm$ for some constant c'. Since E_A and E_B are disjoint, we know $|E_B| \leq m - |E_A| \leq (1 - c')m$. By symmetry, we have $|E_A| \leq (1 - c')m$. $\square$

We need the following theorem about Nested Dissection from [LRT79].

Theorem 6.7.5 (Nested Dissection, Theorem 6 of [LRT79])**.** *Let $\mathcal{G}$ be any class of graphs closed under subgraph on which a v^{α}-separator exists for $\alpha > 1/2$. Let $\boldsymbol{A} \in \mathbb{R}^{v \times v}$ be symmetric and positive definite (that is, all eigenvalues of $\boldsymbol{A}$ are positive). Let $G_{\boldsymbol{A}}$ be a graph over vertices $\{1, \dots, v\}$ where vertices i, j are connected if and only if $\boldsymbol{A}(i, j) \neq 0$. If $G_{\boldsymbol{A}} \in \mathcal{G}$, then we can find a permutation matrix $\boldsymbol{P}$ and a lower triangular matrix $\boldsymbol{L}$ with $O(v^{2\alpha})$ non-zeros in $O(v^{3\alpha})$ time such that $\boldsymbol{A} = \boldsymbol{P}\boldsymbol{L}\boldsymbol{L}^{\top}\boldsymbol{P}^{\top}$.*

Combining Corollary 6.7.4 and Theorem 6.7.5, we prove Lemma 6.7.2.

Proof of Lemma 6.7.2. Our approach for finding a sparse Cholesky factorization is the same as Section 4 of [MT90]. Specifically, we find a family of separators for $\mathcal{X}$ by recursively applying Corollary 6.7.4 to the remaining sets of edges A, B. Since the separator size grows as a function of the exterior vertices, at the top level of the recursion, we include all the exterior edges $O(r^{2/3})$ in the root separator, which only increases the root separator by a constant factor. In each remaining recursion step, suppose we want to separate $\mathcal{Y}$, a sub-complex of $\mathcal{X}$; we let $\bar{\mathcal{Y}}$ be the complex consisting of $\mathcal{Y}$ and all the simplexes in $\mathcal{X}$ that contain a vertex in $\mathcal{Y}$. Then we apply a slightly modified version of Corollary 6.7.4 (obtained by a slightly modified Theorem 6.7.3) to $\bar{\mathcal{Y}}$ in which we only separate $\mathcal{Y}$. Here, the exterior-vertex term $O(\bar{v})$ in the size of the separator can be ignored since all the exterior edges of $\bar{\mathcal{Y}}$ have already been included in upper-level separators; we also use the fact that the number of exterior vertices and the number of exterior edges are equal up to a constant factor. This separator family provides an elimination ordering for the edges of $\mathcal{K}$, which is the permutation matrix $\boldsymbol{P}$ in Equation (6.2), and the ordering uniquely determines the matrix $\boldsymbol{L}$. By Theorem 6.7.5, $\boldsymbol{L}$ has $O(r^{4/3})$ non-zeros, and $\boldsymbol{P}, \boldsymbol{L}$ can be found in time $O(r^2)$.

One issue left is that $\boldsymbol{M}$ in Lemma 6.7.2 is positive semidefinite but *not* positive definite. During the process of numeric factorization, the first row and column of some Schur complements are all-zero. We simply ignore these zeros and proceed (Ref: Chapter 4.2.8 of [GV96]). This produces a Cholesky factorization of $\boldsymbol{P}^{\top}\boldsymbol{M}\boldsymbol{P} = \boldsymbol{L}\boldsymbol{L}^{\top}$ such that only k columns of $\boldsymbol{L}$ are non-zero, where k is the rank of $\boldsymbol{M}$. We permute the rows and the columns of $\boldsymbol{L}$ by multiplying permutation matrices $\boldsymbol{P}_1, \boldsymbol{P}_2$ so that

$$\boldsymbol{P}_1\boldsymbol{L}\boldsymbol{P}_2 = \begin{bmatrix}\boldsymbol{T} & \boldsymbol{0}\end{bmatrix} \overset{\text{def}}{=} \begin{bmatrix}\boldsymbol{H}_1 & \boldsymbol{0} \\ \boldsymbol{H}_2 & \boldsymbol{0}\end{bmatrix},$$

where $\boldsymbol{H}_1$ is a $k \times k$ non-singular lower-triangular matrix. Then, solving $\boldsymbol{M}\boldsymbol{x} = \boldsymbol{b}$ is equivalent to solving

$$\boldsymbol{T}\boldsymbol{T}^\top \boldsymbol{z} = \boldsymbol{P}_1 \boldsymbol{P}^\top \boldsymbol{b} \stackrel{\text{def}}{=} \boldsymbol{f}, \ \boldsymbol{P}_1 \boldsymbol{P}^\top \boldsymbol{x} = \boldsymbol{z}.$$

We solve the first system by solving $\boldsymbol{T}\boldsymbol{y} = \boldsymbol{f}$ and $\boldsymbol{T}^\top \boldsymbol{z} = \boldsymbol{y}$. We let $\boldsymbol{y}$ satisfy $\boldsymbol{H}_1 \boldsymbol{y} = \boldsymbol{f}[1:k]$. Since $\boldsymbol{H}_1$ has full rank and $O(r^{4/3})$ non-zeros, such a $\boldsymbol{y}$ exists and can be found in $O(r^{4/3})$ time. Since $\boldsymbol{y} \in \operatorname{Im}(\boldsymbol{T})$, we know $\boldsymbol{T}\boldsymbol{y} = \boldsymbol{f}$. Then we let $\boldsymbol{z}[k+1:v] = 0$, where v is the number of vertices in $\mathcal{X}$, and we let $\boldsymbol{H}_1^\top \boldsymbol{z}[1:k] = \boldsymbol{y}$. Again, such $\boldsymbol{z}$ exists and can be found in $O(r^{4/3})$ time. Finally, we compute $\boldsymbol{x} = \boldsymbol{P}\boldsymbol{P}_1^\top \boldsymbol{z}$ in linear time. □

Proof of Lemma 6.5.7. We apply Lemma 6.7.2 to each diagonal block matrix of $\boldsymbol{L}_1^{\text{up}}(F,F)$ to compute a sparse Cholesky factorization as in Equation (6.2) in time

$$O\left(\frac{n}{r} \cdot r^2\right) = O(nr).$$

Given this Cholesky factorization, we can solve a system in $\boldsymbol{L}_1^{\text{up}}(F,F)$ in time

$$O\left(\frac{n}{r} \cdot r^{4/3}\right) = O\left(nr^{1/3}\right).$$

□

6.7.2 Solver for the Schur Complement

In this section, we establish a fast approximate solver for $\operatorname{Sc}(\boldsymbol{L}_1^{\text{up}})_C$ and prove Lemma 6.5.8. Recall that C contains all the edges in $\mathcal{T}$ (an r-hollowing of $\mathcal{K}$), and $\operatorname{Sc}(\boldsymbol{L}_1^{\text{up}}]_C$ is the Schur complement of the up-Laplacian operator of $\mathcal{K}$ onto C. The idea is to run the Preconditioned Conjugate Gradient (PCG) for systems in $\operatorname{Sc}(\boldsymbol{L}_1^{\text{up}})_C$ with preconditioner $\boldsymbol{L}_{1,\mathcal{T}}^{\text{up}} = \partial_{2,\mathcal{T}} \partial_{2,\mathcal{T}}^\top$, which is the first up-Laplacian operator of $\mathcal{T}$. By Theorem 6.3.4, the number of PCG iterations is $O(\sqrt{\kappa} \log(\kappa/\epsilon))$ where $\kappa = \kappa(\operatorname{Sc}(\boldsymbol{L}_1^{\text{up}})_C, \boldsymbol{L}_{1,\mathcal{T}}^{\text{up}})$ is the relative condition number and ϵ is the error parameter; in each PCG iteration, we need to solve a system in $\boldsymbol{L}_{1,\mathcal{T}}^{\text{up}}$, multiply $\operatorname{Sc}(\boldsymbol{L}_1^{\text{up}}]_C$ with $O(1)$ vectors, and implement $O(1)$ vector operations. In Section 6.7.2, we upper bound the relative condition number. In Section 6.7.2, we prove Lemma 6.5.8 about the solver for Schur complement.

We will need the following observation.

Claim 6.7.6. *Let $\mathcal{X}$ be a simplicial complex. Changing the orientations of the triangles in $\mathcal{X}$ does not change its first up-Laplacian operator.*

Proof. Let $\partial_{2,\mathcal{X}}$ be the second boundary operator of $\mathcal{X}$, $\boldsymbol{W}_{2,\mathcal{X}}$ the diagonal matrix for the triangle weights, and $\boldsymbol{L}_{1,\mathcal{X}}^{\text{up}}$ the first up-Laplacian. Changing the orientations of the triangles in $\mathcal{X}$ corresponds to multiplying a ± 1 diagonal matrix $\boldsymbol{X}$ to the right of $\partial_{2,\mathcal{X}}$. Observe

$$\partial_{2,\mathcal{X}} (\boldsymbol{X} \boldsymbol{W}_{2,\mathcal{X}} \boldsymbol{X}) \partial_{2,\mathcal{X}}^\top = \partial_{2,\mathcal{X}} \boldsymbol{W}_{2,\mathcal{X}} \partial_{2,\mathcal{X}}^\top = \boldsymbol{L}_{1,\mathcal{X}}^{\text{up}}.$$

Thus the statement holds. □

Preconditioning the Schur Complement

We will upper bound the relative condition number of $\mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_C$ and $\boldsymbol{L}_{1,\mathcal{T}}^{\mathrm{up}}$ and prove will Lemma 6.5.6.

We decompose the r-hollowing $\mathcal{T}$ into two parts. Recall that the boundary of each region triangulates a spherical shell in $\mathbb{R}^3$. Let $\mathcal{B}_1$ be a 2-complex of the union of all the inner spheres of the boundaries, and let $\mathcal{B}_2$ be a 2-complex of the rest of the boundaries. Let C_1 be the set of edges of $\mathcal{B}_1$. Let $\widehat{\boldsymbol{L}_1^{\mathrm{up}}}$ be the first up-Laplacian of the union of $\mathcal{B}_1$ and all the interior simplexes in all the regions. Then,

$$\boldsymbol{L}_{1,\mathcal{T}}^{\mathrm{up}} = \boldsymbol{L}_{1,\mathcal{B}_1}^{\mathrm{up}} + \boldsymbol{L}_{1,\mathcal{B}_2}^{\mathrm{up}} \text{ and } \mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_C = \mathrm{Sc}(\widehat{\boldsymbol{L}_1^{\mathrm{up}}})_{C_1} + \boldsymbol{L}_{1,\mathcal{B}_2}^{\mathrm{up}}.$$

Claim 6.7.7. *If $\boldsymbol{L}_{1,\mathcal{B}_1}^{up} \preccurlyeq Sc(\widehat{\boldsymbol{L}_1^{up}})_{C_1} \preccurlyeq \alpha \boldsymbol{L}_{1,\mathcal{B}_1}^{up}$, where $\alpha > 1$, then*

$$\boldsymbol{L}_{1,\mathcal{T}}^{up} \preccurlyeq Sc(\boldsymbol{L}_1^{up})_C \preccurlyeq \alpha \boldsymbol{L}_{1,\mathcal{T}}^{up}.$$

Proof. We have

$$\begin{aligned} \mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_C &\succcurlyeq \boldsymbol{L}_{1,\mathcal{B}_1}^{\mathrm{up}} + \boldsymbol{L}_{1,\mathcal{B}_2}^{\mathrm{up}} = \boldsymbol{L}_{1,\mathcal{T}}^{\mathrm{up}}, \\ \mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_C &\preccurlyeq \alpha \boldsymbol{L}_{1,\mathcal{B}_1}^{\mathrm{up}} + \boldsymbol{L}_{1,\mathcal{B}_2}^{\mathrm{up}} \preccurlyeq \alpha \boldsymbol{L}_{1,\mathcal{T}}^{\mathrm{up}}. \end{aligned}$$

□

To prove Lemma 6.5.6, it suffices to show the following lemma.

Lemma 6.7.8.

$$\boldsymbol{L}_{1,\mathcal{B}_1}^{up} \preccurlyeq Sc(\widehat{\boldsymbol{L}_1^{up}})_{C_1} \preccurlyeq O(r) \boldsymbol{L}_{1,\mathcal{B}_1}^{up}.$$

The first inequality of Lemma 6.7.8 follows immediately from a well-known fact about Schur complements, stated in Fact 2.3.7. For completeness, we restate it below.

Fact 6.7.9. *Let $\boldsymbol{A}$ be a symmetric and PSD matrix:*

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}(F,F) & \boldsymbol{A}(F,C) \\ \boldsymbol{A}(C,F) & \boldsymbol{A}(C,C) \end{bmatrix}.$$

Let $Sc(\boldsymbol{A})_C = \boldsymbol{A}(C,C) - \boldsymbol{A}(C,F)\boldsymbol{A}(F,F)^{\dagger}\boldsymbol{A}(F,C)$ be the Schur complement of $\boldsymbol{A}$ onto C. Then, for any $\boldsymbol{x} \in \mathbb{R}^{|C|}$,

$$\min_{\boldsymbol{y} \in \mathbb{R}^{|F|}} \begin{bmatrix} \boldsymbol{y}^\top & \boldsymbol{x}^\top \end{bmatrix} \boldsymbol{A} \begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{x} \end{bmatrix} = \boldsymbol{x}^\top Sc(\boldsymbol{A})_C \boldsymbol{x}.$$

As a corollary, $Sc(\boldsymbol{A})_C$ is symmetric and PSD.

In the rest of the section, we prove the second inequality in Lemma 6.7.8. Since both $\widehat{\boldsymbol{L}_1^{\mathrm{up}}}$ and $\boldsymbol{L}_{1,\mathcal{B}_1}^{\mathrm{up}}$ can be written as a block diagonal matrix where each block corresponds to a region w.r.t. $\mathcal{T}$ (after proper row and column permutation), it suffices to show the inequality in Lemma 6.7.8 holds for each region, restated in the following lemma.

Lemma 6.7.10. *Consider an r-hollowing region. Let $\mathcal{B}$ be the 2-complex of the inner sphere of the boundary, and let B be the set of edges in $\mathcal{B}$. Let $\mathcal{X}$ be a 2-complex of the union of $\mathcal{B}$ and the interior simplexes. Then,*

$$Sc(\boldsymbol{L}_{1,\mathcal{X}}^{up})_B \preccurlyeq O(r)\, \boldsymbol{L}_{1,\mathcal{B}}^{up}.$$

We first show that the images of $\mathrm{Sc}(\boldsymbol{L}_{1,\mathcal{X}}^{\mathrm{up}})_C$ and $\boldsymbol{L}_{1,\mathcal{B}}^{\mathrm{up}}$ are equal.

Claim 6.7.11.

$$Im(Sc(\boldsymbol{L}_{1,\mathcal{X}}^{up})_B) = Im(\boldsymbol{L}_{1,\mathcal{B}}^{up}).$$

Proof. In the proof, we drop the subscript $\mathcal{X}$ to simplify our notations by letting $\boldsymbol{L}_1^{\mathrm{up}} = \boldsymbol{L}_{1,\mathcal{X}}^{\mathrm{up}}$ and $\partial_2 = \partial_{2,\mathcal{X}}$. We let A be the set of edges in $\mathcal{X}$ but not in B, and let V_A be the set of vertices in $\mathcal{X}$ but not in $\mathcal{B}$ and V_B the set of vertices in $\mathcal{B}$.

Since both the two matrices $\mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_B, \boldsymbol{L}_{1,\mathcal{B}}^{\mathrm{up}}$ are symmetric and PSD, the statement in the claim is equivalent to $\mathrm{Ker}(\mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_B) = \mathrm{Ker}(\boldsymbol{L}_{1,\mathcal{B}}^{\mathrm{up}})$. By Fact 6.7.9,

$$\boldsymbol{L}_{1,\mathcal{B}}^{\mathrm{up}} \preccurlyeq \mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_B \implies \mathrm{Ker}(\mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_B) \subseteq \mathrm{Ker}(\boldsymbol{L}_{1,\mathcal{B}}^{\mathrm{up}}).$$

It remains to show $\mathrm{Ker}(\mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_B) \supseteq \mathrm{Ker}(\boldsymbol{L}_{1,\mathcal{B}}^{\mathrm{up}})$.

Let $\boldsymbol{x}_B$ be an arbitrary vector in $\mathrm{Ker}(\boldsymbol{L}_{1,\mathcal{B}}^{\mathrm{up}})$. We want to show $\boldsymbol{x}_B \in \mathrm{Ker}(\mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_B)$, that is, $\boldsymbol{x}_B^\top \mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_B \boldsymbol{x}_B = 0$. By Fact 6.7.9, it suffices to show that there exists an $\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_A \\ \boldsymbol{x}_B \end{bmatrix}$ such that $\boldsymbol{x}^\top \boldsymbol{L}_1^{\mathrm{up}} \boldsymbol{x} = 0$. This is equivalent to $\boldsymbol{x} \in \mathrm{Ker}(\partial_2^\top)$. Suppose we add tetrahedra, triangles, and necessary edges to $\mathcal{X}$ and get a new simplicial complex $\mathcal{X}'$ so that $\mathcal{B}$ is the boundary of $\mathcal{X}'$ and $\mathcal{X}'$ triangulates a 3-ball. Since operator $\partial_2^\top$ maps the vector space of edges to the vector space of triangles and no new edges in $\mathcal{X}'$ can appear in a triangle in $\mathcal{X}$, it suffices to show there exists an $\boldsymbol{x}' = \begin{bmatrix} \boldsymbol{x}'_A \\ \boldsymbol{x}_B \end{bmatrix}$ such that $\boldsymbol{x}' \in \mathrm{Ker}(\partial_{2,\mathcal{X}'}^\top)$ (restricting $\boldsymbol{x}'_A$ to $\boldsymbol{x}_A$ gives $\boldsymbol{x} \in \mathrm{Ker}(\partial_{2,\mathcal{X}}^\top)$). Since the first Betti number of $\mathcal{X}'$ is zero,

$$\boldsymbol{x} \in \mathrm{Ker}(\partial_{2,\mathcal{X}'}^\top) \iff \boldsymbol{x} \perp \mathrm{Im}(\partial_{2,\mathcal{X}'}) \iff \boldsymbol{x} \in \mathrm{Im}(\partial_{1,\mathcal{X}'}^\top). \tag{6.3}$$

Let A' be the set of interior edges in $\mathcal{X}'$, and $V_{A'}$ the set of interior vertices in $\mathcal{X}'$. We can write

$$\partial_{1,\mathcal{X}'}^\top = \begin{bmatrix} \partial_{1,\mathcal{X}'}^\top(A', V_{A'}) & \partial_{1,\mathcal{X}'}^\top(A', V_B) \\ \boldsymbol{0} & \partial_{1,\mathcal{X}'}^\top(B, V_B) \end{bmatrix},$$

where $\partial_{1,\mathcal{X}'}(V_B, B) = \partial_{1,\mathcal{B}}$. Since $\boldsymbol{x}_B \in \mathrm{Ker}(\partial_{2,\mathcal{B}}^\top)$ and the first Betti number of $\mathcal{B}$ is 0, by an argument similar to Equation (6.3), we have $\boldsymbol{x}_B \in \mathrm{Im}(\partial_{1,\mathcal{B}}^\top)$, that is, $\boldsymbol{x}_B = \partial_{1,\mathcal{B}}^\top \boldsymbol{y}_B$ for some $\boldsymbol{y}_B$. Setting $\boldsymbol{x}'_A = \partial_1^\top(A', V_B)\boldsymbol{y}_B$, we have $\boldsymbol{x} \in \mathrm{Im}(\partial_{1,\mathcal{X}'}^\top)$. □

Claim 6.7.12.

$$\lambda_{\max}(Sc(\boldsymbol{L}_{1,\mathcal{X}}^{up})_B) = O(w_{\max}),$$

where $w_{\max}$ is the maximum triangle weight in $\mathcal{X}$.

Proof. We drop the subscript $\mathcal{X}$ in the proof to simplify our notations. The Courant-Fischer Minimax Theorem (see Theorem 4.5.7, or Theorem 8.1.2 of [GV96]) states that for any symmetric matrix $\boldsymbol{A}$,

$$\lambda_{\max}(\boldsymbol{A}) = \max_{\boldsymbol{x}:\|\boldsymbol{x}\|_2=1} \boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x}.$$

Apply this theorem and Fact 6.7.9,

$$\lambda_{\max}(\mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_B) = \max_{\boldsymbol{x}:\|\boldsymbol{x}\|_2=1} \boldsymbol{x}^\top \mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_B \boldsymbol{x} \le \max_{\boldsymbol{x}:\|\boldsymbol{x}\|_2=1} \begin{bmatrix} \boldsymbol{0}^\top & \boldsymbol{x}^\top \end{bmatrix} \boldsymbol{L}_1^{\mathrm{up}} \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{x} \end{bmatrix} \le \lambda_{\max}(\boldsymbol{L}_1^{\mathrm{up}}).$$

Below, we bound $\lambda_{\max}(\boldsymbol{L}_1^{\mathrm{up}})$:

$$\begin{aligned}\lambda_{\max}(\boldsymbol{L}_1^{\mathrm{up}}) = \max_{\boldsymbol{x}:\|\boldsymbol{x}\|_2=1} \boldsymbol{x}^\top \boldsymbol{L}_1^{\mathrm{up}} \boldsymbol{x} &= \max_{\boldsymbol{x}:\|\boldsymbol{x}\|_2=1} \sum_{\sigma:\text{triangle in } \mathcal{X}} \boldsymbol{W}_2(\sigma,\sigma)(\partial_2(:,\sigma)^\top \boldsymbol{x})^2 \\ &\leq 3w_{\max} \cdot \max_{\boldsymbol{x}:\|\boldsymbol{x}\|_2=1} \sum_{\substack{\sigma=[i,j,k]:\\ \text{triangle in } \mathcal{X}}} (\boldsymbol{x}(i)^2 + \boldsymbol{x}(j)^2 + \boldsymbol{x}(k)^2) = O(w_{\max}).\end{aligned}$$

The last inequality holds since each edge appears in at most $O(1)$ triangles by Lemma 6.3.1. Thus, $\lambda_{\max}(\mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_B) = O(w_{\max})$. □

One more piece we need is a lower bound for $\lambda_{\min}(\boldsymbol{L}_{1,\mathcal{B}}^{\mathrm{up}})$, which will be established via eigenvalues of graph Laplacian matrices.

Theorem 6.7.13 (Section 4.2 of [Moh91])**.** *Let G be an unweighted graph over n vertices with diameter D, the length of the longest path in G. Let $\boldsymbol{L}_G$ be the graph Laplacian matrix of G. Then,*

$$\lambda_{\min}(\boldsymbol{L}_G) \geq \frac{4}{nD}.$$

Claim 6.7.14.

$$\lambda_{\min}(\boldsymbol{L}_{1,\mathcal{B}}^{up}) = \Omega(w_{\min} \cdot r^{-1}),$$

where $w_{\min}$ is the minimum triangle weight in $\mathcal{B}$.

Proof. We drop the subscript $\mathcal{B}$ in the proof to simplify our notations. Again we decompose the matrix as a sum of rank-1 matrices for each triangle in $\mathcal{B}'$:

$$\boldsymbol{L}_1^{\mathrm{up}} = \sum_{\sigma:\text{triangle in } \mathcal{B}} \boldsymbol{W}_2(\sigma,\sigma)\partial_2(:,\sigma)\partial_2(:,\sigma)^\top \succcurlyeq w_{\min} \cdot \partial_2\partial_2^\top.$$

So,

$$\lambda_{\min}(\boldsymbol{L}_1^{\mathrm{up}}) \geq w_{\min} \cdot \lambda_{\min}(\partial_2\partial_2^\top) = w_{\min} \cdot \lambda_{\min}(\partial_2^\top\partial_2).$$

Since $\mathcal{B}$ triangulates a two-sphere, each edge of $\mathcal{B}$ appears in exactly two triangles. Since changing the orientations of the triangles in $\mathcal{B}$ does not change $\boldsymbol{L}_1^{\mathrm{up}}$ (by Claim 6.7.6), we assume all the triangles in $\mathcal{B}$ are oriented clockwise. Then, each column of $\partial_2^\top$ has exactly one entry with value 1 and one entry -1 and all others 0. That is, $\partial_2^\top\partial_2$ is the Laplacian of the dual graph of $\mathcal{B}$: the vertices are the triangles in $\mathcal{B}$, and two vertices are adjacent if and only if the corresponding two triangles share a common edge. The dual graph has $O(r^{2/3})$ vertices and diameter $O(r^{1/3})$. By Theorem 6.7.13, $\lambda_{\min}(\partial_2^\top\partial_2) = \Omega(w_{\min} \cdot r^{-1})$. □

Combining all the claims above, we prove Lemma 6.7.10.

Proof of Lemma 6.7.10. Let $\boldsymbol{\Pi}$ be the orthogonal projection matrix onto $\mathrm{Im}(\boldsymbol{L}_{1,\mathcal{B}}^{\mathrm{up}}) = \mathrm{Im}(\mathrm{Sc}(\boldsymbol{L}_{1,\mathcal{X}}^{\mathrm{up}})_B)$ (by Claim 6.7.11). By Claim 6.7.12, $\mathrm{Sc}(\boldsymbol{L}_{1,\mathcal{X}}^{\mathrm{up}})_B \preccurlyeq O(w_{\max})\boldsymbol{\Pi}$. By Claim 6.7.14, $\boldsymbol{\Pi} \preccurlyeq O(\frac{r}{w_{\min}})\boldsymbol{L}_{1,\mathcal{B}}^{\mathrm{up}}$. Combining all together, we have

$$\mathrm{Sc}(\boldsymbol{L}_{1,\mathcal{X}}^{\mathrm{up}})_B \preccurlyeq O(rU)\boldsymbol{L}_{1,\mathcal{B}}^{\mathrm{up}},$$

where $U = \frac{w_{\max}}{w_{\min}} = O(1)$. □

Proof of Lemma 6.5.8

By Theorem 6.3.4 and Lemma 6.5.6, the number of Preconditioned Conjugate Gradient (PCG) iterations is

$$\widetilde{O}\left(\kappa\left(\mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_C, \boldsymbol{L}_{1,\mathcal{T}}^{\mathrm{up}}\right)^{1/2}\right) = \widetilde{O}(r^{1/2}).$$

In each PCG iteration, we solve a system in $\boldsymbol{L}_{1,\mathcal{T}}^{\mathrm{up}}$ and multiply $\mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_C$ with $O(1)$ vectors. Recall

$$\mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_C = \boldsymbol{L}_1^{\mathrm{up}}(C,C) - \boldsymbol{L}_1^{\mathrm{up}}(C,F)\boldsymbol{L}_1^{\mathrm{up}}(F,F)^{\dagger}\boldsymbol{L}_1^{\mathrm{up}}(F,C).$$

In our preprocessing, we compute a Cholesky factorization of $\boldsymbol{L}_1^{\mathrm{up}}(F,F)$ in time $O(r^2 \cdot \frac{n}{r}) = O(nr)$; then, we can multiply $\mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_C$ onto a vector in time $O(r^{4/3} \cdot \frac{n}{r}) = O(nr^{1/3})$. Similarly, we solve a system in $\boldsymbol{L}_{1,\mathcal{T}}^{\mathrm{up}}$ by Nested Dissection. By our construction, $\mathcal{T}$ has $O(r^{2/3} \cdot \frac{n}{r}) = O(nr^{-1/3})$ triangles. The Cholesky factorization runs in time $O(n^2 r^{-2/3})$, and each system solve runs in time $O(n^{4/3} r^{-4/9})$. So, the runtime per PCG iteration is

$$O\left(nr^{1/3} + n^{4/3} r^{-4/9}\right).$$

Therefore, the total time is

$$\widetilde{O}\left(r^{1/2}\left(nr^{1/3} + n^{4/3}r^{-4/9}\right) + nr + n^2 r^{-2/3}\right) = \widetilde{O}\left(nr + n^{4/3} r^{5/18} + n^2 r^{-2/3}\right).$$

This completes the proof of Lemma 6.5.8.

6.8 Projection onto the Image of Up-Laplacian

In this section, we will show how to approximately project a vector onto the image of the first up-Laplacian of a well-shaped 3-complex with a given r-hollowing and prove Lemma 6.5.9. The following lemma gives an explicit formula for the orthogonal projection onto the up-Laplacian.

Lemma 6.8.1. *Let $\mathcal{K}$ be a simplicial complex with boundary operator ∂_2. For any partition $F \cup C$ of the 2-simplexes of $\mathcal{K}$, the orthogonal projection $\boldsymbol{\Pi}_1^{up}$ for $\mathcal{K}$ can be decomposed as*

$$\boldsymbol{\Pi}_1^{up} = \boldsymbol{\Pi}_{Im(\partial_2(:,F))} + \boldsymbol{\Pi}_{Ker(\partial_2^\top(F,:))}\partial_2(:,C)(Sc(\boldsymbol{L}_2^{down})_C)^{\dagger}\partial_2^\top(C,:)\boldsymbol{\Pi}_{Ker(\partial_2^\top(F,:))}. \tag{6.4}$$

In addition, the second matrix on the right-hand side of the above equation is the orthogonal projection matrix onto the image of $\boldsymbol{\Pi}_{Ker(\partial_2^\top(F,:))}\partial_2(:,C)$.

Proof. For any $\boldsymbol{b} \in \mathbb{R}^n$, we define

$$\begin{aligned} \boldsymbol{f}_F &= \partial_2^\top(F,:)\boldsymbol{b}, \quad \boldsymbol{f}_C = \partial_2^\top(C,:)\boldsymbol{b}, \\ \boldsymbol{h} &= \boldsymbol{f}_C - \boldsymbol{L}_2^{\mathrm{down}}(C,F)(\boldsymbol{L}_2^{\mathrm{down}}(F,F))^{\dagger}\boldsymbol{f}_F, \\ \boldsymbol{x}_C &= (\mathrm{Sc}(\boldsymbol{L}_2^{\mathrm{down}})_C)^{\dagger}\boldsymbol{h}, \\ \boldsymbol{x}_F &= (\boldsymbol{L}_2^{\mathrm{down}}(F,F))^{\dagger}\left(\boldsymbol{f}_F - \boldsymbol{L}_2^{\mathrm{down}}(F,C)\boldsymbol{x}_C\right). \end{aligned} \tag{6.5}$$

Let $\boldsymbol{x} = \begin{bmatrix}\boldsymbol{x}_C \\ \boldsymbol{x}_F\end{bmatrix}$. Applying Lemma 6.5.5 with $\delta = 0$, we have $\partial_2^\top\partial_2\boldsymbol{x} = \partial_2^\top\boldsymbol{b}$. That is, $\boldsymbol{x}$ can be written as $\boldsymbol{x} = \boldsymbol{x}_1 + \boldsymbol{x}_2$ where $\boldsymbol{x}_1 = (\partial_2^\top\partial_2)^{\dagger}\partial_2^\top\boldsymbol{b}$ and $\boldsymbol{x}_2$ is in $\mathrm{Ker}(\partial_2)$. Then,

$$\partial_2\boldsymbol{x} = \partial_2\boldsymbol{x}_1 = \boldsymbol{\Pi}_1^{\mathrm{up}}\boldsymbol{b}.$$

By Equation (6.5),

$$\boldsymbol{\Pi}_1^{\text{up}} \boldsymbol{b} = \partial_2 \boldsymbol{x} = \boldsymbol{\Pi}_{\text{Im}(\partial_2(:,F))} \boldsymbol{b} + \boldsymbol{\Pi}_{\text{Ker}(\partial_2^\top(F,:))} \partial_2(:,C)(\text{Sc}(\boldsymbol{L}_2^{\text{down}})_C)^\dagger \partial_2^\top(C,:) \boldsymbol{\Pi}_{\text{Ker}(\partial_2^\top(F,:))} \boldsymbol{b}.$$

Thus, the equation in the statement holds. By Fact 2.3.10,

$$\text{Sc}(\boldsymbol{L}_2^{\text{down}})_C = \partial_2^\top(C,:) \boldsymbol{\Pi}_{\text{Ker}(\partial_2^\top(F,:))} \partial_2(:,C).$$

The second matrix on the right-hand side of the equation in the statement is the orthogonal projection onto the image of $\boldsymbol{\Pi}_{\text{Ker}(\partial_2^\top(F,:))} \partial_2(:,C)$. □

To apply $\boldsymbol{\Pi}_1^{\text{up}}$ to a vector $\boldsymbol{b}$, we will need to (1) project $\boldsymbol{b}$ onto $\text{Im}(\partial_2(:,F))$, (2) project $\boldsymbol{b}$ onto $\text{Ker}(\partial_2^\top(F,:))$, and (3) solve a system in $\text{Sc}(\boldsymbol{L}_2^{\text{down}})_C$. We let F be the set of all the interior triangles of $\mathcal{K}$ w.r.t. the given r-hollowing, and let F be the set of all the boundary triangles. Similar to our up-Laplacian solver, we will apply the Nested Dissection for the "F" part and the Preconditioned Conjugate Gradient for the Schur complement onto C.

With a slight modification of the edge separator for a 3-complex (Corollary 6.7.4), we obtain a triangle separator for a 3-complex as the corollary below.

Corollary 6.8.2 (Triangle Separator for a 3-Complex). *Let $\mathcal{X}$ be a 3-complex that satisfies the requirements in Theorem 6.7.3 and has a connected 2-skeleton. Let p be the number of triangles in $\mathcal{X}$. Then, there exists an algorithm that removes $O(t^{2/3} + \bar{v})$ triangle in linear time. The remaining triangles can be divided into two sets, each containing at most cp triangles (where $c < 1$ is a constant), with no shared edges between the two sets.*

Proof. The proof is very similar to that of Corollary 6.7.4, by replacing edges with triangles. □

Given a triangle separator algorithm for a 3-complex, we can obtain a solver for $\boldsymbol{L}_2^{\text{down}}(F,F) = \partial_2^\top(F,:)\partial_2(:,F)$ by Nested Dissection. Note $\boldsymbol{L}_2^{\text{down}}[\sigma_1,\sigma_2] \neq 0$ if and only if two triangles σ_1, σ_2 share a common edge.

Lemma 6.8.3. *Let $\mathcal{X}$ be a 3-complex with $O(r)$ simplexes embedded in $\mathbb{R}^3$ such that (1) each tetrahderon of $\mathcal{X}$ has $O(1)$ aspect ratio and (2) $\mathcal{X}$ has $O(r^{2/3})$ exterior simplexes. Let F be the set of interior triangles of $\mathcal{X}$, and let $\boldsymbol{L}_{2,\mathcal{X}}^{down}$ be the second down-Laplacian of $\mathcal{X}$ and $\boldsymbol{M} \overset{\text{def}}{=} \boldsymbol{L}_{2,\mathcal{X}}^{down}(F,F)$. Then, there is a permutation matrix $\boldsymbol{P}$ and a lower triangular matrix $\boldsymbol{L}$ with $O(r^{4/3})$ non-zeros such that $\boldsymbol{M} = \boldsymbol{P}\boldsymbol{L}\boldsymbol{L}^\top\boldsymbol{P}^\top$. In addition, we can find such $\boldsymbol{P}$ and $\boldsymbol{L}$ in time $O(r^2)$. Given the above factorization, for any $\boldsymbol{b} \in Im(\boldsymbol{M})$, we can compute an $\boldsymbol{x}$ such that $\boldsymbol{M}\boldsymbol{x} = \boldsymbol{b}$ in $O(r^{4/3})$ time.*

Claim 6.8.4. *Given any 1-chain vector $\boldsymbol{b}$, we can compute $\boldsymbol{\Pi}_{Im(\partial_2(:,F))}\boldsymbol{b}$ and $\boldsymbol{\Pi}_{Ker(\partial_2^\top(F,:))}\boldsymbol{b}$ in time $O(nr)$.*

Proof. We have

$$\boldsymbol{\Pi}_{\text{Im}(\partial_2(:,F))}\boldsymbol{b} = \partial_2(:,F)\left(\partial_2^\top(F,:)\partial_2(:,F)\right)^\dagger \partial_2^\top(F,:)\boldsymbol{b}.$$

We can compute $\partial_2^\top(F,:)\boldsymbol{b}$ in time $O(n)$. To apply $\left(\partial_2^\top(F,:)\partial_2(:,F)\right)^\dagger$, we compute a Cholesky factorization of $\partial_2^\top(F,:)\partial_2(:,F)$ in time $O(r^2 \cdot \frac{n}{r}) = O(nr)$ via Nested Dissection

(Lemma 6.8.3), where the Cholesky factorization has $O(r^{4/3} \cdot \frac{n}{r}) = O(nr^{1/3})$ non-zeros; given such a Cholesky factorization, we can apply $\left(\partial_2^\top(F,:)\partial_2(:,F)\right)^\dagger$ to a vector in time $O(nr^{1/3})$. Then, multiplying $\partial_2(:,F)$ with a vector runs in linear time. So, the total runtime of computing $\boldsymbol{\Pi}_{\mathrm{Im}(\partial_2(:,F))}\boldsymbol{b}$ is $O(nr)$. Since $\boldsymbol{\Pi}_{\mathrm{Ker}(\partial_2^\top(F,:))} = \boldsymbol{I} - \boldsymbol{\Pi}_{\mathrm{Im}(\partial_2(:,F))}$, we can apply $\boldsymbol{\Pi}_{\mathrm{Ker}(\partial_2^\top(F,:))}$ to a vector in the same time up to a constant. □

It remains to bound the runtime of (approximately) solving a system in the Schur complement $\mathrm{Sc}(\boldsymbol{L}^{\mathrm{down}})_C$. We run the Preconditioned Conjugate Gradient and precondition the Schur complement by the boundary itself $\partial_2^\top(C,:)\partial_2(:,C)$.

6.8.1 Preconditioning the Schur Complement

In this section, we prove the following lemma for the relative condition number of the preconditioner for the Schur complement.

Lemma 6.8.5.

$$\kappa(Sc(\boldsymbol{L}_2^{down})_C, \partial_2^\top(C,:)\partial_2(:,C)) = O(r).$$

We will bound the relative condition number for each region and then combine them to get Lemma 6.8.5. Recall that in each region of an r-hollowing, the boundary triangulates a spherical shell in $\mathbb{R}^3$. We call the boundary triangles in a region containing an edge on the inner sphere of the boundary *the boundary layer*. Since each region boundary has its shell width of at least 5, boundary layers from different regions are disjoint. For the ith region, let $\boldsymbol{S}_i$ be the Schur complement of the interior triangles and the triangles in the boundary layer of the region i onto its boundary layer. We can write the Schur complement

$$\mathrm{Sc}(\boldsymbol{L}_2^{\mathrm{down}})_C = \mathrm{diag}(\boldsymbol{S}_1, \ldots, \boldsymbol{S}_k) + \boldsymbol{A},$$

where $\boldsymbol{A}$ is a PSD matrix. Similarly, let $\boldsymbol{M}_i$ be the boundary layer of the region i. We write the boundary

$$\partial_2^\top(C,:)\partial_2(:,C) = \mathrm{diag}(\boldsymbol{M}_1, \ldots, \boldsymbol{M}_k) + \boldsymbol{A}.$$

By an argument similar to Claim 6.7.7, proving Lemma 6.8.5 reduces to proving the following lemma.

Lemma 6.8.6. *For each i,*

$$\Omega(r^{-1})\boldsymbol{\Pi}_i\boldsymbol{M}_i\boldsymbol{\Pi}_i \preccurlyeq \boldsymbol{S}_i \preccurlyeq \boldsymbol{M}_i.$$

In the rest of this subsection, we prove Lemma 6.8.6. We locally use ∂_2 for the boundary operator *of the region* i and drop the region index i in the rest of this subsection. We write

$$\partial_2^\top = \begin{bmatrix} \boldsymbol{B}_{int} \\ \boldsymbol{B}_{bd} \end{bmatrix},$$

where the rows of $\boldsymbol{B}_{int}$ correspond to the interior triangles and the rows of $\boldsymbol{B}_{bd}$ the *boundary layer* triangles. The Schur complement onto the boundary layer triangles, by Fact 2.3.10,

$$\boldsymbol{S} = \boldsymbol{B}_{bd}\boldsymbol{\Pi}_{\mathrm{Ker}(\boldsymbol{B}_{int})}\boldsymbol{B}_{bd}^\top.$$

Let $\boldsymbol{M} \stackrel{\mathrm{def}}{=} \boldsymbol{B}_{bd}\boldsymbol{B}_{bd}^\top$, and let $\boldsymbol{\Pi}$ be the orthogonal projection onto the image of $\boldsymbol{S}$. Then, $\boldsymbol{S} \preccurlyeq \boldsymbol{M}$, and

$$\boldsymbol{S} \succcurlyeq \boldsymbol{\Pi}\boldsymbol{M}^{1/2}\boldsymbol{M}^{\dagger/2}\boldsymbol{S}\boldsymbol{M}^{\dagger/2}\boldsymbol{M}^{1/2}\boldsymbol{\Pi} \succcurlyeq \lambda_{\min}(\boldsymbol{M}^{\dagger/2}\boldsymbol{S}\boldsymbol{M}^{\dagger/2}) \cdot \boldsymbol{\Pi}\boldsymbol{M}\boldsymbol{\Pi}.$$

Claim 6.8.7.

$$\lambda_{\min}\left(\boldsymbol{M}^{\dagger/2}\boldsymbol{S}\boldsymbol{M}^{\dagger/2}\right) = \lambda_{\min}\left(\boldsymbol{\Pi}_{Im(\boldsymbol{B}_{bd}^\top)}\boldsymbol{\Pi}_{Ker(\boldsymbol{B}_{int})}\boldsymbol{\Pi}_{Im(\boldsymbol{B}_{bd}^\top)}\right).$$

Proof. We take the singular value decomposition: $\boldsymbol{B}_{bd} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top$, where the columns of $\boldsymbol{U}$ (resp., $\boldsymbol{V}$) form an orthonormal basis of $\mathrm{Im}(\boldsymbol{B}_{bd})$ (resp., $\mathrm{Im}(\boldsymbol{B}_{bd}^\top)$). Then,

$$\boldsymbol{M}^{\dagger/2}\boldsymbol{S}\boldsymbol{M}^{\dagger/2} = \boldsymbol{U}\boldsymbol{D}^{-1}\boldsymbol{U}^\top\boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top\boldsymbol{\Pi}_{\mathrm{Ker}(\boldsymbol{B}_{int})}\boldsymbol{V}\boldsymbol{D}\boldsymbol{U}^\top\boldsymbol{U}\boldsymbol{D}^{-1}\boldsymbol{U}^\top = \boldsymbol{U}\boldsymbol{V}^\top\boldsymbol{\Pi}_{\mathrm{Ker}(\boldsymbol{B}_{int})}\boldsymbol{V}\boldsymbol{U}^\top.$$

Since $\boldsymbol{U}$'s columns are orthonormal,

$$\lambda_{\min}(\boldsymbol{M}^{\dagger/2}\boldsymbol{S}\boldsymbol{M}^{\dagger/2}) = \lambda_{\min}(\boldsymbol{V}^\top\boldsymbol{\Pi}_{\mathrm{Ker}(\boldsymbol{B}_{int})}\boldsymbol{V}).$$

We claim

$$\lambda_{\min}(\boldsymbol{V}^\top\boldsymbol{\Pi}_{\mathrm{Ker}(\boldsymbol{B}_{int})}\boldsymbol{V}) = \lambda_{\min}(\boldsymbol{V}\boldsymbol{V}^\top\boldsymbol{\Pi}_{\mathrm{Ker}(\boldsymbol{B}_{int})}\boldsymbol{V}\boldsymbol{V}^\top), \tag{6.6}$$

which implies the claim statement. Let λ be an eigenvalue of $\boldsymbol{V}^\top\boldsymbol{\Pi}_{\mathrm{Ker}(\boldsymbol{B}_{int})}\boldsymbol{V}$ with eigenvector $\boldsymbol{u}$. Then,

$$\begin{aligned}\boldsymbol{V}^\top\boldsymbol{\Pi}_{\mathrm{Ker}(\boldsymbol{B}_{int})}\boldsymbol{V}\boldsymbol{u} = \lambda\boldsymbol{u} &\implies \boldsymbol{V}\boldsymbol{V}^\top\boldsymbol{\Pi}_{\mathrm{Ker}(\boldsymbol{B}_{int})}\boldsymbol{V}\boldsymbol{V}^\top\boldsymbol{V}\boldsymbol{u} = \lambda\boldsymbol{V}\boldsymbol{u}\\ &\implies \boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{B}_{bd}^\top)}\boldsymbol{\Pi}_{\mathrm{Ker}(\boldsymbol{B}_{int})}\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{B}_{bd}^\top)}\boldsymbol{V}\boldsymbol{u} = \lambda\boldsymbol{V}\boldsymbol{u}.\end{aligned}$$

That is, λ is an eigenvalue of $\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{B}_{bd}^\top)}\boldsymbol{\Pi}_{\mathrm{Ker}(\boldsymbol{B}_{int})}\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{B}_{bd}^\top)}$ with eigenvector $\boldsymbol{V}\boldsymbol{u}$. Let μ be an eigenvector of $\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{B}_{bd}^\top)}\boldsymbol{\Pi}_{\mathrm{Ker}(\boldsymbol{B}_{int})}\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{B}_{bd}^\top)}$ with eigenvector $\boldsymbol{w}$. Then,

$$\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{B}_{bd}^\top)}\boldsymbol{\Pi}_{\mathrm{Ker}(\boldsymbol{B}_{int})}\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{B}_{bd}^\top)}\boldsymbol{w} = \mu\boldsymbol{w} \implies \boldsymbol{V}^\top\boldsymbol{\Pi}_{\mathrm{Ker}(\boldsymbol{B}_{int})}\boldsymbol{V}\boldsymbol{V}^\top\boldsymbol{w} = \mu\boldsymbol{V}^\top\boldsymbol{w}.$$

That is, μ is an eigenvalue of $\boldsymbol{V}^\top\boldsymbol{\Pi}_{\mathrm{Ker}(\boldsymbol{B}_{int})}\boldsymbol{V}$ with eigenvector $\boldsymbol{V}^\top\boldsymbol{w}$. So, Equation (6.6) holds. □

Claim 6.8.8. *The image of* $\boldsymbol{\Pi}_{Im(\boldsymbol{B}_{bd}^\top)}\boldsymbol{\Pi}_{Ker(\boldsymbol{B}_{int})}\boldsymbol{\Pi}_{Im(\boldsymbol{B}_{bd}^\top)}$ *is*

$$\mathcal{U} = \{\boldsymbol{u} \in Im(\boldsymbol{B}_{bd}^\top) : \boldsymbol{u} \perp Im(\boldsymbol{B}_{bd}^\top) \cap Im(\boldsymbol{B}_{int}^\top)\}.$$

Proof. We first find an orthogonal basis of the kernel of $\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{B}_{bd}^\top)}\boldsymbol{\Pi}_{\mathrm{Ker}(\boldsymbol{B}_{int})}\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{B}_{bd}^\top)}$, which is $\mathcal{U}' \stackrel{\text{def}}{=} \{\boldsymbol{u} : \boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{B}_{bd}^\top)}\boldsymbol{\Pi}_{\mathrm{Ker}(\boldsymbol{B}_{int})}\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{B}_{bd}^\top)}\boldsymbol{u} = \boldsymbol{0}\} = \{\boldsymbol{u} : \boldsymbol{\Pi}_{\mathrm{Ker}(\boldsymbol{B}_{int})}\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{B}_{bd}^\top)}\boldsymbol{u} = \boldsymbol{0}\}$. Clearly, $\mathrm{Ker}(\boldsymbol{B}_{bd}) \subset \mathcal{U}'$. We consider $\boldsymbol{u} \in \mathrm{Im}(\boldsymbol{B}_{bd}^\top)$. In this case, $\boldsymbol{\Pi}_{\mathrm{Ker}(\boldsymbol{B}_{int})}\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{B}_{bd}^\top)}\boldsymbol{u} = \boldsymbol{\Pi}_{\mathrm{Ker}(\boldsymbol{B}_{int})}\boldsymbol{u} = \boldsymbol{0}$ if and only if $\boldsymbol{u} \in \mathrm{Im}(\boldsymbol{B}_{int}^\top)$. So, the claim statement holds. □

Lemma 6.8.9.

$$\lambda_{\min}\left(\boldsymbol{\Pi}_{Im(\boldsymbol{B}_{bd}^\top)}\boldsymbol{\Pi}_{Ker(\boldsymbol{B}_{int})}\boldsymbol{\Pi}_{Im(\boldsymbol{B}_{bd}^\top)}\right) = \Omega(r^{-1}).$$

Proof. By the Courant-Fischer min-max theorem and Claim 6.8.8,

$$\lambda_{\min}\left(\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{B}_{bd}^\top)}\boldsymbol{\Pi}_{\mathrm{Ker}(\boldsymbol{B}_{int})}\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{B}_{bd}^\top)}\right) = \min_{\boldsymbol{u}\in\mathcal{U}} \frac{\boldsymbol{u}^\top\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{B}_{bd}^\top)}\boldsymbol{\Pi}_{\mathrm{Ker}(\boldsymbol{B}_{int})}\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{B}_{bd}^\top)}\boldsymbol{u}}{\boldsymbol{u}^\top\boldsymbol{u}}, \tag{6.7}$$

where $\mathcal{U} = \{\boldsymbol{u} \in \mathrm{Im}(\boldsymbol{B}_{bd}^\top) : \boldsymbol{u} \perp \mathrm{Im}(\boldsymbol{B}_{bd}^\top) \cap \mathrm{Im}(\boldsymbol{B}_{int}^\top)\} \setminus \{\boldsymbol{0}\}$.

Recall that in each region, the boundary triangulates a spherical shell in $\mathbb{R}^3$. We further decompose ∂_2 according to the inner sphere of the boundary:

$$\partial_2 = \left[\boldsymbol{B}_{int}^\top \mid \boldsymbol{B}_{bd}^\top\right] = \begin{bmatrix} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} & | & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{B}_2 & | & \boldsymbol{B}_3 \\ \boldsymbol{0} & \boldsymbol{0} & | & \boldsymbol{B}_4 \end{bmatrix}. \tag{6.8}$$

Here, the blocks of the rows from top to bottom correspond to the interior edges, the boundary edges on the boundary inner sphere, and the other boundary edges, respectively; the blocks of the columns from left to right correspond to the interior triangles with only interior edges, the interior triangles with both interior and boundary edges, and the boundary triangles, respectively. The column in ∂_2 corresponding to a triangle whose edges are all on the boundary inner sphere can be written as a linear combination of the other boundary triangle columns. We remove these boundary inner sphere triangles so that every remaining triangle has at most one edge on the boundary inner sphere. Without loss of generality, we can orient and reorder the edges and the triangles so that

$$\boldsymbol{B}_2 = \operatorname{diag}(\boldsymbol{1}^\top, \ldots, \boldsymbol{1}^\top), \quad \boldsymbol{B}_3 = \operatorname{diag}(\boldsymbol{1}^\top, \ldots, \boldsymbol{1}^\top). \tag{6.9}$$

In addition, in each triangle with one edge on the boundary inner sphere, we let the other two edges point away from the inner sphere so that every column of $\boldsymbol{B}_{12}$ and of $\boldsymbol{B}_4$ has exactly two non-zero entries with values 1 and -1 each.

We want to characterize $\operatorname{Im}(\boldsymbol{B}_{int}^\top) \cap \operatorname{Im}(\boldsymbol{B}_{bd}^\top)$. Let

$$\mathcal{V} \stackrel{\text{def}}{=} \left\{ \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{B}_3 \boldsymbol{x} \\ \boldsymbol{0} \end{bmatrix} : \boldsymbol{x} \in \operatorname{Ker}(\boldsymbol{B}_4) \right\}.$$

By Equation (6.8), $\operatorname{Im}(\boldsymbol{B}_{int}^\top) \cap \operatorname{Im}(\boldsymbol{B}_{bd}^\top) \subseteq \mathcal{V}$. We will show $\mathcal{V} \subseteq \operatorname{Im}(\boldsymbol{B}_{int}^\top) \cap \operatorname{Im}(\boldsymbol{B}_{bd}^\top)$, that is, $\mathcal{V} \subseteq \operatorname{Im}(\boldsymbol{B}_{int}^\top)$. Since the boundary layer triangulates a spherical shell in $\mathbb{R}^3$, whose first Betti number is zero, $\mathcal{V}$ is orthogonal to the image of $\partial_1^\top$ of the boundary layer. Without loss of generality, we can also assume the 2-complex of the interior triangles touching the boundary inner sphere has the first Betti number being zero. Otherwise, we shift the boundary inner sphere towards the boundary outer sphere and include the boundary layer in the interior part. This can be done since, in the given r-hollowing, each region boundary triangulates a spherical shell with a "hop" shell width of at least 5. Under this assumption, $\mathcal{V} \subseteq \operatorname{Im}\left(\begin{bmatrix} \boldsymbol{B}_{12} \\ \boldsymbol{B}_2 \\ \boldsymbol{0} \end{bmatrix}\right) \subseteq \operatorname{Im}(\boldsymbol{B}_{int}^\top)$. Thus, $\mathcal{V} = \operatorname{Im}(\boldsymbol{B}_{int}^\top) \cap \operatorname{Im}(\boldsymbol{B}_{bd}^\top)$.

Then,

$$\mathcal{U} = \left\{ \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{B}_3 \boldsymbol{y} \\ \boldsymbol{B}_4 \boldsymbol{y} \end{bmatrix} : \boldsymbol{B}_3 \boldsymbol{y} \perp \boldsymbol{B}_3 \boldsymbol{x}, \ \forall \boldsymbol{x} \in \operatorname{Ker}(\boldsymbol{B}_4) \right\} \setminus \{\boldsymbol{0}\}.$$

By Equation (6.7) and (6.8),

$$\begin{aligned} \lambda_{\min}\left(\boldsymbol{\Pi}_{\operatorname{Im}(\boldsymbol{B}_{bd}^\top)} \boldsymbol{\Pi}_{\operatorname{Ker}(\boldsymbol{B}_{int})} \boldsymbol{\Pi}_{\operatorname{Im}(\boldsymbol{B}_{bd}^\top)}\right) &\geq \min_{\boldsymbol{y} \neq \boldsymbol{0} : \boldsymbol{B}_3^\top \boldsymbol{B}_3 \boldsymbol{y} \perp \operatorname{Ker}(\boldsymbol{B}_4)} \frac{\|\boldsymbol{B}_4 \boldsymbol{y}\|^2}{\|\boldsymbol{B}_3 \boldsymbol{y}\|^2 + \|\boldsymbol{B}_4 \boldsymbol{y}\|^2} \\ &= \min_{\boldsymbol{y} \neq \boldsymbol{0} : \boldsymbol{B}_3^\top \boldsymbol{B}_3 \boldsymbol{y} \perp \operatorname{Ker}(\boldsymbol{B}_4)} \frac{1}{\|\boldsymbol{B}_3 \boldsymbol{y}\|^2 / \|\boldsymbol{B}_4 \boldsymbol{y}\|^2 + 1}. \end{aligned}$$

It suffices to show $\frac{\|\boldsymbol{B}_3\boldsymbol{y}\|^2}{\|\boldsymbol{B}_4\boldsymbol{y}\|^2} = O(r)$.

Without loss of generality, we can assume each diagonal block of $\boldsymbol{B}_3$ in Equation (6.9) has equal dimensions c by duplicating columns in $\boldsymbol{B}_{bd}^\top$ (which does not change its image).

$$\boldsymbol{B}_3^\top \boldsymbol{B}_3 = \operatorname{diag}(\boldsymbol{J}, \dots, \boldsymbol{J}) \overset{\text{def}}{=} c\boldsymbol{\Pi}$$

is a multiple of projection matrix where $\boldsymbol{J}$ is the all-one matrix in dimensions $c \times c$. Since $\boldsymbol{B}_3^\top \boldsymbol{B}_3 \boldsymbol{y} \perp \operatorname{Ker}(\boldsymbol{B}_4)$, we know $\boldsymbol{B}_3^\top \boldsymbol{B}_3 \boldsymbol{y} = c\boldsymbol{\Pi}\boldsymbol{y} \in \operatorname{Im}(\boldsymbol{B}_4^\top)$. Since $\boldsymbol{\Pi}$ is an orthogonal projection, we have $\|\boldsymbol{\Pi}\boldsymbol{y}\| \le \left\|\boldsymbol{\Pi}_{\operatorname{Im}(\boldsymbol{B}_4^\top)}\boldsymbol{y}\right\|$. Note $\boldsymbol{B}_4^\top \boldsymbol{B}_4$ can be treated as a graph Laplacian matrix. By the eigenvalue bound in Theorem 6.7.13,

$$\|\boldsymbol{B}_4\boldsymbol{y}\|^2 \ge \Omega(r^{-1}) \left\|\boldsymbol{\Pi}_{\operatorname{Im}(\boldsymbol{B}_4^\top)}\boldsymbol{y}\right\|^2 \ge \Omega(r^{-1}) \|\boldsymbol{\Pi}\boldsymbol{y}\|^2 = \Omega(r^{-1}) \cdot \frac{1}{c} \|\boldsymbol{B}_3\boldsymbol{y}\|^2 .$$

Therefore,

$$\frac{\|\boldsymbol{B}_3\boldsymbol{y}\|^2}{\|\boldsymbol{B}_4\boldsymbol{y}\|^2} = O(r). \qquad \square$$

Combining all the lemmas and claims above, we prove Lemma 6.8.6.

6.8.2 Proof of Lemma 6.5.9

Given a vector $\boldsymbol{b}$, we approximate $\boldsymbol{\Pi}_1^{\text{up}}\boldsymbol{b}$ by the following steps: (1) compute $\boldsymbol{b}_1 = \boldsymbol{\Pi}_{\operatorname{Im}(\partial_2(:,F))}\boldsymbol{b}$; (2) compute $\boldsymbol{b}_2 = \partial_2^\top(C,:)\boldsymbol{\Pi}_{\operatorname{Ker}(\partial_2^\top(F,:))}\boldsymbol{b}$; (3) approximately solve $\operatorname{Sc}(\boldsymbol{L}_2^{\text{down}})_C \boldsymbol{b}_3 = \boldsymbol{b}_2$ via Preconditioned Conjugate Gradient and get an approximate solution $\widetilde{\boldsymbol{b}_3}$ up to error $\delta \le \frac{\epsilon}{\|\boldsymbol{L}_1^{\text{up}}\|}$; (4) compute $\boldsymbol{b}_4 = \boldsymbol{\Pi}_{\operatorname{Ker}(\partial_2^\top(F,:))}\partial_2(:,C)\widetilde{\boldsymbol{b}_3}$; (5) compute $\boldsymbol{b}_5 = \boldsymbol{b}_1 + \boldsymbol{b}_4$. Let $\widetilde{\boldsymbol{\Pi}}_1^{\text{up}}$ be the above operator so that $\boldsymbol{b}_5 = \widetilde{\boldsymbol{\Pi}}_1^{\text{up}}\boldsymbol{b}$.

Claim 6.8.10. *$\widetilde{\boldsymbol{\Pi}}_1^{up}\boldsymbol{b}$ is in the image of $\boldsymbol{L}_1^{up}$.*

Proof. By Lemma 6.8.1, the image of $\boldsymbol{L}_1^{\text{up}}$ can be decomposed as a direct sum of two orthogonal subspaces: $\operatorname{Im}(\partial_2(:,F))$ and $\operatorname{Im}(\boldsymbol{\Pi}_{\operatorname{Ker}(\partial_2^\top(F,:))}\partial_2(:,C))$. Note $\widetilde{\boldsymbol{\Pi}}_1^{\text{up}}\boldsymbol{b} = \boldsymbol{b}_1 + \boldsymbol{b}_4$, where $\boldsymbol{b}_1 \in \operatorname{Im}(\partial_2(:,F))$ and $\boldsymbol{b}_4 \in \operatorname{Im}(\boldsymbol{\Pi}_{\operatorname{Ker}(\partial_2^\top(F,:))}\partial_2(:,C))$. So, $\widetilde{\boldsymbol{\Pi}}_1^{\text{up}}\boldsymbol{b} \in \operatorname{Im}(\boldsymbol{L}_1^{\text{up}})$. $\square$

We bound the error of $\widetilde{\boldsymbol{\Pi}}_1^{\text{up}}\boldsymbol{b}$:

$$\begin{aligned}
\left\|\widetilde{\boldsymbol{\Pi}}_1^{\text{up}}\boldsymbol{b} - \boldsymbol{\Pi}_1^{\text{up}}\boldsymbol{b}\right\| &= \left\|\boldsymbol{\Pi}_{\operatorname{Ker}(\partial_2^\top(F,:))}\partial_2(:,C)\left((\operatorname{Sc}(\boldsymbol{L}_2^{\text{down}})_C)^\dagger - \boldsymbol{S}^\dagger\right)\partial_2^\top(C,:)\boldsymbol{\Pi}_{\operatorname{Ker}(\partial_2^\top(F,:))}\boldsymbol{b}\right\| \\
&\le \delta \left\|\partial_2^\top(C,:)\boldsymbol{\Pi}_{\operatorname{Ker}(\partial_2^\top(F,:))}\boldsymbol{b}\right\| \left\|\partial_2^\top(C,:)\boldsymbol{\Pi}_{\operatorname{Ker}(\partial_2^\top(F,:))}\right\| \\
&\le \delta\lambda_{\max}(\boldsymbol{L}_1^{\text{up}}) \|\boldsymbol{\Pi}_1^{\text{up}}\boldsymbol{b}\| \\
&\le \epsilon \|\boldsymbol{\Pi}_1^{\text{up}}\boldsymbol{b}\|, && \text{by the setting of } \delta
\end{aligned}$$

Finally, we bound the runtime of our approximate projection algorithm. The proof is similar to that of Lemma 6.5.8. By Theorem 6.3.4 and Lemma 6.8.5, the number of Preconditioned Conjugate Gradient (PCG) iterations is

$$\widetilde{O}\left(\kappa(\operatorname{Sc}(\boldsymbol{L}_2^{\text{down}})_C, \partial_2^\top(C,:)\partial_2(:,C))^{1/2}\right) = \widetilde{O}(r^{1/2}).$$

In each PCG iteration, we solve a system in $\partial_2^\top(C,:)\partial_2(:,C)$ and multiply $\mathrm{Sc}(\boldsymbol{L}_2^{\text{down}})_C$ with $O(1)$ vectors. Recall

$$\mathrm{Sc}(\boldsymbol{L}_2^{\text{down}})_C = \partial_2^\top(C,:)\boldsymbol{\Pi}_{\mathrm{Ker}(\partial_2^\top(F,:))}\partial_2(:,C).$$

In our preprocessing, we compute a Cholesky factorization of $\partial_2^\top(F,:)\boldsymbol{\Pi}_{\mathrm{Ker}(\partial_2^\top(F,:))}\partial_2(:,F)$ in time $O(nr)$ (by the proof of Claim 6.8.4); then, we can multiply $\mathrm{Sc}(\boldsymbol{L}_2^{\text{down}})_C$ onto a vector in time $O(nr^{1/3})$. Similarly, we solve a system in $\partial_2^\top(C,:)\partial_2(:,C)$ by nested dissection. By our construction, C has $O(r^{2/3}\cdot\frac{n}{r}) = O(nr^{-1/3})$ triangles. The Cholesky factorization runs in time $O(n^2r^{-2/3})$, and each system solve runs in time $O(n^{4/3}r^{-4/9})$. So, the runtime per PCG iteration is

$$O\left(nr^{1/3} + n^{4/3}r^{-4/9}\right).$$

Therefore, the total time is

$$\widetilde{O}\left(r^{1/2}\left(nr^{1/3} + n^{4/3}r^{-4/9}\right) + nr + n^2r^{-2/3}\right) = \widetilde{O}\left(nr + n^{4/3}r^{5/18} + n^2r^{-2/3}\right).$$

We finish the proof of Lemma 6.5.9.

6.9 Proof of the Main Theorem

Given all the four operators in Lemma 6.5.2, 6.5.3, 6.5.4, and 6.5.9, we prove the following main theorem.

Theorem 6.9.1 (Restatement of Theorem 6.4.1)**.** *Let $\mathcal{K}$ be a pure 3-complex embedded in $\mathbb{R}^3$ consisting of n stable simplexes and with a known r-hollowing. Let $\boldsymbol{L}_1$ be the 1-Laplacian operator of $\mathcal{K}$, and let $\boldsymbol{\Pi}_1$ be the orthogonal projection matrix onto the image of $\boldsymbol{L}_1$. For any vector $\boldsymbol{b}$ and $\epsilon > 0$, we can find a solution $\tilde{\boldsymbol{x}}$ such that $\|\boldsymbol{L}_1\tilde{\boldsymbol{x}} - \boldsymbol{\Pi}_1\boldsymbol{b}\|_2 \le \epsilon\|\boldsymbol{\Pi}_1\boldsymbol{b}\|_2$ in time $O\left(nr + n^{4/3}r^{5/18}\log(n/\epsilon) + n^2r^{-2/3}\right)$.*

Proof. Let κ be the maximum of $\kappa(\boldsymbol{L}_1^{\text{down}})$ and $\kappa(\boldsymbol{L}_1^{\text{up}})$. Let $\delta > 0$ be a parameter to be determined later. Let $\widetilde{\boldsymbol{\Pi}}_1^{\text{down}} = \widetilde{\boldsymbol{\Pi}}_1^{\text{down}}(\delta), \widetilde{\boldsymbol{\Pi}}_1^{\text{up}} = \widetilde{\boldsymbol{\Pi}}_1^{\text{up}}(\delta)$ be defined in Lemma 6.5.2 and 6.5.9, and let $\boldsymbol{Z}_1^{\text{down}}$ be the operator in Lemma 6.5.3 with no error and $\boldsymbol{Z}_1^{\text{up}}$ in Lemma 6.5.4 with error δ. Let

$$\begin{aligned}
&\widetilde{\boldsymbol{b}}^{\text{up}} \overset{\text{def}}{=} \widetilde{\boldsymbol{\Pi}}_1^{\text{up}}\boldsymbol{b}, \quad \widetilde{\boldsymbol{b}}^{\text{down}} \overset{\text{def}}{=} \widetilde{\boldsymbol{\Pi}}_1^{\text{down}}\boldsymbol{b},\\
&\widetilde{\boldsymbol{x}}^{\text{up}} \overset{\text{def}}{=} \boldsymbol{Z}_1^{\text{up}}\widetilde{\boldsymbol{b}}^{\text{up}}, \quad \widetilde{\boldsymbol{x}}^{\text{down}} \overset{\text{def}}{=} \boldsymbol{Z}_1^{\text{down}}\widetilde{\boldsymbol{b}}^{\text{down}},\\
&\tilde{\boldsymbol{x}} \overset{\text{def}}{=} \widetilde{\boldsymbol{\Pi}}_1^{\text{up}}\widetilde{\boldsymbol{x}}^{\text{up}} + \widetilde{\boldsymbol{\Pi}}_1^{\text{down}}\widetilde{\boldsymbol{x}}^{\text{down}}.
\end{aligned}$$

Then,

$$\begin{aligned}
&\|\boldsymbol{L}_1\tilde{\boldsymbol{x}} - \boldsymbol{\Pi}_1\boldsymbol{b}\|_2\\
&\quad\le \left\|\boldsymbol{L}_1^{\text{up}}\widetilde{\boldsymbol{\Pi}}_1^{\text{up}}\widetilde{\boldsymbol{x}}^{\text{up}} - \widetilde{\boldsymbol{b}}^{\text{up}}\right\|_2 + \left\|\boldsymbol{L}_1^{\text{down}}\widetilde{\boldsymbol{\Pi}}_1^{\text{down}}\widetilde{\boldsymbol{x}}^{\text{down}} - \widetilde{\boldsymbol{b}}^{\text{down}}\right\|_2 + \left\|\widetilde{\boldsymbol{b}}^{\text{up}} + \widetilde{\boldsymbol{b}}^{\text{down}} - \boldsymbol{\Pi}_1\boldsymbol{b}\right\|_2.
\end{aligned}$$

We will upper bound the three terms on the right-hand side separately.

- For the first term,
$$\begin{aligned}\left\|\boldsymbol{L}_1^{\mathrm{up}}\widetilde{\boldsymbol{\Pi}}_1^{\mathrm{up}}\widetilde{\boldsymbol{x}}^{\mathrm{up}} - \widetilde{\boldsymbol{b}}^{\mathrm{up}}\right\|_2 &\leq \left\|\boldsymbol{L}_1^{\mathrm{up}}\widetilde{\boldsymbol{\Pi}}_1^{\mathrm{up}}\widetilde{\boldsymbol{x}}^{\mathrm{up}} - \boldsymbol{L}_1^{\mathrm{up}}\boldsymbol{\Pi}_1^{\mathrm{up}}\widetilde{\boldsymbol{x}}^{\mathrm{up}}\right\|_2 + \left\|\boldsymbol{L}_1^{\mathrm{up}}\widetilde{\boldsymbol{x}}^{\mathrm{up}} - \widetilde{\boldsymbol{b}}^{\mathrm{up}}\right\|_2 \\ &\leq \left\|\boldsymbol{L}_1^{\mathrm{up}}\widetilde{\boldsymbol{\Pi}}_1^{\mathrm{up}}\widetilde{\boldsymbol{x}}^{\mathrm{up}} - \boldsymbol{L}_1^{\mathrm{up}}\boldsymbol{\Pi}_1^{\mathrm{up}}\widetilde{\boldsymbol{x}}^{\mathrm{up}}\right\|_2 + \delta\left\|\widetilde{\boldsymbol{b}}^{\mathrm{up}}\right\|, \quad \text{by Lemma 6.5.4}\end{aligned}$$
  By Lemma 6.5.9,
$$\begin{aligned}\left\|\boldsymbol{L}_1^{\mathrm{up}}\widetilde{\boldsymbol{\Pi}}_1^{\mathrm{up}}\widetilde{\boldsymbol{x}}^{\mathrm{up}} - \boldsymbol{L}_1^{\mathrm{up}}\boldsymbol{\Pi}_1^{\mathrm{up}}\widetilde{\boldsymbol{x}}^{\mathrm{up}}\right\|_2 &\leq \left\|\boldsymbol{L}_1^{\mathrm{up}}\right\|_2 \left\|(\widetilde{\boldsymbol{\Pi}}_1^{\mathrm{up}} - \boldsymbol{\Pi}_1^{\mathrm{up}})\boldsymbol{\Pi}_1^{\mathrm{up}}\widetilde{\boldsymbol{x}}^{\mathrm{up}}\right\|_2 \\ &\leq \delta\left\|\boldsymbol{L}_1^{\mathrm{up}}\right\|_2 \left\|\boldsymbol{\Pi}_1^{\mathrm{up}}\widetilde{\boldsymbol{x}}^{\mathrm{up}}\right\|_2.\end{aligned}$$
  Let $\boldsymbol{y} \stackrel{\mathrm{def}}{=} (\boldsymbol{L}_1^{\mathrm{up}})^{\dagger}\widetilde{\boldsymbol{b}}^{\mathrm{up}}$. By Lemma 6.5.4,
$$\left\|\boldsymbol{\Pi}_1^{\mathrm{up}}\widetilde{\boldsymbol{x}}^{\mathrm{up}} - \boldsymbol{y}\right\|_2 \leq \left\|(\boldsymbol{L}_1^{\mathrm{up}})^{\dagger}\right\|_2 \left\|\boldsymbol{L}_1^{\mathrm{up}}\boldsymbol{\Pi}_1^{\mathrm{up}}\widetilde{\boldsymbol{x}}^{\mathrm{up}} - \widetilde{\boldsymbol{b}}^{\mathrm{up}}\right\|_2 \leq \delta\left\|(\boldsymbol{L}_1^{\mathrm{up}})^{\dagger}\right\|_2 \left\|\widetilde{\boldsymbol{b}}^{\mathrm{up}}\right\|_2.$$
  By the triangle inequality,
$$\left\|\boldsymbol{\Pi}_1^{\mathrm{up}}\widetilde{\boldsymbol{x}}^{\mathrm{up}}\right\|_2 \leq \left\|\boldsymbol{y}\right\|_2 + \delta\left\|(\boldsymbol{L}_1^{\mathrm{up}})^{\dagger}\right\|_2 \left\|\widetilde{\boldsymbol{b}}^{\mathrm{up}}\right\|_2 \leq (1+\delta)\left\|(\boldsymbol{L}_1^{\mathrm{up}})^{\dagger}\right\|_2 \left\|\widetilde{\boldsymbol{b}}^{\mathrm{up}}\right\|_2.$$
  So,
$$\left\|\boldsymbol{L}_1^{\mathrm{up}}\widetilde{\boldsymbol{\Pi}}_1^{\mathrm{up}}\widetilde{\boldsymbol{x}}^{\mathrm{up}} - \boldsymbol{L}_1^{\mathrm{up}}\boldsymbol{\Pi}_1^{\mathrm{up}}\widetilde{\boldsymbol{x}}^{\mathrm{up}}\right\|_2 \leq \delta(1+\delta)\kappa\left\|\widetilde{\boldsymbol{b}}^{\mathrm{up}}\right\|_2,$$
  and
$$\left\|\boldsymbol{L}_1^{\mathrm{up}}\widetilde{\boldsymbol{\Pi}}_1^{\mathrm{up}}\widetilde{\boldsymbol{x}}^{\mathrm{up}} - \widetilde{\boldsymbol{b}}^{\mathrm{up}}\right\|_2 \leq 3\delta\kappa\left\|\widetilde{\boldsymbol{b}}^{\mathrm{up}}\right\|_2.$$

- For the second term, the operator $\boldsymbol{Z}_1^{\mathrm{down}}$ has no error, which means $\boldsymbol{L}_1^{\mathrm{down}}\widetilde{\boldsymbol{x}}^{\mathrm{down}} = \widetilde{\boldsymbol{b}}^{\mathrm{down}}$. Then,
$$\begin{aligned}\left\|\boldsymbol{L}_1^{\mathrm{down}}\widetilde{\boldsymbol{\Pi}}_1^{\mathrm{down}}\widetilde{\boldsymbol{x}}^{\mathrm{down}} - \widetilde{\boldsymbol{b}}^{\mathrm{down}}\right\|_2 &= \left\|\boldsymbol{L}_1^{\mathrm{down}}\widetilde{\boldsymbol{\Pi}}_1^{\mathrm{down}}\widetilde{\boldsymbol{x}}^{\mathrm{down}} - \boldsymbol{L}_1^{\mathrm{down}}\widetilde{\boldsymbol{x}}^{\mathrm{down}}\right\|_2 \\ &\leq \delta(1+\delta)\kappa\left\|\widetilde{\boldsymbol{b}}^{\mathrm{down}}\right\|_2.\end{aligned}$$

- For the third term,
$$\begin{aligned}\left\|\widetilde{\boldsymbol{b}}^{\mathrm{up}} + \widetilde{\boldsymbol{b}}^{\mathrm{down}} - \boldsymbol{\Pi}_1\boldsymbol{b}\right\|_2^2 &= \left\|(\widetilde{\boldsymbol{\Pi}}^{\mathrm{up}} - \boldsymbol{\Pi}^{\mathrm{up}})\boldsymbol{b}\right\|_2^2 + \left\|(\widetilde{\boldsymbol{\Pi}}^{\mathrm{down}} - \boldsymbol{\Pi}^{\mathrm{down}})\boldsymbol{b}\right\|_2^2 \\ &\leq \delta^2\left(\left\|\boldsymbol{\Pi}^{\mathrm{up}}\boldsymbol{b}\right\|_2^2 + \left\|\boldsymbol{\Pi}^{\mathrm{down}}\boldsymbol{b}\right\|_2^2\right) \\ &\qquad \text{by Lemma 6.5.2, 6.5.9, Fact 2.3.6} \\ &= \delta^2\left\|\boldsymbol{\Pi}_1\boldsymbol{b}\right\|_2^2.\end{aligned}$$

Combining all the above inequalities,
$$\begin{aligned}\left\|\boldsymbol{L}_1\widetilde{\boldsymbol{x}} - \boldsymbol{\Pi}_1\boldsymbol{b}\right\|_2 &\leq 3\delta\kappa\left\|\widetilde{\boldsymbol{b}}^{\mathrm{up}}\right\|_2 + 2\delta\kappa\left\|\widetilde{\boldsymbol{b}}^{\mathrm{down}}\right\|_2 + \delta\left\|\boldsymbol{\Pi}_1\boldsymbol{b}\right\|_2 \\ &\leq 3\delta\kappa(1+\delta)\left\|\boldsymbol{\Pi}_1^{\mathrm{up}}\boldsymbol{b}\right\|_2 + 2\delta\kappa(1+\delta)\left\|\boldsymbol{\Pi}_1^{\mathrm{down}}\boldsymbol{b}\right\|_2 + \delta\left\|\boldsymbol{\Pi}_1\boldsymbol{b}\right\|_2 \\ &\leq 11\delta\kappa\left\|\boldsymbol{\Pi}_1\boldsymbol{b}\right\|_2.\end{aligned}$$
Choosing $\delta \leq \frac{\epsilon}{11\kappa}$, we have
$$\left\|\boldsymbol{L}_1\widetilde{\boldsymbol{x}} - \boldsymbol{\Pi}_1\boldsymbol{b}\right\|_2 \leq \epsilon\left\|\boldsymbol{\Pi}_1\boldsymbol{b}\right\|_2.$$
The runtime bound follows Lemma 6.5.2, 6.5.3, 6.5.4, and 6.5.9. □

6.10 A Union of Pure 3-Complexes

In this section, we consider a union $\mathcal{U}$ of h pure 3-complex chunks $\mathcal{K}_1, \ldots, \mathcal{K}_h$ that are composed of stable simplexes, and these chunks are glued together by identifying certain subsets of their exterior simplexes. Let k be the number of simplexes shared by more than one chunk. We remark that $\mathcal{U}$ may *not* be embeddable in $\mathbb{R}^3$ and the first and second Betti numbers of $\mathcal{U}$ are *no longer zero*. This makes designing efficient solvers for 1-Laplacian systems of $\mathcal{U}$ much harder. Edelsbrunner and Parsa [EP14] showed that computing the first Betti number of a simplicial complex linearly embedded in $\mathbb{R}^4$ with m simplexes is as hard as computing the rank of a 0-1 matrix with $\Theta(m)$ non-zeros; Ding, Kyng, Probst Gutenberg, and Zhang [Din+22] showed that (approximately) solving 1-Laplacian systems for simplicial complexes in $\mathbb{R}^4$ is as hard as (approximately) solving general sparse systems of linear equations.

We design a 1-Laplacian solver for $\mathcal{U}$ whose runtime is comparable to that of the 1-Laplacian solver for a single pure 3-complex when both h and k are small and prove Theorem 6.4.2. The approximate down-projection operator for in Lemma 6.5.2 and the approximate solver for systems in the down-Laplacian in Lemma 6.5.3 hold for any simplicial complexes. Thus, it suffices to generalize the approximate up-projection operator in Lemma 6.5.9 and the approximate solver for systems in the up-Laplacian in Lemma 6.5.4 from a single chunk to a union of chunks.

Lemma 6.10.1 (Up-Laplacian Solver)**.** *Let $\mathcal{U}$ be a union of pure 3-complexes glued together by identifying certain subsets of their exterior simplexes. Each 3-complex chunk is embedded in $\mathbb{R}^3$, comprises n_i stable simplexes, and has a known $\Theta(n_i^{3/5})$-hollowing. For any $\boldsymbol{b} \in Im(\boldsymbol{L}_1^{up})$ and $\epsilon > 0$, we can compute an $\tilde{\boldsymbol{x}}$ such that $\|\boldsymbol{L}_1^{up}\tilde{\boldsymbol{x}} - \boldsymbol{b}\|_2 \leq \epsilon \|\boldsymbol{b}\|_2$ in time $\widetilde{O}\left(n^{8/5} + n^{3/10}k^2 + k^3\right)$ where n is the number of simplexes in $\mathcal{U}$ and k is the number of simplexes shared by more than one chunk.*

Lemma 6.10.2 (Up-Projection Operator)**.** *Let $\mathcal{U}$ be a 3-complex satisfying the conditions in Lemma 6.10.2. For any $\epsilon > 0$, there exists an operator $\widetilde{\boldsymbol{\Pi}}_1^{up}$ such that*

$$\forall \boldsymbol{b}, \qquad \left\|\widetilde{\boldsymbol{\Pi}}_1^{up}\boldsymbol{b} - \boldsymbol{\Pi}_1^{up}\boldsymbol{b}\right\|_2 \leq \epsilon \left\|\boldsymbol{\Pi}_1^{up}\boldsymbol{b}\right\|_2.$$

In addition, $\widetilde{\boldsymbol{\Pi}}_1^{up}\boldsymbol{b}$ can be computed in time $\widetilde{O}\left(n^{8/5} + n^{3/10}k^2 + k^3\right)$, where n is the number of exterior simplexes of $\mathcal{U}$ and k is the number of simplexes shared by more than one chunk.

Our approaches align with those for a single chunk. We partition the simplexes of $\mathcal{K}$ into $F \cup C$, then deal with the "F" part and the Schur complement separately. Given the definition of r-hollowing, the exterior simplexes of each chunk $\mathcal{K}_i$ must belong to the boundary of some region. We let C be the union of the hollowing boundary of each chunk and let F be the remaining simplexes. Then, F is a union of disjoint subcomplexes, each embedded in $\mathbb{R}^3$ and can be handled by Nested Dissection. We precondition the Schur complement by the union of the hollowing boundaries. However, systems in this preconditioner cannot be approximately solved Nested Dissection directly since it may not allow an embedding in $\mathbb{R}^3$. We will need a slightly more careful treatment.

Proof of Lemma 6.10.1. We let C be the union of the hollowing boundary edges in each chunk and let F be the union of the hollowing interior edges. Suppose $\mathcal{U}$ has h pure 3-complex chunks. By our definition of r-hollowing, we can write $\boldsymbol{L}_1^{\mathrm{up}}[F,F] =$

$\mathrm{diag}(\boldsymbol{L}_1^{\mathrm{up}}[F_1, F_1], \ldots, \boldsymbol{L}_1^{\mathrm{up}}[F_h, F_h])$ where F_i contains all the hollowing interior edges in the ith chunk. Let $r_i = n_i^{3/5}$ be the hollowing parameter for the ith chunk. By Lemma 6.5.7, with a pre-processing time

$$O\left(\sum_{i=1}^{h} n_i r_i\right) = O\left(\sum_{i=1}^{h} n_i^{8/5}\right) = O\left(n^{8/5}\right),$$

for any $\boldsymbol{b} \in \mathrm{Im}(\boldsymbol{L}_1^{\mathrm{up}}[F, F])$, we can find $\boldsymbol{x}$ such that $\boldsymbol{L}_1^{\mathrm{up}}[F, F]\boldsymbol{x} = \boldsymbol{b}$ in time

$$O\left(\sum_{i=1}^{h} n_i r_i^{1/3}\right) = O\left(\sum_{i=1}^{h} n_i^{6/5}\right) = O\left(n^{6/5}\right).$$

To solve the system in the Schur complement $\mathrm{Sc}[\boldsymbol{L}_1^{\mathrm{up}}]_C$, we precondition it by the union of the hollowing boundaries of each chunk, denoted by $\mathcal{T}_U$. By Claim 6.7.7 and Lemma 6.7.8, the relative condition number is $O(\max_i n_i^{3/5})$. Let $C_1 \subset C$ contain the edges shared by more than one chunk and $C_2 = C \setminus C_1$. Then, the submatrix $\boldsymbol{L}_{1,\mathcal{T}_U}^{\mathrm{up}}[C_2, C_2]$ is a block diagonal matrix where each block corresponds to a chunk. We solve $\boldsymbol{L}_{1,\mathcal{T}_U}^{\mathrm{up}}$ by Lemma 6.5.5: We solve a system in $\boldsymbol{L}_{1,\mathcal{T}_U}^{\mathrm{up}}[C_2, C_2]$ by Nested Dissection and solve the Schur complement onto C_1 by directly inverting the Schur complement. With a pre-processing time

$$O\left(\sum_{i=1}^{h}\left(\frac{n_i}{r_i} \cdot r_i^{2/3}\right)^2 + k^3\right) = O\left(\sum_{i=1}^{h} n_i^{8/5} + k^3\right) = O(n^{8/5} + k^3),$$

we can solve a system in $\boldsymbol{L}_{1,\mathcal{T}_U}^{\mathrm{up}}$ in time

$$O\left(\sum_{i=1}^{h}\left(\frac{n_i}{r_i} \cdot r_i^{2/3}\right)^{4/3} + k^2\right) = O\left(\sum_{i=1}^{h} n_i^{16/15} + k^2\right) = O(n^{16/15} + k^2).$$

Therefore, the total runtime of approximately solving a system in $\boldsymbol{L}_1^{\mathrm{up}}$ is $\widetilde{O}\left(n^{8/5} + n^{3/10}k^2 + k^3\right)$. $\square$

We can prove Lemma 6.10.2 by a similar argument.

Bibliography

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. "Network Flows". In: (1993).

[AW21] J. Alman and V. V. Williams. "A refined laser method and faster matrix multiplication". In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2021, pp. 522–539.

[Axe85] O. Axelsson. "A survey of preconditioned iterative methods for linear systems of algebraic equations". In: *BIT Numerical Mathematics* 25.1 (1985), pp. 165–187.

[AY10] N. Alon and R. Yuster. "Solving linear systems through nested dissection". In: *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*. IEEE. 2010, pp. 225–234.

[BHV08a] E. G. Boman, B. Hendrickson, and S. Vavasis. "Solving elliptic finite element systems in near-linear time with support preconditioners". In: *SIAM Journal on Numerical Analysis* 46.6 (2008), pp. 3264–3284.

[BHV08b] E. G. Boman, B. Hendrickson, and S. Vavasis. "Solving elliptic finite element systems in near-linear time with support preconditioners". In: *SIAM Journal on Numerical Analysis* 46.6 (2008), pp. 3264–3284.

[BKV09] C. Barnhart, N. Krishnan, and P. H. Vance. "Multicommodity Flow Problems". en. In: *Encyclopedia of Optimization*. Ed. by C. A. Floudas and P. M. Pardalos. Boston, MA: Springer US, 2009, pp. 2354–2362. ISBN: 978-0-387-74759-0.

[Bla+22] M. Black, W. Maxwell, A. Nayyeri, and E. Winkelman. "Computational Topology in a Collapsing Universe: Laplacians, Homology, Cohomology". In: *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2022, pp. 226–251.

[BN22] M. Black and A. Nayyeri. "Hodge Decomposition and General Laplacian Solvers for Embedded Simplicial Complexes". In: *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2022.

[Bro+17] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. "Geometric deep learning: going beyond euclidean data". In: *IEEE Signal Processing Magazine* 34.4 (2017), pp. 18–42.

[BS20] S. Barbarossa and S. Sardellitti. "Topological signal processing over simplicial complexes". In: *IEEE Transactions on Signal Processing* 68 (2020), pp. 2992–3007.

[BV21] M. Bafna and N. Vyas. “Optimal fine-grained hardness of approximation of linear equations”. In: *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*. Ed. by N. Bansal, E. Merelli, and J. Worrell. Vol. 198. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 20:1–20:19. ISBN: 978-3-95977-195-5.

[Car09] G. Carlsson. “Topology and Data”. In: *Bulletin of the American Mathematical Society* 46.2 (2009), pp. 255–308.

[Cha+16] F. Chazal, V. de Silva, M. Glisse, and S. Oudot. *The Structure and Stability of Persistence Modules*. Springer, 2016. ISBN: 978-3-319-42545-0.

[Che+20] L. Chen, G. Goranci, M. Henzinger, R. Peng, and T. Saranurak. “Fast Dynamic Cuts, Distances and Effective Resistances via Vertex Sparsifiers”. In: *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*. 2020, pp. 1135–1146.

[Che+22] L. Chen, R. Kyng, Y. P. Liu, R. Peng, M. P. Gutenberg, and S. Sachdeva. “Maximum flow and minimum-cost flow in almost-linear time”. In: *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2022, pp. 612–623.

[Chi67] D. R. Chillingworth. “Collapsing three-dimensional convex polyhedra”. In: *Mathematical Proceedings of the Cambridge Philosophical Society*. Vol. 63. Cambridge University Press. 1967, pp. 353–357.

[Chi80] D. R. Chillingworth. “Collapsing three-dimensional convex polyhedra: correction”. In: *Mathematical Proceedings of the Cambridge Philosophical Society*. Vol. 88. Cambridge University Press. 1980, pp. 307–310.

[Chr+11] P. Christiano, J. A. Kelner, A. Madry, D. A. Spielman, and S.-H. Teng. “Electrical Flows, Laplacian Systems, and Faster Approximation of Maximum Flow in Undirected Graphs”. In: *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*. 2011, pp. 273–282.

[CLS21] M. B. Cohen, Y. T. Lee, and Z. Song. “Solving Linear Programs in the Current Matrix Multiplication Time”. In: *Journal of the ACM (JACM)* 68.1 (2021), pp. 1–39.

[CMZ18] C. K. Chui, H. Mhaskar, and X. Zhuang. “Representation of functions on big data associated with directed graphs”. In: *Applied and Computational Harmonic Analysis* 44.1 (2018), pp. 165–188.

[Coh+14a] M. B. Cohen, B. T. Fasy, G. L. Miller, A. Nayyeri, R. Peng, and N. Walkington. “Solving 1-laplacians in nearly linear time: Collapsing and expanding a topological ball”. In: *Proceedings of the 2014 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2014, pp. 204–216.

[Coh+14b] M. B. Cohen, R. Kyng, G. L. Miller, J. W. Pachocki, R. Peng, A. B. Rao, and S. C. 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*. ACM. 2014, pp. 343–352.

[Coh+17a] M. B. Cohen, J. Kelner, J. Peebles, R. Peng, A. B. Rao, A. Sidford, and A. 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.* 2017, pp. 410–419.

[Coh+17b] M. B. Cohen, J. Kelner, J. Peebles, R. Peng, A. B. Rao, A. Sidford, and A. 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. Montreal, Canada: ACM, 2017, pp. 410–419. ISBN: 978-1-4503-4528-6.

[Coh+18] M. B. Cohen, J. Kelner, R. Kyng, J. Peebles, R. Peng, A. B. Rao, and A. 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).* IEEE. 2018, pp. 898–909.

[DDH07] J. Demmel, I. Dumitriu, and O. Holtz. “Fast Linear Algebra Is Stable”. In: *Numerische Mathematik* 108.1 (2007), pp. 59–91.

[Din+22] M. Ding, R. Kyng, M. P. Gutenberg, and P. Zhang. “Hardness results for laplacians of simplicial complexes via sparse-linear equation complete gadgets”. In: *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).* Ed. by M. Bojańczyk, E. Merelli, and D. P. Woodruff. Vol. 229. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 53:1–53:19. ISBN: 978-3-95977-235-8.

[Din70] E. A. Dinic. “Algorithm for Solution of a Problem of Maximum Flow in Networks with Power Estimation”. In: *Soviet Math. Doklady.* Vol. 11. 1970, pp. 1277–1280.

[DKM09] A. Duval, C. Klivans, and J. Martin. “Simplicial matrix-tree theorems”. In: *Transactions of the American Mathematical Society* 361.11 (2009), pp. 6073–6114.

[DKM15] A. M. Duval, C. J. Klivans, and J. L. Martin. “Cuts and flows of cell complexes”. In: *Journal of Algebraic Combinatorics* 41.4 (2015), pp. 969–999.

[DKT08] M. Desbrun, E. Kanso, and Y. Tong. “Discrete differential forms for computational modeling”. In: *Discrete differential geometry.* Springer, 2008, pp. 287–324.

[DKZ22] M. Ding, R. Kyng, and P. Zhang. “Two-Commodity Flow Is Equivalent to Linear Programming Under Nearly-Linear Time Reductions”. In: *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).* Ed. by M. Bojańczyk, E. Merelli, and D. P. Woodruff. Vol. 229. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 54:1–54:19. ISBN: 978-3-95977-235-8.

[DR80] D. P. Dobkin and S. P. Reiss. “The complexity of linear programming”. In: *Theoretical Computer Science* 11.1 (1980), pp. 1–18.

[DS07] S. I. Daitch and D. A. Spielman. “Support-Graph Preconditioners for 2-Dimensional Trusses”. In: *CoRR* abs/cs/0703119 (2007). arXiv: cs/0703119.

[DS08] S. I. Daitch and D. A. Spielman. “Faster approximate lossy generalized flow via interior point algorithms”. In: *Proceedings of the 40th annual ACM symposium on Theory of computing*. STOC ’08. Available at http://arxiv.org/abs/0803.0988. Victoria, British Columbia, Canada: ACM, 2008, pp. 451–460. ISBN: 978-1-60558-047-0.

[DW02] X. Dong and M. L. Wachs. “Combinatorial Laplacian of the matching complex”. In: *the electronic journal of combinatorics* (2002), R17–R17.

[DZ23a] J. van Den Brand and D. J. Zhang. “Faster high accuracy multi-commodity flow from single-commodity techniques”. In: *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2023, pp. 493–502.

[DZ23b] M. Ding and P. Zhang. “Efficient 1-Laplacian Solvers for Well-Shaped Simplicial Complexes: Beyond Betti Numbers and Collapsing Sequences”. In: *31st Annual European Symposium on Algorithms (ESA 2023)*. Ed. by I. L. Gørtz, M. Farach-Colton, S. J. Puglisi, and G. Herman. Vol. 274. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 41:1–41:19. ISBN: 978-3-95977-295-2.

[Eck44] B. Eckmann. “Harmonische Funktionen Und Randwertaufgaben in Einem Komplex”. In: *Commentarii Mathematici Helvetici* 17.1 (1944), pp. 240–255. ISSN: 1420-8946.

[EH10] H. Edelsbrunner and J. Harer. *Computational Topology: An Introduction*. American Mathematical Soc., 2010.

[EP14] H. Edelsbrunner and S. Parsa. “On the computational complexity of Betti numbers: reductions from matrix rank”. In: *Proceedings of the twenty-fifth annual ACM-SIAM symposium on discrete algorithms*. SIAM. 2014, pp. 152–160.

[ET75] S. Even and R. E. Tarjan. “Network Flow and Testing Graph Connectivity”. In: *SIAM journal on computing* 4.4 (1975), pp. 507–518.

[Eva76] J. R. Evans. “A Combinatorial Equivalence between A Class of Multicommodity Flow Problems and the Capacitated Transportation Problem”. en. In: *Mathematical Programming* 10.1 (1976), pp. 401–404. ISSN: 1436-4646.

[Eva78] J. R. Evans. “The Simplex Method for Integral Multicommodity Networks”. en. In: *Naval Research Logistics Quarterly* 25.1 (1978), pp. 31–37. ISSN: 00281441, 19319193.

[FF56] L. R. Ford and D. R. Fulkerson. “Maximal Flow through a Network”. In: *Canadian journal of Mathematics* 8 (1956), pp. 399–404.

[Fle00] L. K. Fleischer. “Approximating Fractional Multicommodity Flow Independent of the Number of Commodities”. In: *SIAM Journal on Discrete Mathematics* 13.4 (2000), pp. 505–520. ISSN: 0895-4801.

[Fri96] J. Friedman. “Computing Betti numbers via combinatorial Laplacians”. In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of Computing*. 1996, pp. 386–391.

[Fri98] J. Friedman. “Computing Betti numbers via combinatorial Laplacians”. In: *Algorithmica* 21.4 (1998), pp. 331–346.

[Geo73] A. George. “Nested dissection of a regular finite element mesh”. In: *SIAM Journal on Numerical Analysis* 10.2 (1973), pp. 345–363.

[Ghr08a] R. Ghrist. “Barcodes: The Persistent Topology of Data”. In: *Bulletin of the American Mathematical Society* 45.1 (2008), pp. 61–75.

[Ghr08b] R. Ghrist. “Barcodes: the persistent topology of data”. In: *Bulletin of the American Mathematical Society* 45.1 (2008), pp. 61–75.

[GK07] N. Garg and J. Könemann. “Faster and Simpler Algorithms for Multicommodity Flow and Other Fractional Packing Problems”. In: *SIAM Journal on Computing* 37.2 (2007), pp. 630–652. ISSN: 0097-5397.

[GLP21] Y. Gao, Y. P. Liu, and R. Peng. “Fully Dynamic Electrical Flows: Sparse Maxflow Faster Than Goldberg-Rao”. In: *arXiv:2101.07233 [cs]* (2021). arXiv: 2101.07233 [cs].

[GR98] A. V. Goldberg and S. Rao. “Beyond the Flow Decomposition Barrier”. In: *Journal of the ACM* 45.5 (1998), pp. 783–797. ISSN: 0004-5411.

[GV96] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 1996. ISBN: 978-0-8018-5414-9.

[HS+52] M. R. Hestenes, E. Stiefel, et al. *Methods of conjugate gradients for solving linear systems*. Vol. 49. 1. NBS Washington, DC, 1952.

[IP01] R. Impagliazzo and R. Paturi. “On the complexity of k-SAT”. In: *Journal of Computer and System Sciences* 62.2 (2001), pp. 367–375.

[Ita78] A. Itai. “Two-Commodity Flow”. In: *Journal of the ACM (JACM)* 25.4 (1978), pp. 596–611.

[Jia+11] X. Jiang, L.-H. Lim, Y. Yao, and Y. Ye. “Statistical ranking and combinatorial Hodge theory”. In: *Mathematical Programming* 127.1 (2011), pp. 203–244.

[Jia+21] S. Jiang, Z. Song, O. Weinstein, and H. Zhang. “A faster algorithm for solving general LPs”. In: *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*. 2021, pp. 823–832.

[JS21] A. Jambulapati and A. Sidford. “Ultrasparse ultrasparsifiers and faster laplacian system solvers”. In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2021, pp. 540–559.

[Kar84] N. Karmarkar. “A New Polynomial-Time Algorithm for Linear Programming”. In: *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*. 1984, pp. 302–311.

[Kel+13] J. A. Kelner, L. Orecchia, A. Sidford, and Z. A. Zhu. “A Simple, Combinatorial Algorithm for Solving SDD Systems in Nearly-Linear Time”. In: *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*. 2013, pp. 911–920.

[Kel+14] J. A. Kelner, Y. T. Lee, L. Orecchia, and A. 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*. SIAM, 2014, pp. 217–226.

[Ken78] J. L. Kennington. “A Survey of Linear Cost Multicommodity Network Flows”. In: *Operations Research* 26.2 (1978), pp. 209–236. ISSN: 0030-364X.

[Kha80a] L. G. Khachiyan. “Polynomial algorithms in linear programming”. In: *USSR Computational Mathematics and Mathematical Physics* 20.1 (1980), pp. 53–72.

[Kha80b] L. G. Khachiyan. “Polynomial Algorithms in Linear Programming”. In: *USSR Computational Mathematics and Mathematical Physics* 20.1 (1980), pp. 53–72.

[KLS20] T. Kathuria, Y. P. Liu, and A. Sidford. “Unit Capacity Maxflow in Almost $m^{4/3}$ Time”. In: *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2020, pp. 119–130.

[KMP10] I. Koutis, G. L. Miller, and R. Peng. “Approaching Optimality for Solving SDD Linear Systems”. In: *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*. FOCS ’10. USA: IEEE Computer Society, 2010, pp. 235–244. ISBN: 978-0-7695-4244-7.

[KMP11] I. Koutis, G. L. Miller, and R. Peng. “A Nearly-m log n Time Solver for SDD Linear Systems”. In: *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*. FOCS ’11. Available at http://arxiv.org/abs/1102.4842. Washington, DC, USA: IEEE Computer Society, 2011, pp. 590–598. ISBN: 978-0-7695-4571-4.

[KMT11] I. Koutis, G. L. Miller, and D. Tolliver. “Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing”. In: *Computer Vision and Image Understanding* 115.12 (2011), pp. 1638–1646.

[KS16] R. Kyng and S. Sachdeva. “Approximate gaussian elimination for laplacians-fast, sparse, and simple”. In: *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*. IEEE. 2016, pp. 573–582.

[KWZ20] R. Kyng, D. Wang, and P. 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*. SIAM, 2020, pp. 279–296.

[Kyn+16] R. Kyng, Y. T. Lee, R. Peng, S. Sachdeva, and D. 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. Cambridge, MA, USA: ACM, 2016, pp. 842–850. ISBN: 978-1-4503-4132-5.

[Kyn+18] R. Kyng, R. Peng, R. Schwieterman, and P. Zhang. “Incomplete nested dissection”. In: *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*. 2018, pp. 404–417.

[Kyn+19] R. Kyng, R. Peng, S. Sachdeva, and D. Wang. “Flows in almost linear time via adaptive preconditioning”. In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing.* STOC 2019. Phoenix, AZ, USA: Association for Computing Machinery, 2019, pp. 902–913. ISBN: 9781450367059.

[KZ17] R. Kyng and P. Zhang. “Hardness Results for Structured Linear Systems”. In: *SIAM Journal on Computing* 49.4 (2017), FOCS17–280.

[Lee+19] H. Lee, M. K. Chung, H. Choi, H. Kang, S. Ha, Y. K. Kim, and D. S. Lee. “Harmonic holes as the submodules of brain network and network dissimilarity”. In: *Computational Topology in Image Context: 7th International Workshop, CTIC 2019, Málaga, Spain, January 24-25, 2019, Proceedings 7.* Springer. 2019, pp. 110–122.

[Lei+95] T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos, and S. Tragoudas. “Fast Approximation Algorithms for Multicommodity Flow Problems”. en. In: *Journal of Computer and System Sciences* 50.2 (1995), pp. 228–243. ISSN: 0022-0000.

[Lim20] L.-H. Lim. “Hodge laplacians on graphs”. In: *Siam Review* 62.3 (2020), pp. 685–715.

[LRS13] Y. T. Lee, S. Rao, and N. Srivastava. “A New Approach to Computing Maximum Flows Using Electrical Flows”. In: *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing.* 2013, pp. 755–764.

[LRT79] R. J. Lipton, D. J. Rose, and R. E. Tarjan. “Generalized nested dissection”. In: *SIAM journal on numerical analysis* 16.2 (1979), pp. 346–358.

[LS13] Y. T. Lee and A. Sidford. “Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems”. In: *2013 ieee 54th annual symposium on foundations of computer science.* IEEE. 2013, pp. 147–156.

[LS14] Y. T. Lee and A. Sidford. “Path Finding Methods for Linear Programming: Solving Linear Programs in $\tilde{O}(\sqrt{rank})$ Iterations and Faster Algorithms for Maximum Flow”. In: *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on.* Available at http://arxiv.org/abs/1312.6677 and http://arxiv.org/abs/1312.6713. IEEE. 2014, pp. 424–433.

[LS20] Y. P. Liu and A. Sidford. “Faster Energy Maximization for Faster Maximum Flow”. In: *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing.* 2020, pp. 803–814.

[Ma+11a] W. Ma, J.-M. Morel, S. Osher, and A. Chien. “An L 1-based variational model for Retinex theory and its application to medical images”. In: *CVPR 2011.* IEEE. 2011, pp. 153–160.

[Ma+11b] W. Ma, J.-M. Morel, S. Osher, and A. Chien. “An L 1-based variational model for Retinex theory and its application to medical images”. In: *CVPR 2011.* IEEE. 2011, pp. 153–160.

[Mad10] A. Madry. “Faster Approximation Schemes for Fractional Multicommodity Flow Problems via Dynamic Graph Algorithms”. In: *Proceedings of the Forty-Second ACM Symposium on Theory of Computing.* STOC ’10. New York, NY, USA: Association for Computing Machinery, 2010, pp. 121–130. ISBN: 978-1-4503-0050-6.

[Mad13] A. Madry. “Navigating Central Path with Electrical Flows: From Flows to Matchings, and Back”. In: *Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. FOCS ’13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 253–262. ISBN: 978-0-7695-5135-7.

[Mad16] A. Madry. “Computing Maximum Flow with Augmenting Electrical Flows”. In: *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*. 2016, pp. 593–602.

[Mil+98] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis. “Geometric separators for finite-element meshes”. In: *SIAM Journal on Scientific Computing* 19.2 (1998), pp. 364–386.

[MN21] W. Maxwell and A. Nayyeri. “Generalized max-flows and min-cuts in simplicial complexes”. In: *arXiv preprint arXiv:2106.14116* (2021).

[Moh91] B. Mohar. “Eigenvalues, diameter, and mean distance in graphs”. In: *Graphs and combinatorics* 7.1 (1991), pp. 53–64.

[MT90] G. L. Miller and W. Thurston. “Separators in two and three dimensions”. In: *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. 1990, pp. 300–309.

[Mus+19] C. Musco, P. Netrapalli, A. Sidford, S. Ubaru, and D. P. Woodruff. “Spectrum Approximation Beyond Fast Matrix Multiplication: Algorithms and Hardness”. In: *arXiv:1704.04163 [cs, math]* (2019). arXiv: `1704.04163 [cs, math]`.

[Nie22] Z. Nie. “Matrix anti-concentration inequalities with applications”. In: *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. 2022, pp. 568–581.

[ODO13] B. Osting, J. Darbon, and S. Osher. “Statistical ranking using the l1-norm on graphs”. In: *AIMS Journal on Inverse Problems and Imaging* 7.3 (2013), pp. 907–926.

[OMV00] A. Ouorou, P. Mahey, and J.-P. Vial. “A Survey of Algorithms for Convex Multicommodity Flow Problems”. In: *Management Science* 46.1 (2000), pp. 126–147. ISSN: 0025-1909.

[Pen16] R. Peng. “Approximate undirected maximum flows in $O(m \operatorname{polylog}(n))$ time”. In: *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA ’16. Arlington, Virginia: Society for Industrial and Applied Mathematics, 2016, pp. 1862–1867. ISBN: 9781611974331.

[PS14] R. Peng and D. A. Spielman. “An efficient parallel solver for SDD linear systems”. In: *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*. ACM. 2014, pp. 333–342.

[PV21] R. Peng and S. Vempala. “Solving sparse linear systems faster than matrix multiplication”. In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2021, pp. 504–521.

[PW17] D. Průša and T. Werner. “LP relaxations of some NP-hard problems are as hard as any LP”. In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2017, pp. 1372–1382.

[Ren01] J. Renegar. *A Mathematical View of Interior-Point Methods in Convex Optimization*. Society for Industrial and Applied Mathematics, 2001.

[Ren88] J. Renegar. “A Polynomial-Time Algorithm, Based on Newton’s Method, for Linear Programming”. In: *Mathematical programming* 40.1 (1988), pp. 59–93.

[Ren95] J. Renegar. “Incorporating Condition Measures into the Complexity Theory of Linear Programming”. In: *SIAM Journal on Optimization* 5.3 (1995), pp. 506–524. ISSN: 1052-6234.

[Sav+25] A. Savostianov, M. T. Schaub, N. Guglielmi, and F. Tudisco. “Efficient Sparsification of Simplicial Complexes via Local Densities of States”. In: *arXiv preprint arXiv:2502.07558* (2025).

[Sch+20] M. T. Schaub, A. R. Benson, P. Horn, G. Lippner, and A. Jadbabaie. “Random walks on simplicial complexes and the normalized hodge 1-laplacian”. In: *SIAM Review* 62.2 (2020), pp. 353–391.

[She13] J. Sherman. “Nearly Maximum Flows in Nearly Linear Time”. In: *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. 2013, pp. 263–269.

[She17] J. Sherman. “Area-Convexity, L_∞ Regularization, and Undirected Multicommodity Flow”. In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. 2017, pp. 452–460.

[SS08] D. A. Spielman and N. Srivastava. “Graph sparsification by effective resistances”. In: *Proceedings of the fortieth annual ACM symposium on Theory of computing*. 2008, pp. 563–568.

[ST04] D. A. Spielman and S.-H. Teng. “Nearly-Linear Time Algorithms for Graph Partitioning, Graph Sparsification, and Solving Linear Systems”. In: *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*. STOC ’04. New York, NY, USA: Association for Computing Machinery, 2004, pp. 81–90. ISBN: 978-1-58113-852-8.

[ST08] G. Shklarski and S. Toledo. “Rigidity in finite-element matrices: Sufficient conditions for the rigidity of structures and substructures”. In: *SIAM Journal on Matrix Analysis and Applications* 30.1 (2008), pp. 7–40.

[ST14] D. A. Spielman and S.-H. Teng. “Nearly Linear Time Algorithms for Preconditioning and Solving Symmetric, Diagonally Dominant Linear Systems”. In: *SIAM Journal on Matrix Analysis and Applications* 35.3 (2014), pp. 835–885.

[Str69] V. Strassen. “Gaussian elimination is not optimal”. In: *Numerische mathematik* 13.4 (1969), pp. 354–356.

[Tan16] M. Tancer. “Recognition of collapsible complexes is NP-complete”. In: *Discrete & Computational Geometry* 55.1 (2016), pp. 21–38.

[Ten10] S.-H. Teng. “The Laplacian paradigm: Emerging algorithms for massive graphs”. In: *Theory and Applications of Models of Computation: 7th Annual Conference, TAMC 2010, Prague, Czech Republic, June 7-11, 2010. Proceedings 7*. Springer. 2010, pp. 2–14.

[Ton+03a] Y. Tong, S. Lombeyda, A. N. Hirani, and M. Desbrun. “Discrete multiscale vector field decomposition”. In: *ACM transactions on graphics (TOG)* 22.3 (2003), pp. 445–452.

[Ton+03b] Y. Tong, S. Lombeyda, A. N. Hirani, and M. Desbrun. “Discrete multiscale vector field decomposition”. In: *ACM transactions on graphics (TOG)* 22.3 (2003), pp. 445–452.

[TX98] L. Trevisan and F. Xhafa. “The parallel complexity of positive linear programming”. In: *Parallel Processing Letters* 8.04 (1998), pp. 527–533.

[Vai89] P. M. Vaidya. “Speeding-up Linear Programming Using Fast Matrix Multiplication”. In: *30th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 1989, pp. 332–337.

[van+21] J. van den Brand, Y. T. Lee, Y. P. Liu, T. Saranurak, A. Sidford, Z. Song, and D. Wang. “Minimum Cost Flows, MDPs, and L1-Regression in Nearly Linear Time for Dense Instances”. In: *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2021. New York, NY, USA: Association for Computing Machinery, 2021, pp. 859–869. ISBN: 978-1-4503-8053-9.

[Wan18] I.-L. Wang. “Multicommodity Network Flows: A Survey, Part I: Applications and Formulations”. In: *International Journal of Operations Research* 15.4 (2018), pp. 145–153.

[Wil05] R. Williams. “A new algorithm for optimal 2-constraint satisfaction and its implications”. In: *Theoretical Computer Science* 348.2-3 (2005), pp. 357–365.

[Wil18] V. V. Williams. “On some fine-grained questions in algorithms and complexity”. In: *Proceedings of the international congress of mathematicians: Rio de janeiro 2018*. World Scientific. 2018, pp. 3447–3487.

[WW18] V. V. Williams and R. R. Williams. “Subcubic Equivalences Between Path, Matrix, and Triangle Problems”. In: *Journal of the ACM* 65.5 (2018), 27:1–27:38. ISSN: 0004-5411.

[Xu+12] Q. Xu, Q. Huang, T. Jiang, B. Yan, W. Lin, and Y. Yao. “HodgeRank on random graphs for subjective video quality assessment”. In: *IEEE Transactions on Multimedia* 14.3 (2012), pp. 844–857.

[YL17a] K. Ye and L.-H. Lim. “Cohomology of cryo-electron microscopy”. In: *SIAM Journal on Applied Algebra and Geometry* 1.1 (2017), pp. 507–535.

[YL17b] K. Ye and L.-H. Lim. “Cohomology of cryo-electron microscopy”. In: *SIAM Journal on Applied Algebra and Geometry* 1.1 (2017), pp. 507–535.

[Zom05] A. J. Zomorodian. *Topology for Computing*. Vol. 16. Cambridge university press, 2005.

Appendix A

Missing Proofs of Linear Algebra Facts

Proof of Fact 2.1.1. By the definition of image, $\mathrm{Im}(\boldsymbol{A}) \supseteq \mathrm{Im}(\boldsymbol{A}\boldsymbol{A}^\top)$. It suffices to show $\mathrm{Im}(\boldsymbol{A}) \subseteq \mathrm{Im}(\boldsymbol{A}\boldsymbol{A}^\top)$. Let $\boldsymbol{x}$ be an arbitrary vector in $\mathrm{Im}(\boldsymbol{A})$. Then, $\boldsymbol{x} = \boldsymbol{A}\boldsymbol{y}$ for some $\boldsymbol{y} \in \mathbb{R}^n$. Write $\boldsymbol{y} = \boldsymbol{y}_1 + \boldsymbol{y}_2$ such that $\boldsymbol{y}_1 \in \mathrm{Im}(\boldsymbol{A}^\top)$ and $\boldsymbol{y}_2 \in \mathrm{Ker}(\boldsymbol{A})$. Let $\boldsymbol{z}$ satisfy $\boldsymbol{A}^\top \boldsymbol{z} = \boldsymbol{y}_1$. Then,

$$\boldsymbol{A}\boldsymbol{A}^\top \boldsymbol{z} = \boldsymbol{A}\boldsymbol{y}_1 = \boldsymbol{A}\boldsymbol{y} = \boldsymbol{x}.$$

Thus, $\boldsymbol{x} \in \mathrm{Im}(\boldsymbol{A}\boldsymbol{A}^\top)$. □

Proof of Fact 2.1.2. Since $\boldsymbol{A} \preccurlyeq \boldsymbol{B}$, by definition, $\boldsymbol{C} := \boldsymbol{B} - \boldsymbol{A}$ is PSD, i.e., for all $\boldsymbol{x} \in \mathbb{R}^n$, we have $\boldsymbol{x}^\top \boldsymbol{C}\boldsymbol{x} \geq 0$. Then, for any $\boldsymbol{y} \in \mathbb{R}^m$, let $\boldsymbol{x} = \boldsymbol{V}^\top \boldsymbol{y}$. Substituting this into the quadratic form, we get:

$$\boldsymbol{y}^\top (\boldsymbol{V}\boldsymbol{C}\boldsymbol{V}^\top)\boldsymbol{y} = (\boldsymbol{V}^\top \boldsymbol{y})^\top \boldsymbol{C}(\boldsymbol{V}^\top \boldsymbol{y}) = \boldsymbol{x}^\top \boldsymbol{C}\boldsymbol{x} \geq 0,$$

This shows that $\boldsymbol{V}\boldsymbol{C}\boldsymbol{V}^\top = \boldsymbol{V}(\boldsymbol{B} - \boldsymbol{A})\boldsymbol{V}^\top$ is PSD. Consequently, $\boldsymbol{V}\boldsymbol{A}\boldsymbol{V}^\top \preccurlyeq \boldsymbol{V}\boldsymbol{B}\boldsymbol{V}^\top$. □

Proof of Fact 2.3.2. To minimize $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2$, we compute the gradient with respect to $\boldsymbol{x}$ and set it to zero. This gives the normalized equation:

$$\boldsymbol{A}^\top (\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{b}) = \boldsymbol{0}.$$

Solving it gives $\boldsymbol{x}^* = (\boldsymbol{A}^\top \boldsymbol{A})^\dagger \boldsymbol{A}^\top \boldsymbol{b}$, thus

$$\boldsymbol{A}\boldsymbol{x}^* = \boldsymbol{A}(\boldsymbol{A}^\top \boldsymbol{A})^\dagger \boldsymbol{A}^\top \boldsymbol{b} = \boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\boldsymbol{b}.$$

Substituting $\boldsymbol{A}\boldsymbol{x}^* = \boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\boldsymbol{b}$ back to the ℓ_2 norm of residual gives

$$\|\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{b}\|_2^2 = \left\|(\boldsymbol{I} - \boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})})\boldsymbol{b}\right\|_2^2.$$

□

Proof of Fact 2.3.4. For the first equality, we have

$$\begin{aligned}
\left\|\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^\top \boldsymbol{b}\right\|_{(\boldsymbol{A}^\top \boldsymbol{A})^\dagger} &= (\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^\top \boldsymbol{b})^\top (\boldsymbol{A}^\top \boldsymbol{A})^\dagger (\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^\top \boldsymbol{b}) \\
&= (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})^\top \boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \\
&= (\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\boldsymbol{A}\boldsymbol{x} - \boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\boldsymbol{b})^\top (\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\boldsymbol{A}\boldsymbol{x} - \boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\boldsymbol{b}) \\
&= \left\|\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\boldsymbol{A}\boldsymbol{x} - \boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\boldsymbol{b}\right\|_2^2.
\end{aligned}$$

For the second equality, we have

$$\begin{aligned}
\|\boldsymbol{x}-\boldsymbol{x}^*\|_{\boldsymbol{A}^\top\boldsymbol{A}} &= (\boldsymbol{x}-\boldsymbol{x}^*)^\top\boldsymbol{A}^\top\boldsymbol{A}(\boldsymbol{x}-\boldsymbol{x}^*)\\
&= \boldsymbol{x}^\top\boldsymbol{A}^\top\boldsymbol{A}\boldsymbol{x} - 2\boldsymbol{x}^\top\boldsymbol{A}^\top\boldsymbol{A}\boldsymbol{x}^* + (\boldsymbol{x}^*)^\top\boldsymbol{A}^\top\boldsymbol{A}\boldsymbol{x}^*\\
&= \boldsymbol{x}^\top\boldsymbol{A}^\top\boldsymbol{A}\boldsymbol{x} - 2\boldsymbol{x}^\top\boldsymbol{A}^\top\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\boldsymbol{b} + \boldsymbol{b}^\top\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}^\top\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\boldsymbol{b}\\
&= \left\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\boldsymbol{b}\right\|_2.
\end{aligned}$$

□

Proof of Fact 2.3.5. We have

$$\begin{aligned}
\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_2^2 &\le \left\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\boldsymbol{b}\right\|_2^2 + \left\|(\boldsymbol{I}-\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})})\boldsymbol{b}\right\|_2^2\\
&= \left\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\boldsymbol{b}\right\|_2^2 + \|\boldsymbol{A}\boldsymbol{x}^*-\boldsymbol{b}\|_2^2 && \text{by Fact 2.3.2}\\
&\le \epsilon^2\left\|\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\boldsymbol{b}\right\|_2^2 + \|\boldsymbol{A}\boldsymbol{x}^*-\boldsymbol{b}\|_2^2, && \text{by Definition 2.3.3}
\end{aligned}$$

□

Proof of Fact 2.3.6. We start with proving the first statement. Since $\boldsymbol{A}$ is PSD, eigen-decomposition gives $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$, where $\boldsymbol{U}$ is an orthogonal matrix and $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1,\ldots,\lambda_n)$ with $\lambda_i \ge 0$. Since $\boldsymbol{b} \in \mathrm{Im}(\boldsymbol{A})$ and by Eq. (2.4), we have

$$\boldsymbol{b} = \boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\boldsymbol{b} = \boldsymbol{A}^{1/2}\boldsymbol{A}^{\dagger/2}\boldsymbol{b}.$$

Then we rewrite

$$\begin{aligned}
\|\boldsymbol{A}\boldsymbol{Z}\boldsymbol{b}-\boldsymbol{b}\|_2 &= \left\|\boldsymbol{A}^{1/2}\left(\boldsymbol{A}^{1/2}\boldsymbol{Z}\boldsymbol{A}^{1/2}-\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\right)\boldsymbol{A}^{\dagger/2}\boldsymbol{b}\right\|_2\\
&\le \left\|\boldsymbol{A}^{1/2}\left(\boldsymbol{A}^{1/2}\boldsymbol{Z}\boldsymbol{A}^{1/2}-\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\right)\boldsymbol{A}^{\dagger/2}\right\|_2\|\boldsymbol{b}\|_2\\
&= \left\|\boldsymbol{U}\boldsymbol{\Lambda}^{1/2}\boldsymbol{U}^\top\left(\boldsymbol{\Lambda}^{1/2}\boldsymbol{Z}\boldsymbol{\Lambda}^{1/2}-\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\right)\boldsymbol{U}\boldsymbol{\Lambda}^{-1/2}\boldsymbol{U}^\top\right\|_2\|\boldsymbol{b}\|_2\\
&= \left\|\boldsymbol{\Lambda}^{1/2}\boldsymbol{U}^\top\left(\boldsymbol{\Lambda}^{1/2}\boldsymbol{Z}\boldsymbol{\Lambda}^{1/2}-\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\right)\boldsymbol{U}\boldsymbol{\Lambda}^{-1/2}\right\|_2\|\boldsymbol{b}\|_2\\
&\qquad\qquad\text{since } \boldsymbol{U} \text{ is orthogonal}\\
&\le \left\|\boldsymbol{\Lambda}^{1/2}\right\|_2\left\|\boldsymbol{\Lambda}^{-1/2}\right\|_2\left\|\boldsymbol{A}^{1/2}\boldsymbol{Z}\boldsymbol{A}^{1/2}-\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\right\|_2\|\boldsymbol{b}\|_2\\
&= \frac{\sqrt{\lambda_{\max}}}{\sqrt{\lambda_{\min}}}\left\|\boldsymbol{A}^{1/2}\boldsymbol{Z}\boldsymbol{A}^{1/2}-\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\right\|_2\|\boldsymbol{b}\|_2\\
&\le \sqrt{\kappa(\boldsymbol{A})}\left\|\boldsymbol{A}^{1/2}\boldsymbol{Z}\boldsymbol{A}^{1/2}-\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\right\|_2\|\boldsymbol{b}\|_2\\
&\le \epsilon\sqrt{\kappa(\boldsymbol{A})}\,\|\boldsymbol{b}\|_2,
\end{aligned}$$

where the last step follows from the assumption $(1-\epsilon)\boldsymbol{A}^\dagger \preccurlyeq \boldsymbol{Z} \preccurlyeq (1+\epsilon)\boldsymbol{A}^\dagger$, and multiplying $\boldsymbol{A}^{1/2}$ on both sides gives

$$(1-\epsilon)\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})} \preccurlyeq \boldsymbol{A}^{1/2}\boldsymbol{Z}\boldsymbol{A}^{1/2} \preccurlyeq (1+\epsilon)\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})} \quad\Rightarrow\quad \left\|\boldsymbol{A}^{1/2}\boldsymbol{Z}\boldsymbol{A}^{1/2}-\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\right\|_2 \le \epsilon.$$

To prove the second statement, again, since $\boldsymbol{b} \in \mathrm{Im}(\boldsymbol{A})$, we can write $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x}$ for some $\boldsymbol{x}$:

$$\|\boldsymbol{A}\boldsymbol{Z}\boldsymbol{b}-\boldsymbol{b}\|_2 = \|\boldsymbol{A}\boldsymbol{Z}\boldsymbol{A}\boldsymbol{x}-\boldsymbol{A}\boldsymbol{x}\|_2 \le \epsilon\,\|\boldsymbol{A}\boldsymbol{x}\|_2.$$

Then for any $\boldsymbol{x}$,

$$(1-\epsilon)\left\|\boldsymbol{A}\boldsymbol{x}\right\|_2 \le \left\|\boldsymbol{A}\boldsymbol{Z}\boldsymbol{A}\boldsymbol{x}\right\|_2 \le (1+\epsilon)\left\|\boldsymbol{A}\boldsymbol{x}\right\|_2.$$

As a result,

$$(1-\epsilon)\boldsymbol{A} \preccurlyeq \boldsymbol{A}\boldsymbol{Z}\boldsymbol{A} \preccurlyeq (1+\epsilon)\boldsymbol{A}.$$

Multiplying $\boldsymbol{A}^{\dagger}$ on both sides gives

$$(1-\epsilon)\boldsymbol{A}^{\dagger}\boldsymbol{A}\boldsymbol{A}^{\dagger} \preccurlyeq \boldsymbol{A}^{\dagger}\boldsymbol{A}\boldsymbol{Z}\boldsymbol{A}\boldsymbol{A}^{\dagger} \preccurlyeq (1+\epsilon)\boldsymbol{A}^{\dagger}\boldsymbol{A}\boldsymbol{A}^{\dagger}.$$

By $\boldsymbol{A}^{\dagger}\boldsymbol{A}\boldsymbol{A}^{\dagger} = \boldsymbol{A}^{\dagger}$ and $\boldsymbol{A}^{\dagger}\boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^{\dagger} = \boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}$, we have

$$(1-\epsilon)\boldsymbol{A}^{\dagger} \preccurlyeq \boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\boldsymbol{Z}\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})} \preccurlyeq (1+\epsilon)\boldsymbol{A}^{\dagger}.$$

□

Proof of Fact 2.3.7. We expand the left hand side,

$$\begin{bmatrix}\boldsymbol{x}^{\top} & \boldsymbol{y}^{\top}\end{bmatrix}\boldsymbol{A}\begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{y}\end{bmatrix} = \boldsymbol{x}^{\top}\boldsymbol{A}(F,F)\boldsymbol{x} + 2\boldsymbol{x}^{\top}\boldsymbol{A}(F,C)\boldsymbol{y} + \boldsymbol{y}^{\top}\boldsymbol{A}(C,C)\boldsymbol{y}. \tag{A.1}$$

Taking the derivative w.r.t. $\boldsymbol{x}$ and setting it to be 0 gives that

$$2\boldsymbol{A}(F,F)\boldsymbol{x} + 2\boldsymbol{A}(F,C)\boldsymbol{y} = \boldsymbol{0}.$$

Plugging $\boldsymbol{x} = -\boldsymbol{A}(F,F)^{\dagger}\boldsymbol{A}(F,C)\boldsymbol{y}$ into Eq. (A.1),

$$\min_{\boldsymbol{x}}\begin{bmatrix}\boldsymbol{x}^{\top} & \boldsymbol{y}^{\top}\end{bmatrix}\boldsymbol{A}\begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{y}\end{bmatrix} = \boldsymbol{y}^{\top}\boldsymbol{A}(C,C)\boldsymbol{y} - \boldsymbol{y}^{\top}\boldsymbol{A}(F,C)\boldsymbol{A}(F,F)^{\dagger}\boldsymbol{A}(F,C)^{\top}\boldsymbol{y} = \boldsymbol{y}^{\top}\mathrm{Sc}(\boldsymbol{A})_C\boldsymbol{y}.$$

This completes the proof. □

Proof of Fact 2.3.9. We multiply the right-hand side:

$$\begin{aligned}
&\begin{pmatrix}\boldsymbol{I} & \\ \boldsymbol{A}[C,F]\boldsymbol{A}[F,F]^{\dagger} & \boldsymbol{I}\end{pmatrix}\begin{pmatrix}\boldsymbol{A}[F,F] & \\ & \mathrm{Sc}[\boldsymbol{A}]_C\end{pmatrix}\begin{pmatrix}\boldsymbol{I} & \boldsymbol{A}[F,F]^{\dagger}\boldsymbol{A}[F,C]\\ & \boldsymbol{I}\end{pmatrix}\\
=&\begin{pmatrix}\boldsymbol{I} & \\ \boldsymbol{A}[C,F]\boldsymbol{A}[F,F]^{\dagger} & \boldsymbol{I}\end{pmatrix}\begin{pmatrix}\boldsymbol{A}[F,F] & \boldsymbol{A}[F,C]\\ & \mathrm{Sc}[\boldsymbol{A}]_C\end{pmatrix}\\
=&\begin{pmatrix}\boldsymbol{A}[F,F] & \boldsymbol{A}[F,C]\\ \boldsymbol{A}[C,F] & \boldsymbol{A}[C,F]\boldsymbol{A}[F,F]^{\dagger}\boldsymbol{A}[F,C] + \mathrm{Sc}[\boldsymbol{A}]_C\end{pmatrix}\\
=&\begin{pmatrix}\boldsymbol{A}[F,F] & \boldsymbol{A}[F,C]\\ \boldsymbol{A}[C,F] & \boldsymbol{A}[C,C]\end{pmatrix} = \boldsymbol{A}.
\end{aligned}$$

□

Proof of Fact 2.3.10. By the definition of the Schur complement,

$$\begin{aligned}
\mathrm{Sc}[\boldsymbol{A}]_C &= \boldsymbol{B}_C\boldsymbol{B}_C^{\top} - \boldsymbol{B}_C\boldsymbol{B}_F^{\top}\left(\boldsymbol{B}_F\boldsymbol{B}_F^{\top}\right)^{\dagger}\boldsymbol{B}_F\boldsymbol{B}_C^{\top}\\
&= \boldsymbol{B}_C\left(\boldsymbol{I} - \boldsymbol{B}_F^{\top}\left(\boldsymbol{B}_F\boldsymbol{B}_F^{\top}\right)^{\dagger}\boldsymbol{B}_F\right)\boldsymbol{B}_C^{\top} = \boldsymbol{B}_C\boldsymbol{\Pi}_{\mathrm{Ker}(\boldsymbol{B}_F)}\boldsymbol{B}_C^{\top}.
\end{aligned}$$

□

Appendix B

Missing Proofs of Chapter 4

B.1 Reducing 2-Complex Boundary LE to Combinatorial Laplacian LE

In this section, we formally state Theorem 1.4.5 as below and provide a proof. Recall that we use $\sigma_{\min}(\boldsymbol{A})$ to denote the smallest non-zero eigenvalue.

Theorem B.1.1. *Let $\boldsymbol{L}_1 = \partial_1^\top \partial_1 + \partial_2 \partial_2^\top \in \mathbb{R}^{m \times m}$ be the combinatorial Laplacian of a 2-complex. Let $\boldsymbol{d} \in \mathbb{Z}^m$. Suppose we can solve* LEA $(\boldsymbol{L}_1, \boldsymbol{d}, \epsilon)$ *in time $\widetilde{O}(\mathrm{nnz}(\boldsymbol{L}_1)^c)$ where $c \geq 1$ is a constant. Then, we can solve* LEA $(\partial_2, \boldsymbol{d}, \delta)$ *in time $\widetilde{O}(\mathrm{nnz}(\partial_2)^c)$ by choosing*

$$\epsilon < \delta \frac{\sigma_{\min}(\boldsymbol{L}_1)^{1/2}}{\sigma_{\max}(\partial_2)^2} \frac{1}{\|\boldsymbol{d}\|_2}.$$

Proof. Suppose $\boldsymbol{x}_1$ satisfies

$$\left\|\boldsymbol{L}_1 \boldsymbol{x}_1 - \boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{L}_1)} \boldsymbol{d}\right\|_2 \leq \epsilon \left\|\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{L}_1)} \boldsymbol{d}\right\|_2.$$

By our assumption, we can compute $\boldsymbol{x}_1$ in time $\widetilde{O}(\mathrm{nnz}(\boldsymbol{L}_1)^c)$. We choose

$$\boldsymbol{f} = \partial_2^\top \boldsymbol{x}_1.$$

We claim that $\boldsymbol{f}$ solves LEA $(\partial_2, \boldsymbol{d}, \delta)$. Since $\partial_1 \partial_2 = \boldsymbol{0}$, we have $\boldsymbol{L}_1^\dagger = (\partial_1^\top \partial_1)^\dagger + (\partial_2 \partial_2^\top)^\dagger$ and $\boldsymbol{\Pi}_{\mathrm{Im}(\partial_2)} (\partial_1^\top \partial_1)^\dagger \boldsymbol{d} = \boldsymbol{0}$. Then,

$$\begin{aligned}
\left\|\partial_2 \boldsymbol{f} - \boldsymbol{\Pi}_{\mathrm{Im}(\partial_2)} \boldsymbol{d}\right\|_2 &= \left\|\boldsymbol{x}_1 - (\partial_2 \partial_2^\top)^\dagger \boldsymbol{d}\right\|_{(\partial_2 \partial_2^\top)^2} \\
&\leq \sigma_{\max}(\partial_2) \left\|\boldsymbol{x}_1 - (\partial_2 \partial_2^\top)^\dagger \boldsymbol{d}\right\|_{\partial_2 \partial_2^\top} \\
&= \sigma_{\max}(\partial_2) \left\|\boldsymbol{\Pi}_{\mathrm{Im}(\partial_2)} (\boldsymbol{x}_1 - (\partial_2 \partial_2^\top)^\dagger \boldsymbol{d})\right\|_{\partial_2 \partial_2^\top} \\
&= \sigma_{\max}(\partial_2) \left\|\boldsymbol{\Pi}_{\mathrm{Im}(\partial_2)} (\boldsymbol{x}_1 - \boldsymbol{L}_1^\dagger \boldsymbol{d})\right\|_{\partial_2 \partial_2^\top} \\
&= \sigma_{\max}(\partial_2) \left\|\boldsymbol{x}_1 - \boldsymbol{L}_1^\dagger \boldsymbol{d}\right\|_{\partial_2 \partial_2^\top} \\
&\leq \sigma_{\max}(\partial_2) \left\|\boldsymbol{x}_1 - \boldsymbol{L}_1^\dagger \boldsymbol{d}\right\|_{\boldsymbol{L}_1} \\
&\leq \frac{\sigma_{\max}(\partial_2)}{\sigma_{\min}(\boldsymbol{L}_1)^{1/2}} \left\|\boldsymbol{L}_1 \boldsymbol{x}_1 - \boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{L}_1)} \boldsymbol{d}\right\|_2
\end{aligned}$$

$$\leq \frac{\sigma_{\max}(\partial_2)}{\sigma_{\min}(\boldsymbol{L}_1)^{1/2}}\epsilon \left\|\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{L}_1)}\boldsymbol{d}\right\|_2$$
$$\leq \frac{\delta}{\sigma_{\max}(\partial_2)}$$
$$\leq \delta \left\|\boldsymbol{\Pi}_{\mathrm{Im}(\partial_2)}\boldsymbol{d}\right\|_2, \qquad \text{by Claim 4.6.3}$$

□

B.2 Connections With Interior Point Methods

We now show that in order to solve a generalized maxflow problem in a 2-complex flow network using an Interior Point Method (IPM), it suffices to be able to apply the pseudo-inverse of $\partial_2 \boldsymbol{W} \partial_2^\top$ for diagonal positive weight matrices $\boldsymbol{W}$ (and this problem is essentially equivalent to applying the pseudo-inverse of the combinatorial Laplacian of the complex, c.f. Appendix B.1). We sketch how these pseudo-inverse problems arise when solving a generalized maxflow using IPM, which is motivated by [Mad16]. For the more curious readers, we recommend the book [Ren01] for a complete view of general IPM algorithms.

Given a 2-complex flow network $\mathcal{K}$ with m edges and t triangles, a non-negative capacity vector $\boldsymbol{c} \in \mathbb{R}^t_{\geq 0}$, and a demand vector $\gamma \in \mathbb{R}^m$ such that $\gamma \in \mathrm{Im}(\partial_2)$. The γ-maxflow problem is formulated by the following linear programming:

$$\begin{aligned} \max_{F,\boldsymbol{f}} \quad & F \\ \text{s.t.} \quad & \partial_2 \boldsymbol{f} = F\gamma \\ & -\boldsymbol{c} \leq \boldsymbol{f} \leq \boldsymbol{c} \end{aligned} \tag{B.1}$$

The γ-maxflow in 2-complex flow networks is a generalization of s-t maxflow in graphs. The first constraint encodes the conservation of flows for edges in $\mathcal{K}$. And the second constraint forces the flow on triangles to satisfy the capacity constraints.

We call F the flow value of $\boldsymbol{f}$ when $\partial_2 \boldsymbol{f} = F\gamma$. We assume that the optimal flow value F^* is known by IPM algorithms, which can be estimated by the binary search.

The main idea of IPM is to get rid of inequality constraints by using barrier functions, and then apply Newton's method to a sequence of equality constrained problems. The most widely used barrier function is the logarithmic barrier function, which in the γ-maxflow problem gives

$$V(\boldsymbol{f}) = \sum_{\Delta \in [t]} -\log(\boldsymbol{c}(\Delta) - \boldsymbol{f}(\Delta)) - \log(\boldsymbol{c}(\Delta) + \boldsymbol{f}(\Delta)).$$

Then for a given $0 \leq \alpha < 1$, we define the following Barrier Problem:

$$\begin{aligned} \min_{\boldsymbol{f}} \quad & V(\boldsymbol{f}) \\ \text{s.t.} \quad & \partial_2 \boldsymbol{f} = \alpha F^* \gamma \end{aligned} \tag{B.2}$$

We start with zero flow, i.e., $\alpha_0 = 0$, and then increase $\alpha_{i+1} = \alpha_i + \alpha'$ gradually in each iteration to make progress. Given a small enough α', each iteration is composed of a *progress step* and a *centering step*.

Progress Step. We first take a progress step by making a Newton step to Problem (B.2) at the current point $\boldsymbol{f}$, while increasing the flow value by α', which gives

$$\begin{aligned} \min_{\boldsymbol{\delta}} \quad & \boldsymbol{g}^\top(\boldsymbol{f})\boldsymbol{\delta} + \frac{1}{2}\boldsymbol{\delta}^\top \boldsymbol{H}(\boldsymbol{f})\boldsymbol{\delta} \\ \text{s.t.} \quad & \partial_2 \boldsymbol{\delta} = \alpha' F^* \gamma \end{aligned} \tag{B.3}$$

where $\boldsymbol{g}(\boldsymbol{f})$ and $\boldsymbol{H}(\boldsymbol{f})$ are the gradient and Hessian of V at the current point $\boldsymbol{f}$, respectively.

Problem (B.3) has the Lagrangian

$$\mathcal{L}(\boldsymbol{\delta}, \boldsymbol{x}) = \boldsymbol{g}^\top(\boldsymbol{f})\boldsymbol{\delta} + \frac{1}{2}\boldsymbol{\delta}^\top \boldsymbol{H}(\boldsymbol{f})\boldsymbol{\delta} + \boldsymbol{x}^\top(\alpha' F^* \gamma - \partial_2 \boldsymbol{\delta}).$$

Using the optimality condition, we have

$$\nabla_{\boldsymbol{\delta}} \mathcal{L}(\boldsymbol{\delta}, \boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{f}) + \boldsymbol{H}(\boldsymbol{f})\boldsymbol{\delta} - \partial_2^\top \boldsymbol{x} = \boldsymbol{0},$$

which gives

$$\boldsymbol{\delta} = \boldsymbol{H}^{-1}(\boldsymbol{f})(\partial_2^\top \boldsymbol{x} - \boldsymbol{g}(\boldsymbol{f})).$$

Multiplying ∂_2 in both sides and using the constraint $\partial_2 \boldsymbol{\delta} = \alpha' F^* \gamma$, we obtain

$$\partial_2 \boldsymbol{H}^{-1}(\boldsymbol{f})\partial_2^\top \boldsymbol{x} = \partial_2 \boldsymbol{H}^{-1}(\boldsymbol{f})\boldsymbol{g}(\boldsymbol{f}) + \alpha' F^* \gamma.$$

Thus, we have shown that it suffices to apply the pseudo-inverse of $\partial_2 \boldsymbol{H}^{-1}(\boldsymbol{f})\partial_2^\top$ to solve $\boldsymbol{x}$ and $\boldsymbol{\delta}$:

$$\boldsymbol{x} = \left(\partial_2 \boldsymbol{H}^{-1}(\boldsymbol{f})\partial_2^\top\right)^\dagger (\partial_2 \boldsymbol{H}^{-1}(\boldsymbol{f})\boldsymbol{g}(\boldsymbol{f}) + \alpha' F^* \gamma),$$

$$\boldsymbol{\delta} = \boldsymbol{H}^{-1}(\boldsymbol{f})\partial_2^\top \left(\partial_2 \boldsymbol{H}^{-1}(\boldsymbol{f})\partial_2^\top\right)^\dagger \left(\partial_2 \boldsymbol{H}^{-1}(\boldsymbol{f})\boldsymbol{g}(\boldsymbol{f}) + \alpha' F^* \gamma\right) - \boldsymbol{H}^{-1}(\boldsymbol{f})\boldsymbol{g}(\boldsymbol{f}).$$

Centering Step. We then take a centering step by making a Newton step to Problem (B.2) at the updated point of $\tilde{\boldsymbol{f}} \overset{\text{def}}{=} \boldsymbol{f} + \boldsymbol{\delta}$ without increasing the flow value, which gives

$$\begin{aligned} \min_{\tilde{\boldsymbol{\delta}}} \quad & \boldsymbol{g}^\top(\tilde{\boldsymbol{f}})\tilde{\boldsymbol{\delta}} + \frac{1}{2}\tilde{\boldsymbol{\delta}}^\top \boldsymbol{H}(\tilde{\boldsymbol{f}})\tilde{\boldsymbol{\delta}} \\ \text{s.t.} \quad & \partial_2 \tilde{\boldsymbol{\delta}} = \boldsymbol{0} \end{aligned} \tag{B.4}$$

Similar to the progress step, it suffices to apply the pseudo-inverse of $\partial_2 \boldsymbol{H}^{-1}(\tilde{\boldsymbol{f}})\partial_2^\top$ to solve $\tilde{\boldsymbol{\delta}}$:

$$\tilde{\boldsymbol{\delta}} = \left(\boldsymbol{H}^{-1}(\tilde{\boldsymbol{f}})\partial_2^\top \left(\partial_2 \boldsymbol{H}^{-1}(\tilde{\boldsymbol{f}})\partial_2^\top\right)^\dagger \partial_2 - \boldsymbol{I}\right) \boldsymbol{H}^{-1}(\tilde{\boldsymbol{f}})\boldsymbol{g}(\tilde{\boldsymbol{f}}).$$

Appendix C

Missing Proofs of Chapter 6

Proof of Lemma 6.5.5. Since $\boldsymbol{b} \in \mathrm{Im}(\boldsymbol{L}_1^{\mathrm{up}})$, we know $\boldsymbol{b}_F \in \mathrm{Im}(\boldsymbol{L}_1^{\mathrm{up}}(F,F))$ and $\boldsymbol{b}_F - \boldsymbol{L}_1^{\mathrm{up}}(F,C)\tilde{\boldsymbol{x}}_C \in \mathrm{Im}(\boldsymbol{L}_1^{\mathrm{up}}(F,F))$. We can apply the solver UpLapFSolver to these two vectors. By the statement assumption, we an write $\textsc{UpLapFSolver}(\boldsymbol{b}_F) = \boldsymbol{L}_1^{\mathrm{up}}(F,F)^{\dagger}\boldsymbol{b}_F + \boldsymbol{y}$, where $\partial_2[F,:]^{\top}\boldsymbol{y} = \boldsymbol{0}$. Then,

$$\boldsymbol{h} = \boldsymbol{b}_C - \boldsymbol{L}_1^{\mathrm{up}}(C,F)\cdot\textsc{UpLapFSolver}(\boldsymbol{b}_F) = \boldsymbol{b}_C - \boldsymbol{L}_1^{\mathrm{up}}(C,F)\cdot\boldsymbol{L}_1^{\mathrm{up}}(F,F)^{\dagger}\boldsymbol{b}_F.$$

We first show that $\boldsymbol{h} \in \mathrm{Im}(\mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_C)$ so that we can apply the solver SchurSolver to $\boldsymbol{h}$ and obtain a vector $\tilde{\boldsymbol{x}}_C$ satisfying $\|\mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_C\tilde{\boldsymbol{x}}_C - \boldsymbol{h}\|_2 \le \delta\,\|\boldsymbol{h}\|_2$. Since $\boldsymbol{b} \in \mathrm{Im}(\boldsymbol{L}_1^{\mathrm{up}})$, there exists $\boldsymbol{x} = \begin{bmatrix}\boldsymbol{x}_F\\ \boldsymbol{x}_C\end{bmatrix}$ such that

$$\begin{aligned}
\begin{bmatrix}\boldsymbol{b}_F\\ \boldsymbol{b}_C\end{bmatrix} &= \begin{bmatrix}\boldsymbol{I} & \\ \boldsymbol{L}_1^{\mathrm{up}}(C,F)\boldsymbol{L}_1^{\mathrm{up}}(F,F)^{\dagger} & \boldsymbol{I}\end{bmatrix}\begin{bmatrix}\boldsymbol{L}_1^{\mathrm{up}}(F,F) & \\ & \mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_C\end{bmatrix}\begin{bmatrix}\boldsymbol{I} & \boldsymbol{L}_1^{\mathrm{up}}(F,F)^{\dagger}\boldsymbol{L}_1^{\mathrm{up}}(F,C)\\ & \boldsymbol{I}\end{bmatrix}\begin{bmatrix}\boldsymbol{x}_F\\ \boldsymbol{x}_C\end{bmatrix}\\
&= \begin{bmatrix}\boldsymbol{I} & \\ \boldsymbol{L}_1^{\mathrm{up}}(C,F)\boldsymbol{L}_1^{\mathrm{up}}(F,F)^{\dagger} & \boldsymbol{I}\end{bmatrix}\begin{bmatrix}\boldsymbol{L}_1^{\mathrm{up}}(F,F) & \\ & \mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_C\end{bmatrix}\begin{bmatrix}\boldsymbol{x}_F + \boldsymbol{L}_1^{\mathrm{up}}(F,F)^{\dagger}\boldsymbol{L}_1^{\mathrm{up}}(F,C)\boldsymbol{x}_C\\ \boldsymbol{x}_C\end{bmatrix}\\
&= \begin{bmatrix}\boldsymbol{I} & \\ \boldsymbol{L}_1^{\mathrm{up}}(C,F)\boldsymbol{L}_1^{\mathrm{up}}(F,F)^{\dagger} & \boldsymbol{I}\end{bmatrix}\begin{bmatrix}\boldsymbol{L}_1^{\mathrm{up}}(F,F)\boldsymbol{x}_F + \boldsymbol{L}_1^{\mathrm{up}}(F,C)\boldsymbol{x}_C\\ \mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_C\boldsymbol{x}_C\end{bmatrix}\\
&= \begin{bmatrix}\boldsymbol{L}_1^{\mathrm{up}}(F,F)\boldsymbol{x}_F + \boldsymbol{L}_1^{\mathrm{up}}(F,C)\boldsymbol{x}_C\\ \boldsymbol{L}_1^{\mathrm{up}}(C,F)\boldsymbol{x}_F + \boldsymbol{L}_1^{\mathrm{up}}(C,F)\boldsymbol{L}_1^{\mathrm{up}}(F,F)^{\dagger}\boldsymbol{L}_1^{\mathrm{up}}(F,C)\boldsymbol{x}_C + \mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_C\boldsymbol{x}_C\end{bmatrix}.
\end{aligned} \tag{C.1}$$

Here, the third equality holds since

$$\boldsymbol{L}_1^{\mathrm{up}}(F,F)\boldsymbol{L}_1^{\mathrm{up}}(F,F)^{\dagger}\boldsymbol{L}_1^{\mathrm{up}}(F,C) = \boldsymbol{\Pi}_{\mathrm{Im}(\partial_2[F,:])}\boldsymbol{L}_1^{\mathrm{up}}(F,C) = \boldsymbol{L}_1^{\mathrm{up}}(F,C),$$

and the fourth equality holds similarly by using symmetry. Thus,

$$\begin{aligned}
\boldsymbol{b}_C &= \boldsymbol{L}_1^{\mathrm{up}}(C,F)\boldsymbol{x}_F + \boldsymbol{L}_1^{\mathrm{up}}(C,F)\boldsymbol{L}_1^{\mathrm{up}}(F,F)^{\dagger}\left(\boldsymbol{b}_F - \boldsymbol{L}_1^{\mathrm{up}}(F,F)\boldsymbol{x}_F\right) + \mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_C\boldsymbol{x}_C\\
&= \boldsymbol{L}_1^{\mathrm{up}}(C,F)\boldsymbol{x}_F + \boldsymbol{L}_1^{\mathrm{up}}(C,F)\boldsymbol{L}_1^{\mathrm{up}}(F,F)^{\dagger}\boldsymbol{b}_F - \boldsymbol{L}_1^{\mathrm{up}}(C,F)\boldsymbol{x}_F + \mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_C\boldsymbol{x}_C\\
&= \boldsymbol{L}_1^{\mathrm{up}}(C,F)\boldsymbol{L}_1^{\mathrm{up}}(F,F)^{\dagger}\boldsymbol{b}_F + \mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_C\boldsymbol{x}_C.
\end{aligned}$$

That is, $\boldsymbol{h} \in \mathrm{Im}(\mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_C)$.

Next we look at $\boldsymbol{L}_1^{\mathrm{up}}\tilde{\boldsymbol{x}}$. Let $\boldsymbol{\delta} = \mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_C\tilde{\boldsymbol{x}}_C - \boldsymbol{h}$. We replace $\boldsymbol{x}_F$ and $\boldsymbol{x}_C$ in Equation

(C.1) with $\tilde{\boldsymbol{x}}_F$ and $\tilde{\boldsymbol{x}}_C$:

$$\begin{aligned}
\boldsymbol{L}_1^{\mathrm{up}}\tilde{\boldsymbol{x}} &= \begin{bmatrix} \boldsymbol{I} & \\ \boldsymbol{L}_1^{\mathrm{up}}(C,F)\boldsymbol{L}_1^{\mathrm{up}}(F,F)^{\dagger} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{L}_1^{\mathrm{up}}(F,F)\tilde{\boldsymbol{x}}_F + \boldsymbol{L}_1^{\mathrm{up}}(F,C)\tilde{\boldsymbol{x}}_C \\ \mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_C\tilde{\boldsymbol{x}}_C \end{bmatrix} \\
&= \begin{bmatrix} \boldsymbol{b}_F \\ \boldsymbol{L}_1^{\mathrm{up}}(C,F)\boldsymbol{L}_1^{\mathrm{up}}(F,F)^{\dagger}\boldsymbol{b}_F + \mathrm{Sc}(\boldsymbol{L}_1^{\mathrm{up}})_C\tilde{\boldsymbol{x}}_C \end{bmatrix} \\
&= \begin{bmatrix} \boldsymbol{b}_F \\ \boldsymbol{b}_C + \boldsymbol{\delta} \end{bmatrix}.
\end{aligned}$$

Then,

$$\begin{aligned}
\left\|\boldsymbol{L}_1^{\mathrm{up}}\tilde{\boldsymbol{x}} - \boldsymbol{b}\right\|_2 = \left\|\boldsymbol{\delta}\right\|_2 \le \delta\left\|\boldsymbol{h}\right\|_2 &\le \delta\left(\left\|\boldsymbol{b}_C\right\|_2 + \left\|\boldsymbol{L}_1^{\mathrm{up}}(C,F)\boldsymbol{L}_1^{\mathrm{up}}(F,F)^{\dagger}\right\|_2\left\|\boldsymbol{b}_F\right\|_2\right) \\
&\le \delta\left(1 + \left\|\boldsymbol{L}_1^{\mathrm{up}}(C,F)\boldsymbol{L}_1^{\mathrm{up}}(F,F)^{\dagger}\right\|_2\right)\left\|\boldsymbol{b}\right\|_2 \le \epsilon\left\|\boldsymbol{b}\right\|_2,
\end{aligned}$$

where the last inequality is by our setting of δ.

We compute $\tilde{\boldsymbol{x}}$ by two calls of UpLapFSolver and one call of SchurSolver and $O(1)$ matrix-vector multiplications and vector-vector additions. Thus, the total runtime is $O(t_1(m_F) + t_2(m_C) +$ the number of non-zeros in $\boldsymbol{L}_1^{\mathrm{up}})$. □

Curriculum Vitae

Personal Data

Name:	Ming Ding
Date of Birth:	February 16, 1995
Place of Birth:	Anhui, China
Citizen of:	China

Education

2020 – 2025	ETH Zurich, Zurich, Switzerland Ph.D. in Theoretical Computer Science
2017 – 2020	Shanghai Jiao Tong University, Shanghai, China M.Sc. in Information and Communication Engineering
2015 – 2017	Ecole Centrale Paris, Paris, France Diplôme d'Ingénieur
2013 – 2017	Shanghai Jiao Tong University, Shanghai, China B.Sc. in Information Engineering