58th Annual IEEE Symposium on Foundations of Computer Science

Hardness Results for Structured Linear Systems

Rasmus Kyng
Department of Computer Science
Yale University
New Haven, CT, USA
Email: rasmus.kyng@yale.edu

Peng Zhang
School of Computer Science
Georgia Institute of Technology
Atlanta, GA, USA
Email: pzhang60@gatech.edu

***Abstract*—We show that if the nearly-linear time solvers for Laplacian matrices and their generalizations can be extended to solve just slightly larger families of linear systems, then they can be used to quickly solve all systems of linear equations over the reals. This result can be viewed either positively or negatively: either we will develop nearly-linear time algorithms for solving all systems of linear equations over the reals, or progress on the families we can solve in nearly-linear time will soon halt.**

***Keywords*-Numerical Linear Algebra; Linear System Solvers; Laplacian Solvers; Multi-commodity Flow Problems; Truss Stiffness Matrices; Total Variation Matrices; Complexity Theory; Fine-grained Complexity;**

I. Introduction

We establish a dichotomy result for the families of linear equations that can be solved in nearly-linear time. If nearly-linear time solvers exist for a slight generalization of the families for which they are currently known, then nearly-linear time solvers exist for all linear systems over the reals[1].

This type of reduction is related to the successful research program of fine-grained complexity, such as the result [1] which showed that the existence of a "truly subcubic" time algorithm for All-Pairs Shortest Paths Problem is equivalent to the existence of "truly subcubic" time algorithm for a wide range of other problems. For any constant $a \geq 1$, our result establishes for 2-commodity matrices, and several other classes of graph structured linear systems, that we can solve a linear system in a matrix of this type with s nonzeros in time $\widetilde{O}(s^a)$ if and only if we can solve linear systems in all matrices with polynomially bounded integer entries in time $\widetilde{O}(s^a)$.

In the RealRAM model, given a matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ and a vector $\boldsymbol{c} \in \mathbb{R}^n$, we can solve the linear system $\boldsymbol{Ax} = \boldsymbol{c}$ in $O(n^\omega)$ time, where ω is the matrix multiplication constant, for which the best currently known bound is $\omega < 2.3727$ [2], [3]. Such a running time bound is cost prohibitive for the large sparse matrices often encountered in practice. Iterative methods [4], first order methods [5], and matrix sketches [6] can all be viewed as ways of obtaining significantly better performance in cases where the matrices have additional structure.

In contrast, when $\boldsymbol{A}$ is an $n \times n$ Laplacian matrix with m non-zeros, and polynomially bounded entries, the linear system $\boldsymbol{Ax} = \boldsymbol{c}$ can be solved approximately to ϵ-accuracy in $O((m+n)\log^{1/2+o(1)} n \log(1/\epsilon))$ time [7], [8]. This result spurred a series of major developments in fast graph algorithms, sometimes referred to as "the Laplacian Paradigm" of designing graph algorithms [9]. The asymptotically fastest known algorithms for Maximum Flow in directed unweighted graphs [10], [11], Negative Weight Shortest Paths and Maximum Weight Matchings [12], Minimum Cost Flows and Lossy Generalized Flows [13], [14] all rely on fast Laplacian linear system solvers.

The core idea of the Laplacian paradigm can be viewed as showing that the linear systems that arise from interior point algorithms, or second-order optimization methods, have graph structure, and can be preconditioned and solved using graph theoretic techniques. These techniques could potentially be extended to a range of other problems, provided fast solvers can be found for the corresponding linear systems. Here a natural generalization is in terms of the number of labels per vertex: graph Laplacians correspond to graph labeling problems where each vertex has one label, and these labels interact pairwise via edges. Multi-label variants of these exist in Markov random fields [15], image processing [16], Euclidean embedding of graphs [17], data processing for cryo-electron microscopy [18], [19], [20], phase retrieval [21], [22], and many image processing problems (e.g. [23], [24]). Furthermore, linear systems with multiple labels per vertex arise when solving multi-commodity flow problems using primal-dual methods. Linear systems related to multi-variate labelings of graphs have been formulated as the quadratically-coupled flow problem [25] and Graph-Structured Block Matrices [26]. They also occur naturally in linear elasticity problems for simulating the effect of forces on truss systems [27].

Due to these connections, a central question in the Laplacian paradigm of designing graph algorithms is whether all Graph-Structured Block Matrices can be solved in (approximately) nearly-linear time. Even obtaining subquadratic

Rasmus Kyng was supported by ONR Award N00014-16-1-2374.
Peng Zhang was partially supported by the NSF under Grant No. 1637566.

[1]A full version of this paper is available at https://arxiv.org/abs/1705.02944.

running time would be constitute significant progress. There has been some optimism in this direction due to the existence of faster solvers for special cases: nearly-linear time solvers for Connection Laplacians [28], 1-Laplacians of collapsible 3-D simplicial complexes [29], and algorithms with runtime about $n^{5/4}$ for 2D planar truss stiffness matrices [27]. Furthermore, there exists a variety of faster algorithms for approximating multi-commodity flows to $(1+\epsilon)$ accuracy in time that scales as $\mathrm{poly}(\epsilon^{-1})$ [30], [31], [32], [33], even obtaining nearly-linear running times when the graph is undirected [34], [35], [36].

The subquadratic variants of these routines also interact naturally with tools that in turn utilize Laplacian solvers [25], [33]. These existing tight algorithmic connections, as well as the solver for Connection Laplacians, and the fact that combinatorial preconditioners partly originated from speeding up interior point methods through preconditiong Hessians [37], together provide ample reason to hope that one could develop nearly-linear time solvers for linear systems related to multicommodity flows. Any algorithm that solves such systems to high accuracy in $m^{1+\alpha}$ time would in turn imply multicommodity flow algorithms that run in about $n^{1/2}m^{1+\alpha}$ time [13], while the current best running times are about $n^{2.5}$ [38].

Unfortunately, we show that if linear systems in general 2D truss stiffness matrices or 2-commodity Laplacians can be solved approximately in nearly-linear time, then all linear systems in matrices with polynomially bounded integer entries can be solved in nearly-linear time. In fact, we show in a strong sense that any progress made in developing solvers for these classes of matrices will translate directly into similarly fast solvers for all matrices with polynomially bounded integer entries. Thus developing faster algorithms for these systems will be as difficult as solving *all* linear systems faster.

Since linear system solvers used inside Interior Point Methods play a central role in the Laplacian paradigm for designing high-accuracy algorithms, this may suggest that in the high-accuracy regime the paradigm will not extend to most problems that require multiple labels/variables per edge or vertex. Alternatively, an algorithmic optimist might view our result as a road-map for solving all linear systems via reductions to fast linear system solvers for Graph-Structured Block Matrices.

A. Our Results

Fast linear system solvers for Laplacians, Connection Laplacians, Directed Laplacians, and 2D Planar Truss Stiffness matrices are all based on iterative methods and only produce approximate solutions. The running time for these solvers scales logarithmically with the error parameter ϵ, i.e. as $\log(1/\epsilon)$. Similarly, the running time for iterative methods usually depends on the condition number of the matrix, but for state-of-the-art solvers for Laplacians, Connection Laplacians, and Directed Laplacians, the dependence is logarithmic. Consequently, a central open question is whether fast approximate solvers exist for other structured linear systems, with running times that depend logarithmically on the condition number and the accuracy parameter.

Integral linear systems are reducible to Graph-Structured Block Matrices. Our reductions show that if fast approximate linear system solvers exist for multi-commodity Laplacians, 2D Truss Stiffness, or Total Variation (TV) Matrices, then fast approximate linear system solvers exist for any matrix, in the very general sense of minimizing $\min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{c}\|_2^2$. Thus our result also applies to singular matrices, where we solve the pseudo-inverse problem to high accuracy. Theorem I.1 gives an informal statement of our main result. The result is stated formally in Section III as Theorem III.2.

Theorem I.1 (Hardness for Graph-Structured Linear Systems (Informal))**.** *We consider three types of Graph-Structured Block Matrices: Multi-commodity Laplacians, Truss Stiffness Matrices, and Total Variation Matrices. Suppose that for one or more of these classes, the linear system $\boldsymbol{A}\boldsymbol{x}=\boldsymbol{c}$ in a matrix $\boldsymbol{A}$ with s non-zeros can be solved in time $\widetilde{O}(s^a)$, for some constant $a \geq 1$, with the running time having logarithmic dependence on condition number and accuracy*[2]*. Then linear systems in* all matrices *with polynomially bounded integer entries and condition number can be solved to high accuracy in time $\widetilde{O}(s^a)$, where again s is the number of non-zero entries of the matrix.*

Our results can easily be adapted to show that if fast exact linear system solvers exist for multi-commodity Laplacians, then exact solvers exist for all non-singular integer matrices. However, this is of less interest since there is less evidence that would suggest we should expect fast exact solvers to exist.

The notion of approximation used throughout this paper is the same as that used in the Laplacian solver literature (see Section II-A). To further justify the notion of approximate solutions to linear systems, we show that it lets us solve a natural decision problem for linear systems:

We show that deciding if a vector is approximately in the image of a matrix can be reduced to approximately solving linear systems. We show this in Section 10 of the full version of this paper. We also show that the exact image decision problem requires working with exponentially small numbers, even when the input has polynomially bounded integral entries and condition number. This means that in fixed-point arithmetic, we can only hope to solve an approximate version of the problem. The problem of approximately solving general linear systems can be reduced the problem of approximately solving Graph-Structured Block Matrix

[2]This is the kind of running time guarantee established for Laplacians, Directed Laplacians, Connection Laplacians, and bounded-weight planar 2D Truss Stiffness matrices.

linear systems. Together, these statements imply that we can also reduce the problem of deciding whether a vector is approximately in the image of a general matrix to the problem of approximately solving Graph-Structured Block Matrix linear systems.

We establish surprising separations between many problems known to have fast solutions and problems that are as hard solving general linear systems. Our results trace out several interesting dichotomies: restricted cases of 2D truss stiffness matrices have fast solvers, but fast solvers for all 2D truss stiffness matrices would imply equally fast solvers for all linear systems. TV matrices can be solved quickly in the anisotropic case, but in the isotropic case imply solvers for all linear systems. Fast algorithms exist for multi-commodity problems in the low accuracy regime, but existing approaches for the high accuracy regime seem to require fast solvers for multi-commodity linear systems, which again would imply fast solvers for all linear systems.

Our reductions only require the simplest cases of the classes we consider: 2-Commodity Laplacians are sufficient, as are (non-planar) 2D Truss Stiffness matrices, and Total Variation Matrices with 2-by-2 interactions. Linear systems of these three classes have many applications, and faster solvers for these would be useful in all applications. Trusses have been studied as the canonical multi-variate problem, involving definitions such as Fretsaw extensions [39] and factor widths [40], and fast linear system solvers exist for the planar 2D case with bounded weights [27]. Total Variation Matrices are widely used in image denoising [41]. The anisotropic version can be solved using nearly-linear time linear system solvers [42], while the isotropic version has often been studied using linear systems for which fast solvers are not known [43], [44], [45]. Multi-commodity flow problems have been the subject of extensive study, with significant progress on algorithms with low accuracy [30], [31], [32], [33], [34], [35], [36], while high accuracy approaches use slow general linear system solvers.

B. Approximately Solving Linear Systems and Normal Equations

The simplest notion of solving a linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{c}$, is to seek an $\boldsymbol{x}$ s.t. the equations are exactly satisfied. More generally, if the system is not guaranteed to have a solution, we can ask for an $\boldsymbol{x}$ which minimizes $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{c}\|_2^2$. An $\boldsymbol{x}$ which minimizes this always exists. In general, it may not be unique. Finding an $\boldsymbol{x}$ which minimizes $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{c}\|_2^2$ is equivalent to solving the linear system $\boldsymbol{A}^\top\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^\top\boldsymbol{c}$, which is referred to as the normal equation for the linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{c}$ (see [46]). The problem of solving the normal equations (or equivalently, minimizing $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{c}\|_2^2$), is a generalization of the problem of linear system solving, since the approach works when $\boldsymbol{A}$ is non-singular, while also giving meaningful results when $\boldsymbol{A}$ is singular. The normal equation problem can also be understood in terms of the Moore-Penrose pseudo-inverse of a matrix $\boldsymbol{M}$, which is denoted $\boldsymbol{M}^\dagger$ as $\boldsymbol{x} = (\boldsymbol{A}^\top\boldsymbol{A})^\dagger\boldsymbol{A}^\top\boldsymbol{c}$ is a solution to the normal equations. Taking the view of linear system solving as minimizing $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{c}\|_2^2$ also gives sensible ways to define an approximate solution to a linear system: It is an $\boldsymbol{x}$ that ensures $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{c}\|_2^2$ is close to $\min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{c}\|_2^2$. In Section II, we formally define several notions of approximate solutions to linear systems that we will use throughout the paper.

An important special case of linear systems is when the matrix of coefficients of the system is positive semi-definite. Since $\boldsymbol{A}^\top\boldsymbol{A}$ is always positive semi-definite, solving the normal equations for a linear system falls into this case. Linear systems over positive semi-definite matrices can be solved (approximately) by approaches known as iterative methods, which often lead to much faster algoritms than the approaches used for general linear systems. Iterative methods inherently produce approximate solutions[3].

C. Graph-Structured Block Matrices

Graph-Structured Block Matrices are a type of linear system that arise in many applications. Laplacian matrices and Connection Laplacians both fall in this category.

Suppose we have a collection of n disjoint sets of variables $X_1, \ldots, X_n$, with each set having the same size, $|X_i| = d$. Let $\boldsymbol{x}^i$ denote the vector[4] of variables in X_i, and consider an equation of the form $\boldsymbol{S}\boldsymbol{x}^i - \boldsymbol{T}\boldsymbol{x}^j = \boldsymbol{0}$, where $\boldsymbol{S}$ and $\boldsymbol{T}$ are both $r \times d$ matrices. Now we form a linear system $\boldsymbol{B}\boldsymbol{x} = \boldsymbol{0}$ by stacking m equations of the form given above as the rows of the system. Note that, very importantly, we allow a different choice of $\boldsymbol{S}$ and $\boldsymbol{T}$ for every pair of i and j. This matrix $\boldsymbol{B} \in \mathbb{R}^{mr \times nd}$ we refer to as a Incidence-Structured Block Matrix (ISBM), while we refer to $\boldsymbol{B}^\top\boldsymbol{B}$ as a Graph-Structured Block Matrix (GSBM). Note that $\boldsymbol{B}$ is not usually PSD, but $\boldsymbol{B}^\top\boldsymbol{B}$ is. The number of non-zeros in $\boldsymbol{B}^\top\boldsymbol{B}$ is $O(md^2)$. GSBMs come up in many applications, where we typically want to solve a linear system in the normal equations of $\boldsymbol{B}$.

Laplacian matrices are GSBMs where $d = 1$ and $\boldsymbol{S} = \boldsymbol{T} = w$, where w is a real number, and we allow different w for each pair of i and j. The corresponding ISBM for Laplacians is called an edge-vertex incidence matrix. Connection Laplacians are GSBMs where $d = O(1)$ and $\boldsymbol{S} = \boldsymbol{T}^\top = w\boldsymbol{Q}$, for some rotation matrix $\boldsymbol{Q}$ and a real number w. Again, we allow a different rotation matrix and scaling for every edge. For both Laplacians and Connection Laplacians, there exist linear system solvers that run in time

[3]A seeming counterexample to this is Conjugate Gradient which is an iterative method that produces exact solutions in the RealRAM model. But it requires extremely high precision calculations to exhibit this behaviour in finite precision arithmetic, and so Conjugate Gradient is also best understood as an approximate method.

[4]We use superscripts to index a sequence of vectors or matrices, and we use subscripts to denote entries of a vector or matrix, see Section II.

$O(m \operatorname{polylog}(n, \epsilon^{-1}))$ and produce ϵ approximate solutions to the corresponding normal equations.

We now introduce several classes of ISBMs and their associated GSBMs. Our Theorem III.2 shows that fast linear system solvers for any of these classes would imply fast linear system solvers for all matrices with polynomially bounded entries and condition number.

Definition I.2 (2-commodity incidence matrix)**.** A *2-commodity incidence matrix* is an ISBM where $d = 2$ and $r = 1$, and $\boldsymbol{S} = \boldsymbol{T}$, and we allow three types of $\boldsymbol{S}$: $\boldsymbol{S} = w\begin{pmatrix}1 & 0\end{pmatrix}$, $\boldsymbol{S} = w\begin{pmatrix}0 & 1\end{pmatrix}$ and $\boldsymbol{S} = w\begin{pmatrix}1 & -1\end{pmatrix}$, where in each case w is a real number which may depend on the pair i and j. We denote the set of all 2-commodity incidence matrices by $\mathcal{MC}_2$. The corresponding GSBM is called a 2-commodity Laplacian. The ISBM definition is equivalent to requiring the GSBM to have the form

$$\boldsymbol{L}^1 \otimes \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + \boldsymbol{L}^2 \otimes \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} + \boldsymbol{L}^{1+2} \otimes \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

where $\otimes$ is the tensor product and $\boldsymbol{L}^1$, $\boldsymbol{L}^2$, and $\boldsymbol{L}^{1+2}$ are all Laplacian matrices.

We adopt a convention that the first variable in a set X_i is labelled $\boldsymbol{u}_i$ and the second is labelled $\boldsymbol{v}_i$. Using this convention, given a 2-commodity incidence matrix $\boldsymbol{B}$, the equation $\boldsymbol{B}\boldsymbol{x} = \boldsymbol{0}$ must consist of scalings of the following three types of equations: $\boldsymbol{u}_i - \boldsymbol{u}_j = 0$, $\boldsymbol{v}_i - \boldsymbol{v}_j = 0$, and $\boldsymbol{u}_i - \boldsymbol{v}_i - (\boldsymbol{u}_j - \boldsymbol{v}_j) = 0$.

Definition I.3 (Strict 2-commodity incidence matrix)**.** A *strict 2-commodity incidence matrix* is a 2-commodity incidence matrix where the corresponding 2-commodity Laplacian has the property that $\boldsymbol{L}^1$, $\boldsymbol{L}^2$, and $\boldsymbol{L}^{1+2}$ all have the same non-zero pattern. We denote the set of all strict 2-commodity incidence matrices by $\mathcal{MC}_2^{>0}$. We denote the set of all strict 2-commodity incidence matrices with *integer entries* by $\mathcal{MC}_{2,\mathbb{Z}}^{>0}$.

Linear systems in $\mathcal{MC}_2^{>0}$ are exactly the systems that one has to solve to when solving 2-commodity problems using Interior Point Methods (IPMs). For readers unfamiliar with 2-commodity problems or IPMs, we provide a brief explanation of why this is the case in Section 9 of the full version of this paper. The $\mathcal{MC}_2^{>0}$ is more restrictive than $\mathcal{MC}_2$, and $\mathcal{MC}_{2,\mathbb{Z}}^{>0}$ in turn is even more restrictive. One could hope that fast linear system solvers exist for $\mathcal{MC}_2^{>0}$ or $\mathcal{MC}_{2,\mathbb{Z}}^{>0}$, even if they do not exist for $\mathcal{MC}_2$. However, our reductions show that even getting a fast approximate solver for $\mathcal{MC}_{2,\mathbb{Z}}^{>0}$ with polynomially bounded entries and condition number will lead to a fast solver for all matrices with polynomially bounded entries and condition number.

The next class we consider is 2D Truss Stiffness Matrices. They have been studied extensively in the numerical linear algebra community [39], [40]. For *Planar* 2D Trusses with some bounds on ranges of edges, Daitch and Spielman obtained linear system solvers that run in time $\widetilde{O}(n^{5/4}\log(1/\epsilon))$.

Definition I.4 (2D Truss Incidence Matrices)**.** Let $G = (V, E)$ be a graph whose vertices are n points in 2-dimension: $\boldsymbol{s}^1, \ldots, \boldsymbol{s}^n \in \mathbb{R}^2$. Consider $X_1, \ldots, X_n$ where $d = 2$. A *2D Truss Incidence Matrix* is an ISBM where $d = 2$ and $r = 1$, and for each i and j, we have $\boldsymbol{S} = \boldsymbol{T}$ and $\boldsymbol{S} = w(\boldsymbol{s}^i - \boldsymbol{s}^j)^\top$, and w is a real number that may depend on the pair i and j, but $\boldsymbol{s}^i$ depends only on i and vice versa for $\boldsymbol{s}^j$. We denote the class of all 2D Truss Incidence Matrices by $\mathcal{T}_2$.

Another important class of matrices is Total Variation Matrices (TV matrices). TV matrices come from Interior Point Methods for solving total variation minimization problem in image, see for example [47] and [48]. Not all TV matrices are GSBMs, but many GSBMs can be expressed as TV matrices.

Definition I.5 (TV matrix and 2-TV Incidence Matrices)**.** Let $E_1 \cup \ldots \cup E_s$ be a partition of the edge set of a graph. For each $1 \leq i \leq s$, let $\boldsymbol{B}^i$ be the edge-vertex incidence matrix of E_i, $\boldsymbol{W}^i$ be a diagonal matrix of edge weights, and $\boldsymbol{r}^i$ be a vector satisfying $\boldsymbol{W}^i \succcurlyeq \boldsymbol{r}^i(\boldsymbol{r}^i)^\top$. Given these objects, the associated *total variation matrix* (TV matrix) is a matrix $\boldsymbol{M}$ defined as

$$\boldsymbol{M} = \sum_{1 \leq i \leq s} (\boldsymbol{B}^i)^\top \left(\boldsymbol{W}^i - \boldsymbol{r}^i(\boldsymbol{r}^i)^\top\right) \boldsymbol{B}^i.$$

A *2-TV Incidence Matrix* is defined as any ISBM whose corresponding GSBM is a TV matrix with $\boldsymbol{W}^i \in \mathbb{R}^{2\times 2}$ and $\boldsymbol{r}^i \in \mathbb{R}^2$. We denote the class of all 2-TV incidence matrices by $\mathcal{V}_2$.

D. *Our Reduction: Discussion and an Example*

In this section we give a brief sketch of the ideas behind our reduction from general linear systems, over matrices in $\mathcal{G}$, to multi-commodity linear systems, over matrices in $\mathcal{MC}_2$, and we demonstrate the most important transformation through an example.

The starting point for our approach is the folklore idea that any linear system can be written as a factor-width 3 system by introducing a small number of extra variables. Using a set of multi-commodity constraints, we are able to express one particular factor-width 3 equation, namely $2x'' = x + x'$. After a sequence of preprocessing steps, we are then able to efficiently express arbitrary linear systems over integer matrices using constraints of this form. A number of further issues arise when the initial matrix does not have full column rank, requiring careful weighting of the constraints we introduce.

Given a matrix $\boldsymbol{A}$ with polynomially bounded integer entries and condition number, we reduce the linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{c}$ to a linear system $\boldsymbol{B}\boldsymbol{y} = \boldsymbol{d}$, where $\boldsymbol{B}$ is a strict multi-commodity edge-vertex incidence matrix with integer entries (i.e. in $\mathcal{MC}_{2,\mathbb{Z}}^{>0}$), with polynomially bounded entries and condition number. More precisely, we reduce $\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^\top \boldsymbol{c}$ to $\boldsymbol{B}^\top \boldsymbol{B}\boldsymbol{y} = \boldsymbol{B}^\top \boldsymbol{d}$. These systems always have a solution. We show that we can find an ϵ-approximate solution to the linear system $\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^\top \boldsymbol{c}$ by a simple mapping on any $\boldsymbol{y}$ that ϵ'-approximately solves the linear system $\boldsymbol{B}^\top \boldsymbol{B}\boldsymbol{y} = \boldsymbol{B}^\top \boldsymbol{d}$, where ϵ' is only polynomially smaller than ϵ. If $\boldsymbol{A}$ has s non-zero entries and the maximum absolute value of an entry in $\boldsymbol{A}$ is U, then $\boldsymbol{B}$ will have $O(s\log(sU))$ non-zero entries and our algorithm computes the reduction in time $O(s\log(sU))$. Note that $\boldsymbol{B}^\top \boldsymbol{B}$ has $r = O(s\log(sU))$ non-zeros, because every row of $\boldsymbol{B}$ has $O(1)$ entries. All together, this means that getting a solver for $\boldsymbol{B}^\top \boldsymbol{B}\boldsymbol{x} = \boldsymbol{B}^\top \boldsymbol{d}$ with running time $\widetilde{O}(r^a \log(1/\epsilon))$ will give a solver for $\boldsymbol{A}$ with $\widetilde{O}(s^a \log(1/\epsilon))$ running time.

We achieve this through a chain of reductions. Each reduction produces a new matrix and vector, as well as a new error parameter giving the accuracy required in the new system to achieve the accuracy desired in the original system.

1) We get a new linear system $\boldsymbol{A}^{\mathrm{Z},2}\boldsymbol{x}^{\mathrm{Z},2} = \boldsymbol{c}^{\mathrm{Z},2}$ where $\boldsymbol{A}^{\mathrm{Z},2}$ has integer entries, and the entries of each row of $\boldsymbol{A}^{\mathrm{Z},2}$ sum to zero, i.e. $\boldsymbol{A}^{\mathrm{Z},2}\boldsymbol{1} = \boldsymbol{0}$, and finally in every row the sum of the positive coefficients is a power of two.
2) $\boldsymbol{A}^{\mathrm{Z},2}\boldsymbol{x}^{\mathrm{Z},2} = \boldsymbol{c}^{\mathrm{Z},2}$ is then transformed to $\boldsymbol{B}\boldsymbol{y} = \boldsymbol{d}$, where $\boldsymbol{B}$ is a 2-commodity edge-vertex incidence matrix.
3) $\boldsymbol{B}\boldsymbol{y} = \boldsymbol{d}$ is then transformed to $\boldsymbol{B}^{>0}\boldsymbol{y} = \boldsymbol{d}^{>0}$, where $\boldsymbol{B}^{>0}$ is a strict 2-commodity edge-vertex incidence matrix.
4) $\boldsymbol{B}^{>0}\boldsymbol{y} = \boldsymbol{d}^{>0}$ is then transformed to $\boldsymbol{B}^{>0,\mathbb{Z}}\boldsymbol{y} = \boldsymbol{d}^{>0,\mathbb{Z}}$, where $\boldsymbol{B}^{>0,\mathbb{Z}}$ is a 2-commodity edge-vertex incidence matrix with integer entries.

We will demonstrate step 2, the main transformation, by example. When the context is clear, we drop the superscripts of matrices for simplicity. The reduction handles each row (i.e. equation) of the linear system independently, so we focus on the reduction for a single row.

Consider a linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{c}$, and let us pick a single row (i.e. equation) $\boldsymbol{A}_i\boldsymbol{x} = \boldsymbol{c}_i$ [5]. We will repeatedly pick pairs of existing variables of $\boldsymbol{x}$, say x and x', based on their current coefficients in $\boldsymbol{A}_i\boldsymbol{x} = \boldsymbol{c}_i$, and modify the row by adding $C(2x'' - (x+x'))$ to the left hand side where x'' is a new variable and C is a real number we pick. As we will see in a moment, we can use this pair-and-replace operation to simplify the row until it eventually becomes a 2-commodity equation. At the same time as we modify $\boldsymbol{A}_i$, we also store an auxiliary equation $C(x+x'-2x'') = 0$. Suppose initially that $\boldsymbol{A}_i\boldsymbol{x} = \boldsymbol{c}_i$ is satisfied. After this modification of $\boldsymbol{A}_i\boldsymbol{x} = \boldsymbol{c}_i$, if the auxiliary equation is satisfied, $\boldsymbol{A}_i\boldsymbol{x} = \boldsymbol{c}_i$ is still satisfied by the same values of x and x'. Crucially, we can express the auxiliary equation $C(x + x' - 2x'') = 0$ by a set of ten 2-commodity equations, i.e. a "2-commodity gadget" for this equation. Our final output matrix *will not* contain the equation $C(x+x'-2x'') = 0$ as a row, but will instead contain 10 rows of 2-commodity equations from our gadget construction. Eventually, our pair-and-replace scheme will also transform the row $\boldsymbol{A}_i\boldsymbol{x} = \boldsymbol{c}_i$ into a 2-commodity equation on just two variables.

[5]We use $\boldsymbol{A}_i$ to denote the ith row of $\boldsymbol{A}$, and $\boldsymbol{c}_i$ to denote the ith entry of $\boldsymbol{c}$, see Section II.

Next, we need to understand how the pair-and-replace scheme makes progress. The pairing handles the positive and the negative coefficients of $\boldsymbol{A}_i$ separately, and eventually ensures that $\boldsymbol{A}_i\boldsymbol{x} = \boldsymbol{c}_i$ has only a single positive and a single negative coefficient in the modified row $\boldsymbol{A}_i\boldsymbol{x} = \boldsymbol{c}_i$, in particular it is of the form $ax - ax' = \boldsymbol{c}_i$ for two variables x and x' that appear in the modified vector of variables $\boldsymbol{x}$, i.e. it is a 2-commodity equation.

To understand the pairing scheme, it is helpful to think about the entries of $\boldsymbol{A}$ written using binary (ignoring the sign of the entry). The pairing scheme proceeds in a sequence of rounds: In the first round we pair variables whose 1st (smallest) bit is 1. There must be an even number of variables with smallest bit 1, as the sum of the positive (and respectively negative) coefficients is a power of 2. We then replace the terms corresponding to the 1st bit of the pair with a new single variable with a coefficient of 2. After the first round, every coefficient has zero in the 1st bit. In the next round, we pair variables whose 2nd bit is 1, and replace the terms corresponding to the the 2nd bit of the pair with a new single variable with a coefficient of 4, and so on. Because the positive coefficients sum to a power of two, we are able to guarantee that pairing is always possible. It is not too hard to show that we do not create a large number of new variables or equations using this scheme.

For example, let us consider an equation

$$3\boldsymbol{x}_1 + 5\boldsymbol{x}_2 + 4\boldsymbol{x}_3 + 4\boldsymbol{x}_4 - 16\boldsymbol{x}_5 = 1.$$

Replace $\boldsymbol{x}_1 + \boldsymbol{x}_2$ by $2\boldsymbol{x}_6$. Add auxiliary equation $\boldsymbol{x}_1 + \boldsymbol{x}_2 - 2\boldsymbol{x}_6 = 0$. The equation above becomes

$$2(\boldsymbol{x}_1 + 2\boldsymbol{x}_2 + \boldsymbol{x}_6 + 2\boldsymbol{x}_3 + 2\boldsymbol{x}_4) - 16\boldsymbol{x}_5 = 1.$$

Replace $2(\boldsymbol{x}_1 + \boldsymbol{x}_6)$ by $4\boldsymbol{x}_7$. Add auxiliary equation $2(\boldsymbol{x}_1 + \boldsymbol{x}_6 - 2\boldsymbol{x}_7) = 0$. We now have

$$4(\boldsymbol{x}_2 + \boldsymbol{x}_7 + \boldsymbol{x}_3 + \boldsymbol{x}_4) - 16\boldsymbol{x}_5 = 1.$$

Replace $4(\boldsymbol{x}_2 + \boldsymbol{x}_7)$ by $8\boldsymbol{x}_8$, and $4(\boldsymbol{x}_3 + \boldsymbol{x}_4)$ by $8\boldsymbol{x}_9$. Add auxiliary equations $4(\boldsymbol{x}_2 + \boldsymbol{x}_7 - 2\boldsymbol{x}_8) = 0$, and $4(\boldsymbol{x}_3 + \boldsymbol{x}_4 - 2\boldsymbol{x}_9) = 0$. Our equation above becomes

$$8(\boldsymbol{x}_8 + \boldsymbol{x}_9) - 16\boldsymbol{x}_5 = 1.$$

Replace $8(\boldsymbol{x}_8 + \boldsymbol{x}_9)$ by $16\boldsymbol{x}_{10}$. Add auxiliary equation $8(\boldsymbol{x}_8 + \boldsymbol{x}_9 - 2\boldsymbol{x}_{10}) = 0$. Finally, the equation above has become a 2-commodity equation:

$$16\boldsymbol{x}_{10} - 16\boldsymbol{x}_5 = 1.$$

Now, let us build some intuition for how to replace the equation $C(x + x' - 2x'') = 0$ by ten 2-commodity equations, i.e. a 2-commodity gadget. Recall that each index i corresponds to a $\boldsymbol{u}$-variable $\boldsymbol{u}_i$ and a $\boldsymbol{v}$-variable $\boldsymbol{v}_i$. We think of x, x', x'' all as $\boldsymbol{u}$-variables. Roughly speaking, the 2-commodity equations of the form $\boldsymbol{u}_i - \boldsymbol{u}_j = 0$ and $\boldsymbol{v}_i - \boldsymbol{v}_j = 0$ allow us to set two variables equal, although the effect is more complicated when considering an overconstrained system. The 2-commodity equations of the form $\boldsymbol{u}_i - \boldsymbol{v}_i - (\boldsymbol{u}_j - \boldsymbol{v}_j) = 0$ are even more important to us: The constraint $x + x' - 2x'' = 0$ can be obtained by adding two equations: $x - \boldsymbol{v}_i - (x'' - \boldsymbol{v}_j) = 0$ and $x' - \boldsymbol{v}_j - (x'' - \boldsymbol{v}_i) = 0$. The appearance of the $\boldsymbol{v}_j - \boldsymbol{v}_i$ term in both equations, though with opposite sign, gives a degree of freedom that ensures that we do not impose additional constraints on x, x', and x''. We get the desired constraints listed above by starting with two 2-commodity equations using fresh variables with new indices a, b, c, d: $\boldsymbol{u}_a - \boldsymbol{v}_a - (\boldsymbol{u}_b - \boldsymbol{v}_b) = 0$ and $\boldsymbol{u}_c - \boldsymbol{v}_c - (\boldsymbol{u}_d - \boldsymbol{v}_d) = 0$. We then link together these variables using constraints on pairs of variables of the same type (i.e. $\boldsymbol{u}$ or $\boldsymbol{v}$). First, we link the $\boldsymbol{u}$ and x variables: $\boldsymbol{u}_a - x = 0, \boldsymbol{u}_c - x' = 0$, $\boldsymbol{u}_b - x'' = 0$, $\boldsymbol{u}_d - x'' = 0$. Secondly, we constrain the $\boldsymbol{v}$ variables to give only the single degree of freedom we need: For technical reasons, we introduce two more new indices f and g and set $\boldsymbol{v}_a - \boldsymbol{v}_f = 0$, $\boldsymbol{v}_f - \boldsymbol{v}_d = 0$, and $\boldsymbol{v}_b - \boldsymbol{v}_g = 0$, $\boldsymbol{v}_g - \boldsymbol{v}_c = 0$. Now we are left with a system essentially equivalent to the two equations $x - \boldsymbol{v}_a - (x'' - \boldsymbol{v}_b) = 0$ and $x' - \boldsymbol{v}_b - (x'' - \boldsymbol{v}_a) = 0$. These equations impose exactly the constaint $x + x' - 2x'' = 0$ that we want.

In this way, we process $\boldsymbol{Ax} = \boldsymbol{c}$ to produce a new set of equations $\boldsymbol{By} = \boldsymbol{d}$ where $\boldsymbol{B}$ is a 2-commodity matrix. If $\boldsymbol{Ax} = \boldsymbol{c}$ has an exact solution, this solution can be obtained directly from an exact solution to $\boldsymbol{By} = \boldsymbol{d}$. We also show that an approximate solution to $\boldsymbol{By} = \boldsymbol{d}$ leads to an approximate solution for $\boldsymbol{Ax} = \boldsymbol{c}$, and we show that $\boldsymbol{B}$ does not have much larger entries or condition number than $\boldsymbol{A}$.

The situation is more difficult when $\boldsymbol{Ax} = \boldsymbol{c}$ does not have a solution and we want to obtain an approximate minimizer $\arg\min_{\boldsymbol{x}\in\mathbb{R}^n} \|\boldsymbol{Ax} - \boldsymbol{c}\|_2^2$ from an approximate solution to $\arg\min_{\boldsymbol{y}\in\mathbb{R}^{n'}} \|\boldsymbol{By} - \boldsymbol{d}\|_2^2$. This corresponds to approximately applying the Moore-Penrose pseudo-inverse of $\boldsymbol{A}$ to $\boldsymbol{c}$. We deal with the issues that arise here using a carefully chosen scaling of each auxiliary constraint to ensure a strong relationship between different solutions.

In order to switch from a linear system in a general 2-commodity matrix to a linear system in a *strict* 2-commodity matrix, we need to reason very carefully about the changes to the null space that this transformation inherently produces. By choosing sufficiently small weights, we are nonetheless able to establish a strong relationship between the normal equation solutions despite the change to the null space.

II. Preliminaries

We use subscripts to denote entries of a matrix or a vector: let $\boldsymbol{A}_i$ denote the ith row of 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 vector $\boldsymbol{x}$ and $\boldsymbol{x}_{i:j}$ $(i < j)$ denote the vector of entries $\boldsymbol{x}_i, \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_j$. We use superscripts to index a sequence of matrices or vectors, e.g., $\boldsymbol{A}^1, \boldsymbol{A}^2, \ldots,$ and $\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots$, except when some other meaning is clearly stated.

We use $\boldsymbol{A}^\dagger$ to denote the Moore-Penrose pseudo-inverse of a matrix $\boldsymbol{A}$. We use $\mathrm{im}(\boldsymbol{A})$ to denote the image of a matrix $\boldsymbol{A}$. We use $\|\cdot\|_2$ to denote the Euclidean norm on vectors and the spectral norm on matrices. When $\boldsymbol{M}$ is an $n \times n$ positive semidefinite matrix, we define a norm on vectors $\boldsymbol{x} \in \mathbb{R}^n$ by $\|\boldsymbol{x}\|_{\boldsymbol{M}} \overset{\text{def}}{=} \sqrt{\boldsymbol{x}^\top \boldsymbol{Mx}}$. We let $\mathrm{nnz}(\boldsymbol{A})$ denote the number of non-zero entries in a matrix $\boldsymbol{A}$. We define $\|\boldsymbol{A}\|_{\max} = \max_{i,j} |\boldsymbol{A}_{ij}|$, $\|\boldsymbol{A}\|_1 = \max_j \sum_i |\boldsymbol{A}_{ij}|$ and $\|\boldsymbol{A}\|_\infty = \max_i \sum_j |\boldsymbol{A}_{ij}|$. We let $\min_+(\boldsymbol{A}) = \min_{i,j \text{ s.t. } \boldsymbol{A}_{ij} \neq 0} |\boldsymbol{A}_{ij}|$. Given a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and a vector $\boldsymbol{c} \in \mathbb{R}^m$ for some m, n, we call the tuple $(\boldsymbol{A}, \boldsymbol{c})$ a linear system. Given matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, let $\boldsymbol{\Pi}_{\boldsymbol{A}} \overset{\text{def}}{=} \boldsymbol{A}(\boldsymbol{AA}^\top)^\dagger \boldsymbol{A}^\top$, i.e. the orthogonal projection onto $\mathrm{im}(\boldsymbol{A})$. Note that $\boldsymbol{\Pi}_{\boldsymbol{A}} = \boldsymbol{\Pi}_{\boldsymbol{A}}^\top$ and $\boldsymbol{\Pi}_{\boldsymbol{A}} = \boldsymbol{\Pi}_{\boldsymbol{A}}^2$.

A. Approximately Solving A Linear System

In this section we formally define the notions of approximate solutions to linear systems that we work with throughout this paper.

Definition II.1 (Linear System Approximation Problem, LSA)**.** Given linear system $(\boldsymbol{A}, \boldsymbol{c})$, where $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, and $\boldsymbol{c} \in \mathbb{R}^m$, and given a scalar $0 \le \epsilon \le 1$, we refer to the LSA problem for the triple $(\boldsymbol{A}, \boldsymbol{c}, \epsilon)$ as the problem of finding $\boldsymbol{x} \in \mathbb{R}^n$ s.t.

$$\|\boldsymbol{Ax} - \boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{c}\|_2 \le \epsilon \|\boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{c}\|_2 ,$$

and we say that such an $\boldsymbol{x}$ is a solution to the LSA instance $(\boldsymbol{A}, \boldsymbol{c}, \epsilon)$.

This definition of a LSA instance and solution has several advantages: when $\mathrm{im}(\boldsymbol{A}) = \mathbb{R}^m$, we get $\boldsymbol{\Pi}_{\boldsymbol{A}} = \boldsymbol{I}$, and it reduces to the natural condition $\|\boldsymbol{Ax} - \boldsymbol{c}\|_2 \le \epsilon \|\boldsymbol{c}\|_2$, which because $\mathrm{im}(\boldsymbol{A}) = \mathbb{R}^m$, can be satisfied for any ϵ, and for $\epsilon = 0$ tells us that $\boldsymbol{Ax} = \boldsymbol{c}$.

When $\mathrm{im}(\boldsymbol{A})$ does not include all of $\mathbb{R}^m$, the vector $\boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{c}$ is exactly the projection of $\boldsymbol{c}$ onto $\mathrm{im}(\boldsymbol{A})$, and so a solution can still be obtained for any ϵ. Further, as $(\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{A}})\boldsymbol{c}$ is orthogonal to $\boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{c}$ and $\boldsymbol{Ax}$, it follows that

$$\|\boldsymbol{Ax} - \boldsymbol{c}\|_2^2 = \|(\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{A}})\boldsymbol{c}\|_2^2 + \|\boldsymbol{Ax} - \boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{c}\|_2^2 .$$

Thus, when $\boldsymbol{x}$ is a solution to the LSA instance $(\boldsymbol{A}, \boldsymbol{c}, \epsilon)$, then $\boldsymbol{x}$ also gives an $\epsilon^2 \|\boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{c}\|_2^2$ additive approximation to

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \|\boldsymbol{Ax} - \boldsymbol{c}\|_2^2 = \|(\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{A}})\boldsymbol{c}\|_2^2 . \qquad (1)$$

Similarly, an $\boldsymbol{x}$ which gives an additive $\epsilon^2 \left\| \boldsymbol{\Pi_A c} \right\|_2^2$ approximation to Problem (1) is always a solution to the LSA instance $(\boldsymbol{A}, \boldsymbol{c}, \epsilon)$. These observations prove the following (well-known) fact:

Fact II.2. *Let $\boldsymbol{x}^* \in \arg\min_{\boldsymbol{x}\in\mathbb{R}^m} \left\| \boldsymbol{Ax} - \boldsymbol{c} \right\|_2^2$, then for every $\boldsymbol{x}$,*

$$\left\| \boldsymbol{Ax} - \boldsymbol{c} \right\|_2^2 \leq \left\| \boldsymbol{Ax}^* - \boldsymbol{c} \right\|_2^2 + \epsilon^2 \left\| \boldsymbol{\Pi_A c} \right\|_2^2$$

if and only if $\boldsymbol{x}$ is a solution to the LSA *instance $(\boldsymbol{A}, \boldsymbol{c}, \epsilon)$.*

When the linear system $\boldsymbol{Ax} = \boldsymbol{c}$ does not have a solution, a natural notion of solution is any minimizer of Problem (1). A simple calculation shows that this is equivalent to requiring that $\boldsymbol{x}$ is a solution to the linear system $\boldsymbol{A}^\top \boldsymbol{Ax} = \boldsymbol{A}^\top \boldsymbol{c}$, which always has a solution even when $\boldsymbol{Ax} = \boldsymbol{c}$ does not. The system $\boldsymbol{A}^\top \boldsymbol{Ax} = \boldsymbol{A}^\top \boldsymbol{c}$ is referred to as the *normal equation* associated with $\boldsymbol{Ax} = \boldsymbol{c}$ (see [46]).

Fact II.3. *$\boldsymbol{x}^* \in \arg\min_{\boldsymbol{x}\in\mathbb{R}^n} \left\| \boldsymbol{Ax} - \boldsymbol{c} \right\|_2^2$, if and only if $\boldsymbol{A}^\top \boldsymbol{Ax}^* = \boldsymbol{A}^\top \boldsymbol{c}$, and this linear system always has a solution.*

This leads to a natural question: Suppose we want to approximately solve the linear system $\boldsymbol{A}^\top \boldsymbol{Ax} = \boldsymbol{A}^\top \boldsymbol{c}$. Can we choose our notion of approximation to be equivalent to that of a solution to the LSA instance $(\boldsymbol{A}, \boldsymbol{c}, \epsilon)$?

A second natural question is whether we can choose a notion of distance between a proposed solution $\boldsymbol{x}$ and an optimal solution $\boldsymbol{x}^* \in \arg\min_{\boldsymbol{x}\in\mathbb{R}^n} \left\| \boldsymbol{Ax} - \boldsymbol{c} \right\|_2^2$ s.t. this distance being small is equivalent to $\boldsymbol{x}$ being a solution to the LSA instance $(\boldsymbol{A}, \boldsymbol{c}, \epsilon)$? The answer to both questions is yes, as demonstrated by the following facts:

Fact II.4. *Suppose $\boldsymbol{x}^* \in \arg\min_{\boldsymbol{x}\in\mathbb{R}^n} \left\| \boldsymbol{Ax} - \boldsymbol{c} \right\|_2^2$ then*

1) $\left\| \boldsymbol{A}^\top \boldsymbol{Ax} - \boldsymbol{A}^\top \boldsymbol{c} \right\|_{(\boldsymbol{A}^\top \boldsymbol{A})^\dagger} = \left\| \boldsymbol{Ax} - \boldsymbol{\Pi_A c} \right\|_2 = \left\| \boldsymbol{x} - \boldsymbol{x}^* \right\|_{\boldsymbol{A}^\top \boldsymbol{A}}$.
2) *The following statements are each equivalent to $\boldsymbol{x}$ being a solution to the* LSA *instance $(\boldsymbol{A}, \boldsymbol{c}, \epsilon)$:*
   a) $\left\| \boldsymbol{A}^\top \boldsymbol{Ax} - \boldsymbol{A}^\top \boldsymbol{c} \right\|_{(\boldsymbol{A}^\top \boldsymbol{A})^\dagger} \leq \epsilon \left\| \boldsymbol{A}^\top \boldsymbol{c} \right\|_{(\boldsymbol{A}^\top \boldsymbol{A})^\dagger}$ *if and only if $\boldsymbol{x}$ is a solution to the* LSA *instance $(\boldsymbol{A}, \boldsymbol{c}, \epsilon)$.*
   b) $\left\| \boldsymbol{x} - \boldsymbol{x}^* \right\|_{\boldsymbol{A}^\top \boldsymbol{A}} \leq \epsilon \left\| \boldsymbol{x}^* \right\|_{\boldsymbol{A}^\top \boldsymbol{A}}$ *if and only if $\boldsymbol{x}$ is a solution to the* LSA *instance $(\boldsymbol{A}, \boldsymbol{c}, \epsilon)$.*

For completeness, we prove Fact II.4 in Appendix A of the full version of this paper. Fact II.4 explains connection between our Definition II.1, and the usual convention for measuring error in the Laplacian solver literature [7]. In this setting, we consider a Laplacian matrix $\boldsymbol{L}$, which can be written as $\boldsymbol{L} = \boldsymbol{A}^\top \boldsymbol{A} \in \mathbb{R}^{n\times n}$, and a vector $\boldsymbol{b}$ s.t. $\boldsymbol{\Pi}_{\boldsymbol{A}^\top \boldsymbol{A}} \boldsymbol{b} = \boldsymbol{b}$. This condition on $\boldsymbol{b}$ is easy to verify in the case of Laplacians, since for the Laplacian of a connected graph, $\boldsymbol{\Pi}_{\boldsymbol{A}^\top \boldsymbol{A}} = \boldsymbol{I} - \frac{1}{n}\boldsymbol{1}\boldsymbol{1}^\top$. Additionally, it is also equivalent to the condition that there exists $\boldsymbol{c}$ s.t. $\boldsymbol{b} = \boldsymbol{A}^\top \boldsymbol{c}$. For Laplacians it is possible to compute both $\boldsymbol{A}$ and a vector $\boldsymbol{c}$ s.t. $\boldsymbol{b} = \boldsymbol{A}^\top \boldsymbol{c}$ in time linear in $\mathrm{nnz}(\boldsymbol{L})$. For Laplacian solvers, the approximation error of an approximate solution $\boldsymbol{x}$ is measured by the ϵ s.t. $\left\| \boldsymbol{A}^\top \boldsymbol{Ax} - \boldsymbol{b} \right\|_{(\boldsymbol{A}^\top \boldsymbol{A})^\dagger} \leq \epsilon \left\| \boldsymbol{b} \right\|_{(\boldsymbol{A}^\top \boldsymbol{A})^\dagger}$. By Fact II.4, we see that this is exactly equivalent to $\boldsymbol{x}$ being a solution to the LSA instance $(\boldsymbol{A}, \boldsymbol{c}, \epsilon)$.

B. Measuring the Difficulty of Solving a Linear System

Running times for iterative linear system solvers generally depend on the number of non-zeros in the input matrix, the condition number of the input matrix, the accuracy, and the bit complexity.

In this section, we formally define several measures of complexity of the linear systems we use. This is crucial, because we want to make sure that our reductions do not rely on mapping into extremely ill-conditioned matrices, and so we use these measures to show that this is in fact not the case.

Definition II.5.

1) Given a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, we define the maximum singular value $\sigma_{\max}(\boldsymbol{A})$ in the usual way as $\sigma_{\max}(\boldsymbol{A}) = \max_{\boldsymbol{x}\in\mathbb{R}^n, \boldsymbol{x}\neq \boldsymbol{0}} \sqrt{\frac{\boldsymbol{x}^\top \boldsymbol{A}^\top \boldsymbol{Ax}}{\boldsymbol{x}^\top \boldsymbol{x}}}$.
2) Given a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ which is not all zeros, we define the minimum non-zero singular value $\sigma_{\min}(\boldsymbol{A})$ as $\sigma_{\min}(\boldsymbol{A}) = \min_{\boldsymbol{x}\in\mathbb{R}^n, \boldsymbol{x}\perp \mathrm{null}(\boldsymbol{A})} \sqrt{\frac{\boldsymbol{x}^\top \boldsymbol{A}^\top \boldsymbol{Ax}}{\boldsymbol{x}^\top \boldsymbol{x}}}$.
3) Given a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ which is not all zeros, we define the non-zero condition number of $\boldsymbol{A}$ as $\kappa(\boldsymbol{A}) = \frac{\sigma_{\max}(\boldsymbol{A})}{\sigma_{\min}(\boldsymbol{A})}$.

Definition II.6. The sparse parameter complexity of an LSA instance $(\boldsymbol{A}, \boldsymbol{c}, \epsilon)$ where $\boldsymbol{A} \in \mathbb{Z}^{m\times n}$ and $\mathrm{nnz}(\boldsymbol{A}) \geq \max(m, n)$, and $\epsilon > 0$, is

$$\mathcal{S}(\boldsymbol{A}, \boldsymbol{c}, \epsilon) \stackrel{\text{def}}{=} \left(\mathrm{nnz}(\boldsymbol{A}), U(\boldsymbol{A}), \kappa(\boldsymbol{A}), \epsilon^{-1} \right),$$

where $U(\boldsymbol{A}) = \max\left(\left\| \boldsymbol{A} \right\|_{\max}, \left\| \boldsymbol{c} \right\|_{\max}, \frac{1}{\min_+(\boldsymbol{A})}, \frac{1}{\min_+(\boldsymbol{c})} \right)$.

Note in the definition above that when $\boldsymbol{A} \neq \boldsymbol{0}$ and $\boldsymbol{c} \neq \boldsymbol{0}$ have only integer entries, we trivially have $\min_+(\boldsymbol{A}) \geq 1$ and $\min_+(\boldsymbol{c}) \geq 1$. However, including $\frac{1}{\min_+(\boldsymbol{A})}$, and $\frac{1}{\min_+(\boldsymbol{c})}$ in the definition stated above is useful when working with intermediate matrices whose entries are not integer valued.

C. Matrix Classes and Reductions Between Them

We use the term *matrix class* to refer to an infinite set of matrices $\mathcal{M}$. In this section, we formally define a notion of efficient reduction between linear systems in different classes of matrices.

Definition II.7 (Efficient f-reducibility)**.** Suppose we have two matrix classes $\mathcal{M}_1$ and $\mathcal{M}_2$, and there exist two algorithms $\mathcal{A}_{1\to 2}$ and $\mathcal{A}_{1\leftarrow 2}$ s.t. given an LSA instance $(\boldsymbol{M}^1, \boldsymbol{c}^1, \epsilon)$, where $\boldsymbol{M}^1 \in \mathcal{M}_1$, the call

$\mathcal{A}_{1\to 2}(\boldsymbol{M}^1, \boldsymbol{c}^1, \epsilon_1)$ returns an LSA instance $(\boldsymbol{M}^2, \boldsymbol{c}^2, \epsilon_2)$ s.t. if $\boldsymbol{x}^2$ is a solution to LSA instance $(\boldsymbol{M}^2, \boldsymbol{c}^2, \epsilon_2)$ then $\boldsymbol{x}^1 = \mathcal{A}_{1\leftarrow 2}(\boldsymbol{M}^1, \boldsymbol{M}^2, \boldsymbol{x}^2)$ is a solution to LSA instance $(\boldsymbol{M}^1, \boldsymbol{c}^1, \epsilon_1)$.

Consider a function of $f : \mathbb{R}_+^4 \to \mathbb{R}_+^4$ s.t. every output coordinate is an increasing function of every input coordinate. Suppose that we always have

$$\mathcal{S}(\boldsymbol{M}^2, \boldsymbol{c}^2, \epsilon_2) \leq f(\mathcal{S}(\boldsymbol{M}^1, \boldsymbol{c}^1, \epsilon_1)),$$

and the running times of $\mathcal{A}_{1\to 2}(\boldsymbol{M}^1, \boldsymbol{c}^1, \epsilon_1)$ and $\mathcal{A}_{1\leftarrow 2}(\boldsymbol{M}^1, \boldsymbol{M}^2, \boldsymbol{x}^2)$ are both bounded by $O(\mathrm{nnz}(\boldsymbol{M}^1))$.

Then we say that $\mathcal{M}_1$ is efficiently f-reducible to $\mathcal{M}_2$, which we also write as

$$\mathcal{M}_1 \leq_f \mathcal{M}_2.$$

Lemma II.8. *Suppose $\mathcal{M}_1 \leq_f \mathcal{M}_2$ and $\mathcal{M}_2 \leq_g \mathcal{M}_3$. Then $\mathcal{M}_1 \leq_{g \circ f} \mathcal{M}_3$.*

Proof: The proof is simply by the trivial composition of the two reductions. ■

Definition II.9. We let $\mathcal{G}$ denote the class of all matrices with integer valued entries s.t. there is at least one non-zero entry in every row and column[6].

III. Main Results

In this section, we use the notions of sparse parameter complexity and matrix class reductions to prove our main technical result, Theorem III.1, which shows that linear systems in general matrices with integer entries can be efficiently reduced to linear systems in several different classes of Incidence Structured Block Matrices. From this result, we derive as corollary our main result, Theorem III.2, which states that fast high accuracy solvers for several types of ISBMs imply fast high accuracy solvers for all linear systems in general matrices with integer entries.

Theorem III.1. *Let $f(s, U, K, \epsilon) = (O(s\log(sU)), \mathrm{poly}(UK\epsilon^{-1}s), \mathrm{poly}(UK\epsilon^{-1}s), \mathrm{poly}(UK\epsilon^{-1}s))$, then*
1) $\mathcal{G} \leq_f \mathcal{MC}_{2,\mathbb{Z}}^{>0}$.
2) $\mathcal{G} \leq_f \mathcal{T}_2$.
3) $\mathcal{G} \leq_f \mathcal{V}_2$.

Theorem III.2. *Suppose we have an algorithm which solves every Linear System Approximation Problem $(\boldsymbol{A}, \boldsymbol{c}, \epsilon)$ with sparse parameter complexity $\mathcal{S}(\boldsymbol{A}, \boldsymbol{c}, \epsilon) \leq (s, U, K, \epsilon^{-1})$ in time $O(s^a \,\mathrm{polylog}(s, U, K, \epsilon^{-1}))$ for some $a \geq 1$, whenever $\boldsymbol{A} \in \mathcal{R}$ for at least one of $\mathcal{R} \in \left\{\mathcal{MC}_{2,\mathbb{Z}}^{>0}, \mathcal{T}_2, \mathcal{V}_2\right\}$. I.e. we have a "fast" solver[7] for one of the matrix classes $\mathcal{MC}_{2,\mathbb{Z}}^{>0}, \mathcal{T}_2$, or $\mathcal{V}_2$. Then every Linear System Approximation Problem $(\boldsymbol{A}, \boldsymbol{c}, \epsilon)$ where $\boldsymbol{A} \in \mathcal{G}$ with sparse parameter complexity $\mathcal{S}(\boldsymbol{A}, \boldsymbol{c}, \epsilon) \leq (s, U, K, \epsilon^{-1})$ can be solved in time $O(s^a \,\mathrm{polylog}(s, U, K, \epsilon^{-1}))$.*

Proof: The theorem is a immediate corollary of Theorem III.1. ■

Definition III.3. We let $\mathcal{G}_{z,2}$ denote the class of all matrices with integer valued entries s.t. there is at least one non-zero entry in every row and column, and every row has zero row sum, and for each row, the sum of the positive coefficients is a power of 2.

Lemma III.4. *Let $f(s, U, K, \epsilon) = \big(O(s), O\left(\epsilon^{-1}s^{9/2}U^3\right), O\left(\epsilon^{-1}s^8U^3K\right), O\left(s^{5/2}U^2\epsilon^{-1}\right)\big)$, then*

$$\mathcal{G} \leq_f \mathcal{G}_{z,2}.$$

Lemma III.5. *Let $f(s, U, K, \epsilon) = (O(s\log(sU)), O(s^{3/2}U\log^{1/2}(sU)), O(Ks^4U^2\log^2(sU)), O(sU^2\epsilon^{-1}))$, then*

$$\mathcal{G}_{z,2} \leq_f \mathcal{MC}_2.$$

Lemma III.6. *Let $f(s, U, K, \epsilon) = \big(O(s), O\left(\epsilon^{-1}U^2K\right), O\left(\epsilon^{-1}s^2U^2K\right), O(\epsilon^{-1})\big)$, then*

$$\mathcal{MC}_2 \leq_f \mathcal{MC}_2^{>0}.$$

Lemma III.7. *Let $f(s, U, K, \epsilon) = (s, \epsilon^{-1}sU, 2K, O(\epsilon^{-1}))$, then*

$$\mathcal{MC}_2^{>0} \leq_f \mathcal{MC}_{2,\mathbb{Z}}^{>0}.$$

Lemma III.8. *Let $f(s, U, K, \epsilon)$ be as defined in Lemma III.5 then*

$$\mathcal{G}_{z,2} \leq_f \mathcal{T}_2.$$

Lemma III.9. *Let $f(s, U, K, \epsilon) = (s, U, K, \epsilon^{-1})$, then*

$$\mathcal{MC}_2 \leq_f \mathcal{V}_2.$$

Proof of Theorem III.1: Follows by appropriate composition (Lemma II.8) applied to the the Lemmas above, i.e. III.4, III.5, III.6, III.7, III.8 and III.9. ■

The full version of this paper, available at https://arxiv.org/abs/1705.02944, presents proofs of all the lemmas stated in this section. We also give proofs of Lemmas III.8 and III.9 in the following sections.

IV. 2D Trusses

In this section, we prove Lemma III.8. We show that the reduction algorithm used in proving $\mathcal{G}_{z,2} \leq_f \mathcal{MC}_2$ constructs a 2D Truss Incidence Matrix as per Definition I.4. It follows that for any function f, $\mathcal{G} \leq_f \mathcal{MC}_2$ implies $\mathcal{G} \leq_f \mathcal{T}_2$. The key is to show that a 2-commodity gadget in the reduction corresponds to a 2D truss subgraph, which we call the 2D-truss gadget.

Without loss of generality, we let $\boldsymbol{u}$-variables correspond to the horizonal axis and $\boldsymbol{v}$-variables to the vertical axis of the 2D plane. According to Definition I.2 and I.4:

[6]If there is a row or column with only zeros, then it can always be handled trivially in the context of solving linear systems

[7]The reduction requires only a single linear system solve, and uses the solution in a black-box way. So the reduction also applies if the solver for the class $\mathcal{R}$ only works with high probability or only has running time guarantees in expectation.

1) an equation $\boldsymbol{u}_i - \boldsymbol{u}_j = 0$ in a 2-commodity linear system corresponds to a horizontal edge in the 2D plane;
2) an equation $\boldsymbol{v}_i - \boldsymbol{v}_j = 0$ in a 2-commodity linear system corresponds to a vertical edge in the 2D plane;
3) an equation $\boldsymbol{u}_i - \boldsymbol{v}_i - (\boldsymbol{u}_j - \boldsymbol{v}_j) = 0$ in a 2-commodity linear system corresponds to a diagonal edge in the 2D plane.

Note that our reduction here heavily relies on the ability to choose arbitrary weights. In particular, the weights on the elements are not related at all with the distances between the corresponding vertices.

Our strategy to pick the coordinates of the vertices of the constructed 2D truss is the following: we first pick the coordinates of the original n vertices randomly, and then determine the coordinates of the new vertices constructed in the reduction to satisfy all the truss equations.

For the n original vertices, we pick their $\boldsymbol{u}$-coordinates arbitrarily and pick their $\boldsymbol{v}$-coordinates randomly. Specifically, we pick an n-dimensional random vector $\boldsymbol{y}$ uniformly distributed on the n-dimensional sphere centered at the origin and with radius $R = n^{10}$; we then round each entry of $\boldsymbol{y}$ to have precision $\delta = 10^{-10}$, so that each entry has constant bits. Let $\tilde{\boldsymbol{y}}$ be the vector after rounding. We assign the $\boldsymbol{v}$-coordinate of the ith vertex to be the ith entry of $\tilde{\boldsymbol{y}}$.

We then pick the coordinates of the new vertices in the order they are created. Note that each time we replace two vertices in the current equations, say $\boldsymbol{s}^{j_1}, \boldsymbol{s}^{j_2}$, whose coordinates have already been determined, we create a 2D truss gadget with 7 new vertices, say $\boldsymbol{s}^t, \boldsymbol{s}^{t+1}, \ldots, \boldsymbol{s}^{t+6}$ [8]. According to the construction of this gadget, the new vertices $\boldsymbol{s}^{t+1}, \ldots, \boldsymbol{s}^{t+6}$ only appear in this single gadget, whose coordinates do not affect other vertices. Figure 1 is the corresponding subgraph which satisfies all the equations in the 2D truss gadget. Note the two triangles $(\boldsymbol{s}^{t+3}, \boldsymbol{s}^{t+5}, \boldsymbol{s}^{t+6})$ and $(\boldsymbol{s}^{t+3}, \boldsymbol{s}^{t+4}, \boldsymbol{s}^{t+5})$ need to be isosceles right triangles, which implies $\boldsymbol{v}_t = (\boldsymbol{v}_{j_1} + \boldsymbol{v}_{j_2})/2$. We can always place $\boldsymbol{s}^t, \boldsymbol{s}^{t+1}, \ldots, \boldsymbol{s}^{t+6}$ to get the desired equations, provided $\boldsymbol{v}_{j_1} \neq \boldsymbol{v}_{j_2}$, which is guaranteed with high probability by Lemma IV.1.

We prove the following lemma in the full version of this paper.

Lemma IV.1. *Let $\boldsymbol{a} \in \mathbb{R}^n$ be a fixed vector such that $-2 \le \boldsymbol{a}_i \le 2, \forall i \in [n]$ and $\boldsymbol{a}^\top \mathbf{1} = 0$. Let $\tilde{\boldsymbol{y}}$ be a vector picked as above. Then,*

$$\Pr\left(\boldsymbol{a}^\top \tilde{\boldsymbol{y}} = 0\right) \le \frac{2\delta n^2}{\|\boldsymbol{a}\|_2 R}.$$

By our construction of the truss, for each vertex, its $\boldsymbol{v}$-coordinate can be written as a fixed convex combination

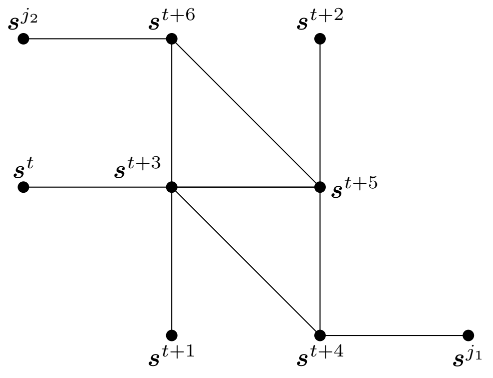

Figure 1. Geometric realization of the 2D truss gadget

of $\tilde{\boldsymbol{y}}$, say $\boldsymbol{c}^\top \tilde{\boldsymbol{y}}$ in which $\boldsymbol{c}^\top \mathbf{1} = 1$ and $\boldsymbol{c}_i \ge 0, \forall i \in [n]$. Consider a pair of two arbitrary vertices, let $\boldsymbol{c}^1$ and $\boldsymbol{c}^2$ be their coefficient vectors corresponding to the convex combinations. These two vertices have same $\boldsymbol{v}$-coordinate if and only if $(\boldsymbol{c}^1 - \boldsymbol{c}^2)^\top \tilde{\boldsymbol{y}} = 0$. Let $\boldsymbol{a} \overset{\text{def}}{=} \boldsymbol{c}^1 - \boldsymbol{c}^2$. Then, $-2 \le \boldsymbol{a}_i \le 2, \forall i \in [n]$, $\boldsymbol{a}^\top \mathbf{1} = 0$, and

$$\|\boldsymbol{a}\|_2 \le \left\|\boldsymbol{c}^1\right\|_2 + \left\|\boldsymbol{c}^2\right\|_2 \le \left\|\boldsymbol{c}^1\right\|_1 + \left\|\boldsymbol{c}^2\right\|_1 = 2.$$

By Lemma IV.1,

$$\Pr\left(\boldsymbol{c}^{1\top} \tilde{\boldsymbol{y}} = \boldsymbol{c}^{2\top} \tilde{\boldsymbol{y}}\right) \le \frac{\delta n^2}{R}.$$

By Lemma III.5, the total number of the vertices in the truss is at most $O\left(n^2 \log n\right)$. By a union bound, the probability that there exist two different vertices with same $\boldsymbol{v}$-coordinate is at most

$$\frac{\delta n^2}{R} \cdot O\left(n^2 \log n\right)^2 = O\left(\frac{\log^2 n}{n^4}\right).$$

Proof of Lemma III.8: Since the linear system for 2D trusses is the same as the linear system for 2-commodity, all complexity parameters of these two linear systems are the same. ■

V. Isotropic Total Variation Minimization

In this section, we prove Lemma III.9. We show the reduction algorithm used in proving $\mathcal{G}_{z,2} \le_f \mathcal{MC}_2$ constructs an 2-TV Incidence Matrix defined in Definition I.5. It follows that for any function f, $\mathcal{G} \le_f \mathcal{MC}_2$ implies $\mathcal{G} \le_f \mathcal{V}_2$.

Lemma III.9 follows from the claim below, which shows that the type $\boldsymbol{L}^{1+2} \otimes \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$ 2-commodity matrices that occur in GSBMs for 2-commodity ISBMs can be constructed as Total Variation matrices. The same is true for the other types of entries that occur in 2-commodity GSBMs.

Claim V.1. *For each edge (i, j) in the graph corresponding to $\boldsymbol{L}^{1+2}$, there exists an edge-vertex incidence matrix $\boldsymbol{N}$, a*

[8] See Algorithm 2 $\mathcal{MC}_2$Gadget in the full version for a detailed construction.

diagonal matrix $\boldsymbol{W}$ and a vector $\boldsymbol{r}$ such that $\boldsymbol{W} \succcurlyeq \boldsymbol{r}\boldsymbol{r}^\top$ and

$$\boldsymbol{L}_{ij}^{1+2} \otimes \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} = \boldsymbol{N}^\top \left(\boldsymbol{W} - \boldsymbol{r}\boldsymbol{r}^\top \right) \boldsymbol{N}.$$

Claim V.1 is proven in the full version of this paper.

Proof of Lemma III.9: Since the linear system related to the isotropic total variation minimization matrix is the same as the linear system for 2-commodity, all complexity parameters of these two linear systems are the same. ■

Acknowledgment

We thank Richard Peng and Daniel Spielman for helpful comments and discussions. We thank our anonymous reviewers for pointing out several typos.

References

[1] V. V. Williams and R. Williams, "Subcubic equivalences between path, matrix and triangle problems," in *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, ser. FOCS '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 645–654. [Online]. Available: http://dx.doi.org/10.1109/FOCS.2010.67

[2] V. Strassen, "Gaussian elimination is not optimal," *Numer. Math.*, vol. 13, no. 4, pp. 354–356, Aug. 1969. [Online]. Available: http://dx.doi.org/10.1007/BF02165411

[3] V. V. Williams, "Multiplying matrices faster than coppersmith-winograd," in *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '12. New York, NY, USA: ACM, 2012, pp. 887–898. [Online]. Available: http://doi.acm.org/10.1145/2213977.2214056

[4] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2003, available at http://www-users.cs.umn.edu/˜saad/toc.pdf.

[5] S. Boyd and L. Vandenberghe, *Convex Optimization*. Camebridge University Press, 2004, available at https://web.stanford.edu/˜boyd/cvxbook/bv_cvxbook.pdf.

[6] D. P. Woodruff, "Sketching as a tool for numerical linear algebra," *Theoretical Computer Science*, vol. 10, no. 1-2, pp. 1–157, 2014, available at http://arxiv.org/abs/1411.4357.

[7] D. Spielman and S. Teng, "Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems," *SIAM Journal on Matrix Analysis and Applications*, vol. 35, no. 3, pp. 835–885, 2014, available at http://arxiv.org/abs/cs/0607105.

[8] 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 mlog1/2n time," in *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, ser. STOC '14. New York, NY, USA: ACM, 2014, pp. 343–352. [Online]. Available: http://doi.acm.org/10.1145/2591796.2591833

[9] S.-H. Teng, "The Laplacian Paradigm: Emerging Algorithms for Massive Graphs," in *Theory and Applications of Models of Computation*, 2010, pp. 2–14.

[10] 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*, ser. FOCS '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 253–262. [Online]. Available: http://dx.doi.org/10.1109/FOCS.2013.35

[11] ——, "Computing maximum flow with augmenting electrical flows," in *Proceedings of the 2016 IEEE 57th Annual Symposium on Foundations of Computer Science*, ser. FOCS '16. Washington, DC, USA: IEEE Computer Society, 2016.

[12] M. B. Cohen, A. Madry, P. Sankowski, and A. Vladu, "Negative-weight shortest paths and unit capacity minimum cost flow in $\tilde{O}(m^{10/7} \log w)$ time: (extended abstract)," in *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '17. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2017, pp. 752–771. [Online]. Available: http://dl.acm.org/citation.cfm?id=3039686.3039734

[13] Y. T. Lee and A. Sidford, "Path finding methods for $\underline{\text{linear}}$ 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*. IEEE, 2014, pp. 424–433, available at http://arxiv.org/abs/1312.6677 and http://arxiv.org/abs/1312.6713.

[14] 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*, ser. STOC '08. New York, NY, USA: ACM, 2008, pp. 451–460, available at http://arxiv.org/abs/0803.0988. [Online]. Available: http://doi.acm.org/10.1145/1374376.1374441

[15] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother, "A comparative study of energy minimization methods for markov random fields with smoothness-based priors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, pp. 1068–1080, 2008.

[16] Y. Wang, J. Yang, W. Yin, and Y. Zhang, "A new alternating minimization algorithm for total variation image reconstruction," *SIAM Journal on Imaging Sciences*, vol. 1, no. 3, pp. 248–272, 2008.

[17] M. B. Cohen, Y. T. Lee, G. Miller, J. Pachocki, and A. Sidford, "Geometric median in nearly linear time," in *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*. ACM, 2016, pp. 9–21, available at: https://arxiv.org/abs/1606.05225.

[18] A. Singer and Y. Shkolnisky, "Three-dimensional structure determination from common lines in cryo-em by eigenvectors and semidefinite programming," *SIAM journal on imaging sciences*, vol. 4, no. 2, pp. 543–572, 2011.

[19] Y. Shkolnisky and A. Singer, "Viewing direction estimation in cryo-em using synchronization," *SIAM journal on imaging sciences*, vol. 5, no. 3, pp. 1088–1110, 2012.

[20] Z. Zhao and A. Singer, "Rotationally invariant image representation for viewing direction classification in cryo-em," *Journal of structural biology*, vol. 186, no. 1, pp. 153–166, 2014.

[21] B. Alexeev, A. S. Bandeira, M. Fickus, and D. G. Mixon, "Phase retrieval with polarization," *SIAM Journal on Imaging Sciences*, vol. 7, no. 1, pp. 35–66, 2014.

[22] S. Marchesini, Y.-C. Tu, and H.-t. Wu, "Alternating projection, ptychographic imaging and phase synchronization," *arXiv preprint arXiv:1402.0550*, 2014.

[23] O. Ozyesil, A. Singer, and R. Basri, "Stable camera motion estimation using convex programming," *SIAM Journal on Imaging Sciences*, vol. 8, no. 2, pp. 1220–1262, 2015.

[24] M. Arie-Nachimson, S. Z. Kovalsky, I. Kemelmacher-Shlizerman, A. Singer, and R. Basri, "Global motion estimation from point matches," in *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2012 Second International Conference on*. IEEE, 2012, pp. 81–88.

[25] J. A. Kelner, G. L. Miller, and R. Peng, "Faster approximate multicommodity flow using quadratically coupled flows," in *Proceedings of the 44th symposium on Theory of Computing*, ser. STOC '12. New York, NY, USA: ACM, 2012, pp. 1–18, available at http://arxiv.org/abs/1202.3367. [Online]. Available: http://arxiv.org/abs/1202.3367

[26] D. A. Spielman, "Nsf award 1562041: Generalized algebraic graph theory: Algorithms and analysis," *ALGORITHMIC FOUNDATIONS*, 2016.

[27] S. I. Daitch and D. A. Spielman, "Support-graph preconditioners for 2-dimensional trusses," *CoRR*, vol. abs/cs/0703119, 2007. [Online]. Available: http://arxiv.org/abs/cs/0703119

[28] 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*, ser. STOC '16. New York, NY, USA: ACM, 2016, pp. 842–850. [Online]. Available: http://doi.acm.org/10.1145/2897518.2897640

[29] 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 Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, 2014, pp. 204–216.

[30] N. Garg and J. Könemann, "Faster and simpler algorithms for multicommodity flow and other fractional packing problems," in *In Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, 1998, pp. 300–309.

[31] A. Madry, "Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms," in *STOC '10: Proceedings of the 42nd ACM symposium on Theory of computing*. New York, NY, USA: ACM, 2010, pp. 121–130.

[32] L. K. Fleischer, "Approximating fractional multicommodity flow independent of the number of commodities," *SIAM Journal on Discrete Mathematics*, vol. 13, pp. 505–520, 2000.

[33] T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos, and S. Tragoudas, "Fast approximation algorithms for multicommodity flow problems," in *JOURNAL OF COMPUTER AND SYSTEM SCIENCES*, 1991, pp. 487–496.

[34] J. Sherman, "Nearly maximum flows in nearly linear time," in *Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, ser. FOCS '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 263–269. [Online]. Available: http://dx.doi.org/10.1109/FOCS.2013.36

[35] 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*, ser. SODA '14. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2014, pp. 217–226. [Online]. Available: http://dl.acm.org/citation.cfm?id=2634074.2634090

[36] R. Peng, "Approximate undirected maximum flows in $o(mpoly\log(n))$ time," in *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '16. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2016, pp. 1862–1867. [Online]. Available: http://dl.acm.org/citation.cfm?id=2884435.2884565

[37] P. M. Vaidya, "Speeding-up linear programming using fast matrix multiplication," in *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 1989, pp. 332–337. [Online]. Available: http://portal.acm.org/citation.cfm?id=1398514.1398712

[38] Y. T. Lee and A. Sidford, "Efficient inverse maintenance and faster algorithms for linear programming," in *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*. IEEE, 2015, pp. 230–249, available at: https://arxiv.org/abs/1503.01752.

[39] G. Shklarski and S. Toledo, "Rigidity in finite-element matrices: Sufficient conditions for the rigidity of structures and substructures," *SIAM J. Matrix Analysis Applications*, vol. 30, no. 1, pp. 7–40, 2008. [Online]. Available: http://dx.doi.org/10.1137/060650295

[40] E. G. Boman, D. Chen, O. Parekh, and S. Toledo, "On factor width and symmetric h-matrices," *Linear algebra and its applications*, vol. 405, pp. 239–248, 2005.

[41] T. Chan and J. Shen, *Image Processing and Analysis: Variational, PDE, Wavelet, and Stochastic Methods*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2005.

[42] I. Koutis, G. L. Miller, and D. Tolliver, "Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing," *Computer Vision and Image Understanding*, vol. 115, no. 12, pp. 1638–1646, 2011.

[43] D. Goldfarb and W. Yin, "Second-order cone programming methods for total variation-based image restoration," *SIAM J. Sci. Comput*, vol. 27, pp. 622–645, 2004.

[44] B. Wohlberg and P. Rodriguez, "An iteratively reweighted norm algorithm for minimization of total variation functionals," *Signal Processing Letters, IEEE*, vol. 14, no. 12, pp. 948 –951, dec. 2007.

[45] H. H. Chin, A. Madry, G. L. Miller, and R. Peng, "Runtime guarantees for regression problems," in *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, ser. ITCS '13. New York, NY, USA: ACM, 2013, pp. 269–282.

[46] L. N. Trefethen and D. Bau III, *Numerical linear algebra*. Siam, 1997, vol. 50.

[47] D. Goldfarb and W. Yin, "Second-order cone programming methods for total variation-based image restoration," *SIAM Journal on Scientific Computing*, vol. 27, no. 2, pp. 622–645, 2005.

[48] H. H. Chin, A. Madry, G. L. Miller, and R. Peng, "Runtime guarantees for regression problems," in *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*. ACM, 2013, pp. 269–282.