MIT Open Access Articles

A simple, combinatorial algorithm for solving SDD systems in nearly-linear time

Massachusetts Institute of Technology

A Simple, Combinatorial Algorithm for Solving SDD Systems in Nearly-Linear Time

Jonathan A. Kelner
kelner@mit.edu
MIT

Lorenzo Orecchia
orecchia@mit.edu
MIT

Aaron Sidford
sidford@mit.edu
MIT

Zeyuan Allen Zhu
zeyuan@csail.mit.edu
MIT

Abstract

In this paper, we present a simple combinatorial algorithm that solves symmetric diagonally dominant (SDD) linear systems in nearly-linear time. It uses very little of the machinery that previously appeared to be necessary for a such an algorithm. It does not require recursive preconditioning, spectral sparsification, or even the Chebyshev Method or Conjugate Gradient. After constructing a "nice" spanning tree of a graph associated with the linear system, the entire algorithm consists of the repeated application of a simple (non-recursive) update rule, which it implements using a lightweight data structure. The algorithm is numerically stable and can be implemented without the increased bit-precision required by previous solvers. As such, the algorithm has the fastest known running time under the standard unit-cost RAM model. We hope that the simplicity of the algorithm and the insights yielded by its analysis will be useful in both theory and practice.

1 Introduction

A matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is *symmetric diagonally dominant* (SDD) if $\mathbf{A}^T = \mathbf{A}$ and $\mathbf{A}_{ii} \geq \sum_{j \neq i} |\mathbf{A}_{ij}|$ for all $i \in [n]$. While the best known algorithm for solving a general linear system takes time $O(n^{2.373})$ [Wil12], a seminal paper by Spieman and Teng [ST04] showed that when $\mathbf{A}$ is SDD one can solve $\mathbf{A}\vec{x} = \vec{b}$ approximately in nearly linear time. [1]

Fast algorithms for solving SDD linear systems have found broad applications across both the theory and practice of computer science. They have long been central to scientific computing, where solving SDD systems is the main computational task in modeling of electrical networks of resistors and performing finite element simulations of a wide range of physical systems (see, e.g., [BHV08]). Beyond this, SDD system solvers have been applied to foundational problems in a wide range of other fields, including machine learning, random processes, computer vision, image processing, network analysis, and computational biology (see, for example, [LLDM08, KMT11, LLB+09, VTX09, HK11]).

More recently, SDD solvers have emerged as a powerful tool in the design of graph algorithms. To every graph G, one can associate an SDD matrix $\mathbf{L} = \mathbf{L}_G$ called its *Laplacian* (defined in Section 2) such that there are deep connections between the combinatorial properties of G and the linear algebraic properties of $\mathbf{L}$. By exploiting these connections, researchers have used nearly linear time algorithms for solving SDD systems to break longstanding barriers and provide new algorithms for a wide and rapidly growing list of fundamental

---

[1] Throughout this paper we are primarily interested in *approximate linear system solvers*, that is algorithms that can compute $\vec{x} \in \mathbb{R}^n$ such that $\|\vec{x} - \vec{x}_{opt}\|_A \leq \varepsilon \|\vec{x}_{opt}\|_A$ for any $\varepsilon \in \mathbb{R} > 0$ where $x_{opt} \in \mathbb{R}^n$ is a vector such that $\mathbf{A}\vec{x}_{opt} = \vec{b}$. When we refer to a *nearly linear time SDD system solver* we mean an an algorithm that computes such a x in time $O(m \log^c n \log \varepsilon^{-1})$ where m is the number of nonzero entries in A and $c \geq 0 \in \mathbb{R}$ is a fixed constant.

graph problems, including maximum flow problems [CKM⁺11], multi-commodity flow problems [KMP12], generating random spanning tree [KM09], graph sparsification [SS11], lossy flow problems [DS08], sparsest cut [She09], distributed routing [KM11], and balanced separator [OSV12], as well as fundamental linear algebraic problems for SDD matrices, including computing the matrix exponential [OSV12] and the largest eigenvalue and corresponding eigenvector [ST12]. For surveys of these solvers and their applications, see [Spi12, Ten10, Vis].

1.1   Previous Nearly Linear Time Algorithms

The first nearly linear time algorithm for solving SDD systems was given by Spielman and Teng [ST04], building on a long line of previous work (e.g., [Vai90, GMZ95, BGH⁺06, BH03, BCHT04]). Their algorithm and its analysis is a technical tour-de-force that required multiple fundamental innovations in spectral and combinatorial graph theory, graph algorithms, and computational linear algebra. Their work included the invention of spectral sparsification and ultra-sparsifiers, better and faster constructions of low-stretch spanning trees, and efficient local clustering algorithms, all of which was used to construct and analyze an intricate recursively preconditioned iterative solver. They divided this work into three papers totaling over 130 pages ([ST08a, ST12, ST08b]), each of which has prompted a new line of inquiry and substantial follow-up work. Their work was was followed by two beautifully insightful papers by Koutis, Miller, and Peng that simplified the SDD system solver while improving its running time to $O(m \log n \log \log n \log \varepsilon^{-1})$ [KMP10, KMP11]. For a more in-depth discussion of the history of this work see [ST12].

These algorithms all rely on the same general framework. They reduce solving general SDD systems to solving systems in graph Laplacians. Given a graph, they show how to obtain a sequence of logarithmically many successively sparser graphs that approximate it, which they construct by adding carefully chosen sets of edges to a low-stretch spanning tree. They then show that the relationship between the combinatorial properties of a graph and the spectral properties of its Laplacian enables them to use the Laplacian of each graph in this sequence as a preconditioner for the one preceding it in a recursively applied iterative solver, such as the Preconditioned Chebyshev Method or Preconditioned Conjugate Gradient.

We remark that multigrid methods (see, e.g., [BHM00]), which are widely used in practice on graphs with sufficiently nice topologies, can be thought of as following a similar multilevel recursively preconditioned iterative framework. Indeed, Koutis *et al.* have an algorithm and implementation based on related techniques that they refer to as "combinatorial multigrid" [BKMT10].Even if one does not demand provable running time bounds, we are not aware of any algorithm whose running time empirically scales nearly linearly on large classes of input graphs that does not roughly follow this general structure.

1.2   Our Results

In this paper, we present a new, simple, combinatorial algorithm that solves SDD systems and has a running time of $O\big(m \log^2 n \log \log n \log(\varepsilon^{-1})\big)$. It uses very little of the machinery that previously appeared to be necessary for a nearly linear time algorithm. It does not require spectral sparsifiers (or variants such as ultra-sparsifiers or incremental sparsifiers), recursive preconditioning, or even the Chebyshev Method or Conjugate Gradient.

To solve an SDD system all our solver requires is a single low-stretch spanning tree[2] of G (not a recursive collection of subgraphs), and a straightforward data structure. Given these, the algorithm can be described in a few lines of pseudocode, and its analysis can be made to fit on a single blackboard.

Due to the complexity of previous nearly linear time solvers and the intricate and delicate nature of their analyses, it was necessary to apply them as a black box. By providing a new, easy-to-understand algorithm, it is our hope that algorithms that use SDD solvers can be improved by "opening up" this black box and modifying it to take advantage of specific features of the problem, and that similar techniques can be applied

---

[2]This can be obtained in nearly-linear time by a simple ball-growing algorithm [AKPW95]; the constructions with the best known parameters use a more intricate, but still nearly-linear-time, region growing technique [EEST05, ABN08, KMP10, AN12]. In Section 11 we discuss how even this requirement can be relaxed.

to related problems (e.g., ones with additional constraints or slight deviations from linearity or diagonal dominance).

Because of the lightweight nature of the algorithm and data structure, we expect it to be fast in practice. Furthermore, the analysis is quite robust, and we believe that the algorithm can be readily adapted to work in multicore, distributed, and even asynchronous settings. We hope to evaluate our algorithm empirically in a follow-up paper.

Numerical Stability and Run-time Comparisons Existing algorithms relied on Preconditioned Chebyshev methods, whose numerical stability is quite difficult to analyze. At present, the best known results show that they can be implemented with finite-precision arithmetic, but the number of bits of precision required is $\log \kappa(\mathbf{L}) \log^c n \log \varepsilon^{-1}$, where $\kappa(\mathbf{L})$ is the condition number of $\mathbf{L}$, and c is some possibly large constant [ST12]. The stated running time of the best existing algorithm is $O(m \log n \log \varepsilon^{-1})$ (ignoring $O(\mathsf{poly}(\log \log n))$ terms and using the analysis from [KMP11] and the best known low-stretch spanning tree algorithm [AN12]), but this assumes arbitrary precision arithmetic. If one analyzes it in the more standard unit-cost RAM model, where one can perform operations only on $O(\log n)$-bit numbers in constant time, this introduces several additional logarithmic factors in the running time.

In contrast, we show that our algorithm is numerically stable and does not require this additional overhead in the bit precision. As such, our algorithm gives the fastest known algorithm for solving SDD systems in the unit-cost RAM model. Our algorithm approximately solves both SDD systems and the dual electrical flow problem in time $O(m \log^2 n \log \log n \log(\varepsilon^{-1}))$. If one allows infinite precision arithmetic to be performed in constant time, this is slower than [KMP11] by a factor of $O(\log n)$ for the SDD system problem and by a factor of $O(\min\{\log(\varepsilon^{-1}), \log n\})$ for the electrical flow problem (ignoring $O(\mathsf{poly}(\log \log n))$ terms).[3]

1.3   Overview of our approach

Using standard reductions techniques we reduce solving arbitrary SDD systems to solving $\mathbf{L}\vec{v} = \vec{\chi}$, where $\mathbf{L} \in \mathbb{R}^{n \times n}$ is the Laplacian of a weighted connected graph $G = (V, E, W)$ with n vertices and m edges and $\vec{\chi} \in \mathbb{R}^n$, see Appendix A for details. To simplify this overview, suppose for now that $\vec{\chi} = \vec{e}_s - \vec{e}_t$ and $\vec{e}_s$ and $\vec{e}_t$ are the unit basis vectors corresponding to two vertices in the graph, s and t respectively. (The general case is similar and discussed in the following sections.)

Such Laplacian systems $\mathbf{L}\vec{v} = \vec{\chi}$ can be viewed as *electrical flow problems*: each edge $e \in E$ can be viewed as a resistor of resistance $r_e = 1/w_e$, where w_e is the weight of this edge, and one unit of electric current needs to be sent from s to t. If $\vec{v}$ is a valid solution to $\mathbf{L}\vec{v} = \vec{\chi}$, then the entries of $\vec{v}$ can be viewed as the *electrical potentials* of this system, and the amount of *electric current or flow* $\vec{f}(e)$ on an edge $e = (i, j)$ from i to j is given by $(\vec{v}_i - \vec{v}_j)/r_e$. The fact that electric flows are induced by vertex potential differences is a crucial property of electrical systems that our algorithm is going to exploit.

While previous solvers worked with the potential vector $\vec{v}$, our algorithm works with the flow vector $\vec{f}$. Our algorithm begins with any arbitrary unit *s-t* flow (e.g., a path from s to t) and maintains its feasibility throughout the execution. If $\vec{f}$ were a valid electrical flow, then it would be induced by some potential vector $\vec{v} \in \mathbb{R}^V$ satisfying $\vec{f}(e) = (\vec{v}_i - \vec{v}_j)/r_e$ for all edges $e = (i, j) \in E$, and in particular, for any cycle C in G, we would have the potential drop $\sum_{e \in C} \vec{f}(e) r_e = 0$. Exploiting this fact, our entire algorithm consists of repeatedly doing the following:

  - Randomly sample a cycle C from some probability distribution.
  - Compute $\sum_{e \in C} \vec{f}(e) r_e$ and if it is non-zero then add a multiple of C to $\vec{f}$ to make it zero.

To turn this into an provably-correct and efficient algorithm we need to do the following:

  • **Specify the cycle distribution.** The cycles are those found by adding edges to a low-stretch spanning tree. They are sampled proportional to their stretch (See Section 2.2 and 3).

---

[3]To the best of our knowledge, to convert their SDD system solver to an ε'-approximate electrical flow solver, one needs to pick $\varepsilon = O(\frac{\varepsilon'}{n})$.

- **Bound the number of iterations.** In Section 4, we will show that repeating this process a nearly linear number of times suffices to yield an ε-approximate solution.

- **Implement the iterations efficiently.** Since a cycle may contain a large number of edges, we cannot simply update the flow edge-by-edge. In Section 5, we give a data structure that allows each iteration to take $O(\log n)$ time and we discuss further implementation details.

Interestingly, the algorithm and its analysis also have a geometric interpretation, based on the Kaczmarz method [Kac37], which iteratively solves a linear system by alternately picking a constraint and projecting the current solution onto its feasible set. In particular, the algorithm can be recast as an instance of the randomized Kaczmarz method of Strohmer and Vershynin [SV09]. This geometric view is presented in Section 9.

In addition, in Section 7 we show how to improve the running time of our algorithm, in Section 8 we prove the numerical stability of our algorithm, in Section 10 we prove that our algorithm can be viewed as a linear operator that spectrally approximates the Laplacian pseudoinverse, and in Section 11 we discuss how to extend our algorithm to solve SDD systems without using low stretch spanning trees. For the reader who simply wants a complete proof of how to solve SDD systems in nearly-linear time, Section 2 - 6 and Appendix A suffice.

2 Preliminaries

Throughout the remainder of the paper let $G = (V, E, w)$ be a weighted, connected, undirected graph with $n = |V|$ vertices, $m = |E|$ edges and edge weights $w_e > 0$. We think of w_e as the *conductance* of e, and we define the *resistance* of e by $r_e \overset{\text{def}}{=} 1/w_e$. For notational convenience, we fix an orientation of the edges so that for any vertices a and b connected by an edge, exactly one of $(a, b) \in E$ or $(b, a) \in E$ holds.

We make extensive use of the following matrices associated to G:

Definition 2.1 (Matrix Definitions). *We define the* incidence matrix, $\mathbf{B} \in \mathbb{R}^{E \times V}$, *the* resistance matrix, $\mathbf{R} \in \mathbb{R}^{E \times E}$, *and the* Laplacian, $\mathbf{L} \in \mathbb{R}^{V \times V}$, *for all* $(a, b), e_1, e_2 \in E$, *and* $a, b, c \in V$, *by*

$$
\mathbf{B}_{(a,b),c} = \begin{cases} 1 & a = c \\ -1 & b = c \\ 0 & \text{otherwise} \end{cases} \quad , \quad \mathbf{R}_{e_1,e_2} = \begin{cases} r_e & e = e_1 = e_2 \\ 0 & \text{otherwise} \end{cases} \quad , \quad \mathbf{L}_{a,b} = \begin{cases} \sum_{\{a,u\} \in E} w_{a,u} & a = b \\ -w_{a,b} & \{a,b\} \in E \\ 0 & \text{otherwise} \end{cases}
$$

For a vector $\vec{f} \in \mathbb{R}^E$ and an edge $e = (a, b) \in E$, we write $\vec{f}(e) = \vec{f}(a, b)$ for the coordinate of $\vec{f}$ corresponding to e, and we adopt the convention that $\vec{f}(b, a) = -\vec{f}(a, b)$. This allows us to think of $\vec{f}$ as a flow on the graph (not necessarily obeying any conservation constraints) that sends $\vec{f}(e)$ units of flow from a to b, and thus $-\vec{f}(e)$ units of flow from b to a.

The following facts follow by simple manipulations of the above definitions:

Claim 2.2. *For all* $\vec{f} \in \mathbb{R}^E$, $x \in \mathbb{R}^V$, $a \in V$ *and* $(a, b) \in E$:

- $\left[\mathbf{B}^T \vec{f}\right]_a = \sum_{(b,a) \in E} \vec{f}(b, a) - \sum_{(a,b) \in E} \vec{f}(a, b)$,
- $\mathbf{L} = \mathbf{B}^T \mathbf{R}^{-1} \mathbf{B}$,
- $[\mathbf{B}x]_{(a,b)} = x(a) - x(b)$, *and*
- $x^T \mathbf{L} x = \sum_{(a,b) \in E} \frac{(x_a - x_b)^2}{r_{a,b}}$.

One can interpret the first assertion in Claim 2.2 as saying that $\mathbf{B}^T \vec{f}$ is a vector in $\mathbb{R}^V$ whose a-th coordinate indicates how much flow $\vec{f}$ leaves (or enters, if it is negative) the graph G at vertex a. We say that $\vec{f}$ is a *circulation* if $\mathbf{B}^T \vec{f} = 0$.

2.1 Electrical Flow

For any vector $\vec{f} \in \mathbb{R}^E$, we define its *energy* $\xi_r(\vec{f})$ by

$$\xi_r(\vec{f}) \stackrel{\text{def}}{=} \sum_{e \in E} r_e \vec{f}(e)^2 = \vec{f}^T \mathbf{R} \vec{f} = \|\vec{f}\|_\mathbf{R}^2 \ .$$

We fix a *demand vector* $\vec{\chi} \in \mathbb{R}^V$ and we say a flow $\vec{f} \in \mathbb{R}^E$ is *feasible (with respect to $\vec{\chi}$)*, or that it *meets the demands*, if $\mathbf{B}^T \vec{f} = \vec{\chi}$. Since G is connected it is straightforward to check that there exists a feasible flow with respect to $\vec{\chi}$ if and only if $\sum_{v \in V} \vec{\chi}(v) = 0$.

Definition 2.3 (Electrical Flow). *For a demand vector $\vec{\chi} \in \mathbb{R}^V$ satisfying $\sum_{a \in V} \vec{\chi}(a) = 0$, the* electrical flow satisfying $\vec{\chi}$ *is the unique minimizer to the following*

$$\vec{f}_{\text{opt}} \stackrel{\text{def}}{=} \underset{\vec{f} \in \mathbb{R}^E \ : \ \mathbf{B}^T \vec{f} = \vec{\chi}}{\arg \min} \ \xi_r(\vec{f}) \ . \tag{1}$$

This quadratic program describes a natural physical problem. Given an electric circuit with nodes in V, for each undirected edge $e = \{a, b\} \in E$ we connect nodes a and b with a resistor of resistance r_e. Next, we fix the amount of current entering and leaving each node and denote this by demand vector $\vec{\chi}$. Recalling from physics that the energy of sending i units of current over a resistor of resistance r is $i^2 \cdot r$ the amount of electric current on each resistor is given by $\vec{f}_{\text{opt}}$.

The central problem of this paper is to efficiently compute an ε-approximate electrical flow.

Definition 2.4 (ε-Approximate Electrical Flow). *For any $\varepsilon \in \mathbb{R}^{\geq 0}$, we say $\vec{f} \in \mathbb{R}^E$ is an ε-approximate electric flow satisfying $\vec{\chi}$ if $\mathbf{B}^T \vec{f} = \vec{\chi}$, and $\xi_r(\vec{f}) \leq (1 + \varepsilon) \cdot \xi_r(\vec{f}_{\text{opt}})$.*

2.1.1 Duality

The electric flow problem is dual to solving $\mathbf{L}\vec{x} = \vec{\chi}$ when $\mathbf{L}$ is the Laplacian for the same graph G. To see this, we note that the Lagrangian dual of (1) is given by

$$\max_{\vec{v} \in \mathbb{R}^V} 2\vec{v}^T \vec{\chi} - \vec{v}^T \mathbf{L} \vec{v} \ . \tag{2}$$

For symmetry we define the *(dual) energy* of $\vec{v} \in \mathbb{R}^V$ as $\zeta_r(\vec{v}) \stackrel{\text{def}}{=} 2\vec{v}^T \vec{\chi} - \vec{v}^T \mathbf{L} \vec{v}$. Setting the gradient of (2) to 0 we see that (2) is minimized by $\vec{v} \in \mathbb{R}^V$ satisfying $\mathbf{L}\vec{v} = \vec{\chi}$. Let $\mathbf{L}^\dagger$ denote the *Moore-Penrose pseduoinverse* of $\mathbf{L}$ and let $\vec{v}_{\text{opt}} \stackrel{\text{def}}{=} \mathbf{L}^\dagger \vec{\chi}$ denote a particular set of optimal voltages . Since the primal program (1) satisfies Slater's condition, we have strong duality, so for all $\vec{v} \in \mathbb{R}^V$

$$\zeta_r(\vec{v}) \leq \zeta_r(\vec{v}_{\text{opt}}) = \xi_r(\vec{f}_{\text{opt}}) \ .$$

Therefore, for feasible $\vec{f} \in \mathbb{R}^E$ and $\vec{v} \in \mathbb{R}^V$ the *duality gap*, $\text{gap}(\vec{f}, \vec{v}) \stackrel{\text{def}}{=} \xi_r(\vec{f}) - \zeta_r(\vec{v})$ is an upper bound on both $\xi_r(\vec{f}) - \xi_r(\vec{f}_{\text{opt}})$ and $\zeta_r(\vec{v}_{\text{opt}}) - \zeta_r(\vec{v})$.

In keeping with the electrical circuit interpretation we refer to a candidate dual solution $\vec{v} \in \mathbb{R}^V$ as *voltages* or *vertex potentials* and we define $\Delta_{\vec{v}}(a, b) \stackrel{\text{def}}{=} \vec{v}(a) - \vec{v}(b)$ to be the *potential drop* of $\vec{v}$ across (a, b). By the KKT conditions we know that

$$\vec{f}_{\text{opt}} = \mathbf{R}^{-1} \mathbf{B} \vec{v}_{\text{opt}} \quad \text{i.e.} \quad \forall e \in E \ : \ \vec{f}_{\text{opt}}(e) = \frac{\Delta_{\vec{v}_{\text{opt}}}(e)}{r_e} \ .$$

For $e \in E$ we call $\frac{\Delta_{\vec{v}}(e)}{r_e}$ the *flow induced by $\vec{v}$ across e* and we call $\vec{f}(e)r_e$ the *potential drop induced by $\vec{f}$ across e*.

The optimality conditions $\vec{f}_{\text{opt}} = \mathbf{R}^{-1} \mathbf{B} \vec{v}_{\text{opt}}$ can be restated solely in terms of flows in a well known variant of *Kirchoff's Potential Law* (KPL) as follows

Lemma 2.5 (KPL). *Feasible $\vec{f} \in \mathbb{R}^E$ is optimal if and only if $\vec{f}^T \mathbf{R} \vec{c} = 0$ for all circulations $\vec{c} \in \mathbb{R}^E$.*

2.2 Spanning Trees and Cycle Space

Let $T \subseteq E$ be a spanning tree of G and let us call the edges in T the *tree edges* and the edges in $E \setminus T$ the *off-tree edges*. Now, by the fact that G is connected and T spans G we know that for every $a, b \in V$ there is a unique path connecting a and b using only tree edges.

Definition 2.6 (Tree Path). *For $a, b \in V$, we define the* tree path $P_{(a,b)} \subseteq V \times V$ *to be the unique path from a to b using edges from T.* [4] *In vector form we let $\vec{p}_{(a,b)} \in \mathbb{R}^E$ denote the unique flow sending 1 unit of flow from a to b, that is nonzero only on T.*

For the off-tree edges we similarly define *tree cycles*.

Definition 2.7 (Tree Cycle). *For $(a, b) \in E \setminus T$, we define the* tree cycle $C_{(a,b)} \overset{\text{def}}{=} \{(a, b)\} \cup P_{(b,a)}$ *to be the unique cycle consisting of edge (a, b) and $P_{(b,a)}$. In vector form we let $\vec{c}_{(a,b)}$ denote the unique circulation sending 1 unit of flow on $C_{(a,b)}$.*

2.2.1 Cycle Space

The tree cycles form a complete characterization of circulations in a graph. The set of all circulations $\{\vec{c} \in \mathbb{R}^E \mid \mathbf{B}^T \vec{c} = 0\}$ is a well-known subspace called *cycle space* [Bol98] and the tree cycles $\{\vec{c}_e \mid e \in E \setminus T\}$ form a basis. This yields an even more succinct description of the KPL optimality condition (Lemma 2.5). A feasible $\vec{f} \in \mathbb{R}^E$ is optimal if and only if $\vec{f}^T R \vec{c}_e = 0$ for all $e \in E \setminus T$.

We can think of each tree cycle C_e as a long resistor consisting of its edges in series with total resistance $\sum_{e \in C_e} r_e$ and flow induced potential drop of $\sum_{e \in C_e} \vec{f}(e) r_e$. KPL optimality then states that $\vec{f} \in \mathbb{R}^E$ is optimal if and only if the potential drop across each of these resistors is 0. Here we define two key quantities relevant to this view.

Definition 2.8 (Cycle Quantities). *For $e \in E \setminus T$ and $\vec{f} \in \mathbb{R}^E$ the* resistance of C_e, R_e, *and the* flow induced potential across C_e, $\Delta_{c_e}(\vec{f})$, *are given by*

$$R_e \overset{\text{def}}{=} \sum_{e' \in C_e} r_{e'} = \vec{c}_e^T \mathbf{R} \vec{c}_e \quad and \quad \Delta_{c_e}(\vec{f}) \overset{\text{def}}{=} \sum_{e \in C_e} r_e \vec{f}(e) = \vec{f}^T \mathbf{R} \vec{c}_e \ .$$

2.2.2 Low-Stretch Spanning Trees

Starting with a feasible flow, our algorithm computes an approximate electrical flow by fixing violations of KPL on randomly sampled tree cycles. How well this algorithm performs is determined by how well the resistances of the off-tree edges are approximated by their corresponding cycle resistances. This quantity, which we refer to as as the *tree condition number*, is in fact a certain condition number of a matrix whose rows are properly normalized instances of $\vec{c}_e$ (see Section 9).

Definition 2.9 (Tree Condition Number). *The* tree condition number *of spanning tree T is given by* $\tau(T) \overset{\text{def}}{=} \sum_{e \in E \setminus T} \frac{R_e}{r_e}$ *, and we abbreviate it as τ when the underlying tree T is clear from context.*

This is closely related to a common quantity associated with a spanning tree called *stretch*.

Definition 2.10 (Stretch). *The* stretch *of $e \in E$, $\text{st}(e)$, and the* total stretch *of T, $\text{st}(T)$, are*

$$\text{st}(e) \overset{\text{def}}{=} \frac{\sum_{e' \in P_e} r_{e'}}{r_e} \quad and \quad \text{st}(T) \overset{\text{def}}{=} \sum_{e \in E} \text{st}(e) \ .$$

---

[4]Note that the edges of $P_{(a,b)}$ are oriented with respect to the path, not the natural orientation of G.

Since $R_e = r_e \cdot (1 + \text{st}(e))$ we see that these quantities are related by $\tau(T) = \text{st}(T) + m - 2n + 2$.

Efficient algorithms for computing spanning trees with low total or average stretch, i.e. *low-stretch spanning trees*, have found numerous applications [AKPW95, EEST05] and all previous nearly-linear-time SDD-system solvers [ST04, ST12, KMP10, KMP11], including the most efficient form of the SDD solver presented in this paper, make use of such trees. There have been multiple breakthroughs in the efficient construction of *low-stretch spanning trees* [AKPW95, EEST05, ABN08, KMP10, AN12] and the latest such result is used in this paper and stated below.

Theorem 2.11 ([AN12]). *In $O(m \log n \log \log n)$ time we can compute a spanning tree T with total stretch $\text{st}(T) = O(m \log n \log \log n)$.*

3 A Simple Nearly-Linear Time SDD Solver

Given a SDD system $\mathbf{A}\vec{x} = \vec{b}$ we wish to efficiently compute $\vec{x}$ such that $\|\vec{x} - \mathbf{A}^\dagger \vec{b}\|_{\mathbf{A}} \le \varepsilon \|\mathbf{A}^\dagger \vec{b}\|_A$. Using standard reduction techniques (detailed in Appendix A) we can reduce solving such SDD systems to solving Laplacian systems corresponding to connected graphs without a loss in asymptotic run time. Therefore it suffices to solve $\mathbf{L}\vec{v} = \vec{\chi}$ in nearly-linear time when $\mathbf{L}$ is the Laplacian matrix for some connected graph G. Here we provide an algorithm that both solves such systems and computes the corresponding ε-approximate electric flows in $O(m \log^2 n \log \log n \log(\varepsilon^{-1} n))$ time.

3.1 A Simple Iterative Approach

Our algorithm focuses on the electric flow problem. First, we compute a low stretch spanning tree, T, and a crude initial feasible $\vec{f}_0 \in \mathbb{R}^E$ taken to be the unique $\vec{f}_0$ that meets the demands and is nonzero only on tree edges. Next, for a fixed number of iterations, K, we perform simple iterative steps, referred to as *cycle updates*, in which we compute a new feasible $\vec{f}_i \in \mathbb{R}^E$ from the previous feasible $\vec{f}_{i-1} \in \mathbb{R}^E$ while attempting to decrease energy. Each iteration, i, consists of sampling an $e \in E \setminus T$ proportional to $\frac{R_e}{r_e}$, checking if $\vec{f}_{i-1}$ violates KPL on C_e (i.e. $\Delta_{c_e}(\vec{f}_i) \ne 0$) and adding a multiple of $\vec{c}_e$ to make KPL hold on C_e (i.e. $\vec{f}_i = \vec{f}_{i-1} - \frac{\Delta_{c_e}(\vec{f}_{i-1})}{R_e} \vec{c}_e$). Since, $\mathbf{B}^T \vec{c}_e = 0$, this operation preserves feasibility. We show that in expectation $\vec{f}_K$ is an ε-approximate electrical flow.

To solve $\mathbf{L}\vec{v} = \vec{\chi}$ we show how to use an ε-approximate electrical flow to derive a candidate solution to $\mathbf{L}\vec{v} = \vec{\chi}$ of comparable quality. In particular, we use the fact that a vector $\vec{f} \in \mathbb{R}^E$ and a spanning tree T induce a natural set of voltages, $\vec{v} \in \mathbb{R}^V$ which we call the *tree induced voltages*.

Definition 3.1 (Tree Induced Voltages). *For $\vec{f} \in \mathbb{R}^E$ and an arbitrary (but fixed) $s \in V$,[5] we define the tree induced voltages $\vec{v} \in \mathbb{R}^V$ by $\vec{v}(a) \overset{\text{def}}{=} \sum_{e \in P_{(a,s)}} \vec{f}(e) r_e$ for $\forall a \in V$.*

Our algorithm simply returns the tree induced voltages for $\vec{f}_K$, denoted $\vec{v}_K$, as the approximate solution to $\mathbf{L}\vec{v} = \vec{\chi}$. The full pseudocode for the algorithm is given in Algorithm 1.

3.2 Algorithm Guarantees

In the next few sections we prove that `SimpleSolver` both computes an ε-approximate electric flow and solves the corresponding Laplacian system in nearly linear time:

Theorem 3.2 (`SimpleSolver`). *The output of `SimpleSolver` satisfies [6]*

$$\mathbb{E}[\xi_r(\vec{f})] \le (1 + \varepsilon) \cdot \xi_r(\vec{f}_{\text{opt}}) \quad \text{and} \quad \mathbb{E}\|\vec{v} - \mathbf{L}^\dagger \vec{\chi}\|_{\mathbf{L}} \le \sqrt{\varepsilon} \cdot \|\mathbf{L}^\dagger \vec{\chi}\|_{\mathbf{L}}$$

and `SimpleSolver` can be implemented to run in time $O(m \log^2 n \log \log n \log(\varepsilon^{-1} n))$.

---

[5] We are primarily concerned with difference between potentials which are invariant under the choice $s \in V$.

[6] Although the theorem is stated as an expected guarantee on $\vec{f}$ and $\vec{v}$, one can easily use Markov bound and Chernoff bound to provide a probabilistic but exact guarantee.

Algorithm 1: `SimpleSolver`

Input : $G = (V, E, r)$, $\vec{\chi} \in \mathbb{R}^V$, $\varepsilon \in \mathbb{R}^+$
Output: $\vec{f} \in \mathbb{R}^E$ and $\vec{v} \in \mathbb{R}^V$

$T :=$ low-stretch spanning tree of G;
$\vec{f}_0 :=$ unique flow on T such that $\mathbf{B}^T \vec{f}_0 = \vec{\chi}$;
$\vec{p}_e := \frac{1}{\tau(T)} \cdot \frac{R_e}{r_e}$ for all $e \in E \setminus T$;
$K = \lceil \tau \log \left(\frac{\mathrm{st}(T) \cdot \tau(T)}{\varepsilon} \right) \rceil$;
for $i = 1$ *to* K **do**
  Pick random $e_i \in E \setminus T$ by probability distribution $\vec{p}$;
  $\vec{f}_i = \vec{f}_{i-1} - \frac{\Delta_{c_e}(\vec{f}_{i-1})}{R_e} \vec{c}_e$;
end

return $\vec{f}_K$ *and its tree induced voltages* $\vec{v}_K$

By construction `SimpleSolver` outputs a feasible flow and, by choosing T to be a low stretch spanning tree with properties guaranteed by Theorem 2.11, we know that the number of iterations of simple solver is bounded by $O(m \log n \log \log n \log(\varepsilon^{-1} n))$. However, in order to prove the theorem we still need to show that (1) each iteration makes significant progress, (2) each iteration can be implemented efficiently, and (3) the starting flow and final voltages are appropriately related to $\xi_r(\vec{f}_{\mathrm{opt}})$. In particular we show:

1. Each iteration of `SimpleSolver` decreases the energy of the current flow by at least an expected $\left(1 - \frac{1}{\tau}\right)$ fraction of the energy distance to optimality (see Section 4).

2. Each iteration of `SimpleSolver` can be implemented to take $O(\log n)$ time (see Section 5). [7]

3. The energy of $\vec{f}_0$ is sufficiently bounded, the quality of tree voltages is sufficiently bounded, and all other parts of the algorithm can be implemented efficiently (see Section 6).

4   Convergence Rate Analysis

In this section we analyze the convergence rate of `SimpleSolver`. The central result is as follows.

Theorem 4.1 (Convergence). *Each iteration i of `SimpleSolver` computes feasible $\vec{f}_i \in \mathbb{R}^E$ such that*

$$\mathbb{E}\left[\xi_r(\vec{f}_i)\right] - \xi_r(\vec{f}_{\mathrm{opt}}) \leq \left(1 - \frac{1}{\tau}\right)^i \left(\xi_r(\vec{f}_0) - \xi_r(\vec{f}_{\mathrm{opt}})\right) \ .$$

Our proof is divided into three steps. In Section 4.1 we analyze the energy gain of a single algorithm iteration, in Section 4.2 we bound the distance to optimality in a single algorithm iteration, and in Section 4.3 we connect these to prove the theorem.

4.1   Cycle Update Progress

For feasible $\vec{f} \in \mathbb{R}^E$, we can decrease $\xi_r(\vec{f})$ while maintaining feasibility, by adding a multiple of a circulation $\vec{c} \in \mathbb{R}^E$ to $\vec{f}$. In fact, we can easily optimize to pick the best multiple.

---

[7]Note that a naïve implementation of cycle updates does not necessarily run in sublinear time. In particular, updating $\vec{f}_i := \vec{f}_{i-1} - \frac{\Delta_{c_e}(\vec{f}_{i-1})}{R_e} \vec{c}_e$ by walking C_e and updating flow values one by one may take more than (even amortized) sublinear time, even though T may be of low total stretch. Since $\mathrm{st}(T)$ is defined with respect to cycle resistances but not with respect to the number of edges in these cycles, it is possible to have a low-stretch tree where each tree cycle still has $\Omega(|V|)$ edges on it. Furthermore, even if all edges have resistances 1 and therefore the average number of edges in a tree cycle is $O(\log n)$, since `SimpleSolver` samples off-tree edges with higher stretch with higher probabilities, the expected number of edges in a tree cycle may still be $\Omega(|V|)$.

Lemma 4.2 (Energy Improvement). *For $\vec{f} \in \mathbb{R}^E$, $\vec{c} \in \mathbb{R}^E$, and $\alpha^* = -\frac{\vec{f}^T \mathbf{R} \vec{c}}{\vec{c}^T \mathbf{R} \vec{c}} \in \mathbb{R}$ we have*

$$\underset{\alpha \in \mathbb{R}}{\arg\min} \ \xi_r\left(\vec{f} + \alpha \vec{c}\right) = -\frac{\vec{f}^T \mathbf{R} \vec{c}}{\vec{c}^T \mathbf{R} \vec{c}} \quad and \quad \xi_r\left(\vec{f} + \alpha^* \vec{c}\right) - \xi_r\left(\vec{f}\right) = -\frac{\left(\vec{f}^T \mathbf{R} \vec{c}\right)^2}{\vec{c}^T \mathbf{R} \vec{c}} \ .$$

Proof. By definition of energy $\xi_r\left(\vec{f} + \alpha \vec{c}\right) = (\vec{f} + \alpha \vec{c})^T \mathbf{R} (\vec{f} + \alpha \vec{c}) = \vec{f}^T \mathbf{R} \vec{f} + 2\alpha \vec{f}^T \mathbf{R} \vec{c} + \alpha^2 \vec{c}^T \mathbf{R} \vec{c}$. Setting the derivative with respect to α to 0 and substituting in $\alpha = \alpha^*$ yields the results. $\qquad \square$

In the special case where $\vec{c} = \vec{c}_e$ is a tree cycle for some off-tree edge $e \in E \setminus T$, since $R_e = \vec{c}_e^T \mathbf{R} \vec{c}_e$ and $\Delta_{c_e}(\vec{f}) = \vec{f}^T \mathbf{R} \vec{c}_e$, this procedure is precisely the iterative step of `SimpleSolver`, i.e. a cycle update. The following lemma follows immediately and states that the energy decrease of a cycle update is exactly the energy of a resistor with resistance R_e and potential drop $\Delta_{c_e}(\vec{f})$.

Lemma 4.3 (Cycle Update). *For feasible $\vec{f} \in \mathbb{R}^E$ and $e \in E \setminus T$ we have*

$$\xi_r\left(\vec{f} - \frac{\Delta_{c_e}(\vec{f})}{R_e} \vec{c}_e\right) - \xi_r\left(\vec{f}\right) = -\frac{\Delta_{c_e}(\vec{f})^2}{R_e} \ .$$

4.2 Distance to Optimality

To bound how far $\vec{f}_i$ in `SimpleSolver` is from optimality we use that the duality gap between $\vec{f}_i$ and its tree induced voltages $\vec{v}_i$ is an upper bound on this distance. Here we derive a simple expression for this quantity in terms of cycle potentials.

Lemma 4.4 (Tree Gap). *For feasible $\vec{f} \in \mathbb{R}^E$ and tree induced voltages $\vec{v} \in \mathbb{R}^V$ we have*

$$\text{gap}(\vec{f}, \vec{v}) = \sum_{e \in E \setminus T} \frac{\Delta_{c_e}(\vec{f})^2}{r_e} \ .$$

Proof. By definition of primal and dual energy we have $\text{gap}(\vec{f}, \vec{v}) = \vec{f}^T \mathbf{R} \vec{f} - (2\vec{v}^T \vec{\chi} - \vec{v}^T \mathbf{L} \vec{v})$. Using that $\mathbf{B}^T \vec{f} = \vec{\chi}$ and $\mathbf{L} = \mathbf{B}^T \mathbf{R}^{-1} \mathbf{B}$ we get

$$\text{gap}(\vec{f}, \vec{v}) = \vec{f}^T \mathbf{R} \vec{f} - 2\vec{v}^T \mathbf{B}^T \vec{f} - \vec{v}^T \mathbf{B}^T \mathbf{R}^{-1} \mathbf{B} \vec{v} = \left(\mathbf{R} \vec{f} - \mathbf{B} \vec{v}\right)^T \mathbf{R}^{-1} \left(\mathbf{R} \vec{f} - \mathbf{B} \vec{v}\right) \ .$$

Therefore $\text{gap}(\vec{f}, \vec{v}) = \sum_{e \in E} \frac{1}{r_e} \left(\vec{f}(e) r_e - \Delta_{\vec{v}}(e)\right)^2$. However, by the uniqueness of tree paths, the antisymmetry of $\vec{f} \in \mathbb{R}^E$, and the definition of tree voltages we have

$$\forall a, b \in V \ : \ \Delta_{\vec{v}}(a, b) = \vec{v}(a) - \vec{v}(b) = \sum_{e \in P_{as}} \vec{f}(e) r_e + \sum_{e \in P_{sb}} \vec{f}(e) r_e = \sum_{e \in P_{ab}} \vec{f}(e) r_e \ .$$

Therefore, $e \in T \Rightarrow \vec{f}(e) r_e - \Delta_{\vec{v}}(e) = 0$ and $e \in E \setminus T \Rightarrow \vec{f}(e) r_e - \Delta_{\vec{v}}(e) = \Delta_{c_e}(\vec{f})$. $\qquad \square$

4.3 Convergence Proof

Here we connect the energy decrease of a cycle update given by Lemma 4.3 and the duality gap formula given by Lemma 4.4 to bound the the convergence of `SimpleSolver`. Throughout this section we use $\vec{v}_i$ to denote the tree induced voltages for flow $\vec{f}_i$.

First, we show that in expectation each iteration i of `SimpleSolver` decreases $\xi_r\left(\vec{f}_{i-1}\right)$ by a $\frac{1}{\tau}$ fraction of the duality gap.

Lemma 4.5 (Expected Progress). *For iteration i of* `SimpleSolver` *we have*

$$\mathbb{E}\left[\xi_r(\vec{f}_i) - \xi_r(\vec{f}_{i-1})\Big|\mathrm{gap}(\vec{f}_{i-1}, \vec{v}_{i-1})\right] = -\frac{\mathrm{gap}(\vec{f}_{i-1}, \vec{v}_{i-1})}{\tau} \quad.$$

Proof. In each iteration i, `SimpleSolver` picks a random $e_i \in E \setminus T$ with probability p_{e_i} and adds a multiple of $\vec{c}_{e_i}$ which by Lemma 4.3 decreases the energy by $\frac{\Delta_{c_e}(\vec{f}_{i-1})^2}{R_e}$. Therefore

$$\mathbb{E}\left[\xi_r(\vec{f}_i) - \xi_r(\vec{f}_{i-1})\Big|\mathrm{gap}(\vec{f}_{i-1}, \vec{v}_{i-1})\right] = \mathbb{E}\left[\sum_{e \in E \setminus T} p_e \left(\frac{-\Delta_{c_e}(\vec{f}_{i-1})^2}{R_e}\right)\Bigg|\mathrm{gap}(\vec{f}_{i-1}, \vec{v}_{i-1})\right]$$

Using that $p_e = \frac{1}{\tau} \cdot \frac{R_e}{r_e}$ and applying Lemma 4.4 yields the result. $\qquad\square$

Next, we show that this implies that each iteration decreases the expected energy difference between the current flow and the optimal flow by a multiplicative $\left(1 - \frac{1}{\tau}\right)$.

Lemma 4.6 (Convergence Rate). *For all $i \geq 0$ let random variable $D_i \stackrel{\mathrm{def}}{=} \xi_r(\vec{f}_i) - \xi_r(\vec{f}_{\mathrm{opt}})$. Then for all iterations $i \geq 1$ we have* $\mathbb{E}[D_i] \leq \left(1 - \frac{1}{\tau}\right)\mathbb{E}[D_{i-1}]$.

Proof. Since in each iteration of `SimpleSolver` one of a finite number of edges is chosen, clearly D_i is a discrete random variable and by law of total expectation we have

$$\mathbb{E}[D_i] = \sum_c \mathbb{E}[D_i|D_{i-1} = c]\Pr[D_{i-1} = c]$$

However, we know $D_{i-1} \leq \mathrm{gap}(\vec{f}_{i-1}, \vec{v}_{i-1})$ so by Lemma 4.5, $\mathbb{E}[D_i|D_{i-1} = c] \leq c - \frac{c}{\tau}$. Therefore:

$$\mathbb{E}[D_i] \leq \sum_c \left[\left(1 - \frac{1}{\tau}\right)c \cdot \Pr[D_{i-1} = c]\right] = \left(1 - \frac{1}{\tau}\right)\mathbb{E}[D_{i-1}] \quad.$$

$\qquad\square$

Finally, by induction on Lemma 4.6 and the definition of D_i we prove Theorem 4.1.

5 Cycle Update Data Structure

In this section we show how to implement each iteration (i.e. cycle update) of `SimpleSolver` in $O(\log n)$ time. In Section 5.1 we formulate this as a data structure problem, in Section 5.2 we present a recursive solution, and in Section 5.3 we provide a linear algebraic interpretation that may be useful in both theory and practice.

5.1 The Data Structure Problem

In each iteration i of `SimpleSolver` we pick a random $(a, b) \in E \setminus T$ and for feasible $\vec{f} \in \mathbb{R}^E$ compute

$$\alpha^* = \frac{\Delta_{c_{(a,b)}}(\vec{f})}{R_{(a,b)}} = \frac{\vec{f}(a,b)r_{a,b} - (\vec{v}(a) - \vec{v}(b))}{R_{(a,b)}} \quad \text{where} \quad \vec{v}(a) = \sum_{e \in P_{(a,s)}} \vec{f}(e)r_e$$

and s is an arbitrary fixed vertex in V which we refer to as the *root*. Then α^* is added to the flow on every edge in the tree cycle $C_{(a,b)}$. By the antisymmetry of flows and the uniqueness of tree paths, this update is equivalent to (1) adding α^* to the flow on edge (a, b), (2) adding $-\alpha^*$ to the flow on every edge in $P_{(s,b)}$ and then (3) adding α^* to the flow on every edge in $P_{(s,a)}$.

Therefore, to implement cycle updates it suffices to store the flow on off-tree edges in an array and implement a data structure that supports the following operations.

10

- `init(T, s ∈ V)`: initialize a data structure D given tree T and root $s \in V$.
- `query`$_D(a \in V)$: return $\vec{v}(a) = \sum_{e \in P_{(s,a)}} \vec{f}(e) r_e$.
- `update`$_D(a \in V, \alpha \in \mathbb{R})$: set $\vec{f}(e) := \vec{f}(e) + \alpha$ for all $e \in P_{(s,a)}$.

5.2 Recursive Solution

While one could solve this data structure problem with a slight modification of link-cut trees [ST83], that dynamic data structure is overqualified for our purpose. In contrast to the problems for which link-cut trees were originally designed, in our setting the tree is static, so that much of the sophistication of link-cut trees may be unnecessary. Here we develop a very simple separator decomposition tree based structure that provides worst-case $O(\log n)$ operations that we believe sheds more light on the electric flow problem and may be useful in practice.

Our solution is based on the well known fact that every tree has a good vertex separator, tracing back to Jordan in 1869 [Jor69].

Lemma 5.1 (Tree Vertex Separator). *For a spanning tree T rooted at s with $n \geq 2$ vertices, in $O(n)$ time we can compute*

$$(d, T_0, \ldots, T_k) = \texttt{tree-decompose}(T) \ ,$$

such that the removal of $d \in V$ (which might equal to s) disconnects T into subtrees $T_0, \ldots, T_k$, where T_0 is rooted at s and contains d as a leaf, while other T_i's are rooted at d. Furthermore, each T_i has at most $n/2 + 1$ vertices.
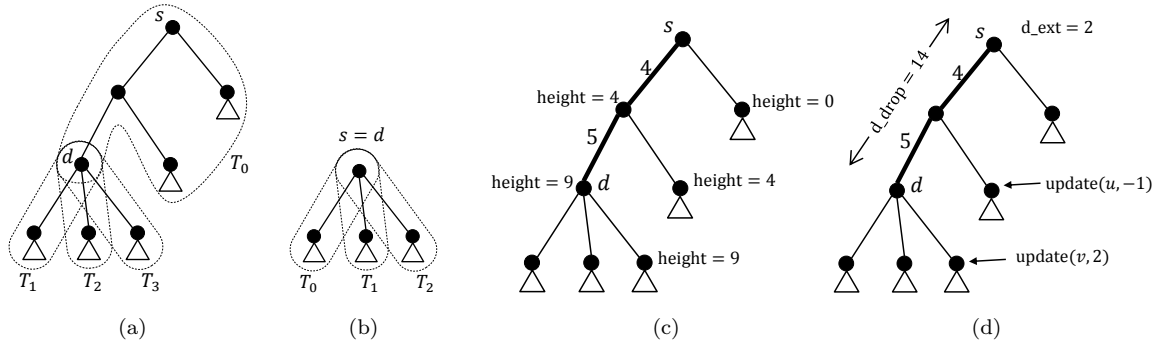

Figure 1: (a) and (b) are examples of tree decompositions when $s \neq d$ and $s = d$, and (c) and (d) are examples on computing `height`, `d_ext` and `d_drop`.

Proof. We start at root s, and at each step follow an edge to a child whose corresponding subtree is the largest among all children. We continue this procedure until at some vertex d (which might be s itself) the sizes of all its subtrees have no more than $\frac{n}{2}$ vertices. This vertex d is the desired vertex separator. Now if $s \neq d$ we let T_0 be the subtree above d rooted at s with d being the leaf, and let each T_i be a sub-tree below d and rooted at d, see Figure 1a; or when $s = d$ we let $T_0, \ldots, T_k$ each denote a subtree below d and rooted at d, see Figure 1b. [8] By pre-computing subtree sizes `tree-decompose`(T) runs in $O(n)$ time. □

Applying this lemma recursively induces a separator decomposition tree from which we build our data structure. To execute `init`(T, s) we compute a vertex separator $(d, T_0, \ldots, T_k)$ and recurse separately on each subtree until we are left with a tree of only two vertices. In addition, we precompute the total weight

---

[8]For technical purposes, when V has only two vertices with s being the root, we always define $d \in V$ such that $d \neq s$ and define `tree-decompose`(T) to return no sub-trees, i.e., $k = -1$. By orienting the tree away from s and precomputing the number of descendents of each node we can easily implement this in $O(n)$ time.

of the intersection of the root to d path $P_{(s,d)}$ and the root to a path $P_{(s,a)}$, denoted by `height`(a) for every $a \in V$, by taking a walk over the tree. See Figure 1c and Algorithm 2 for details.

With this decomposition we can support `query` and `update` by maintaining just 2 values about the paths $P_{(s,d)}$. For each subtree we maintain the following variables:

- `d_drop`, the total potential drop induced on the the path $P_{(s,d)}$, and
- `d_ext`, the total amount of flow that has been updated using the entire $P_{(s,d)}$ path, i.e. the contribution to $P_{(s,d)}$ from vertices beyond d.

It is easy to see that these invariants can be maintained recursively and that they suffice to answer queries and updates efficiently (see Algorithm 3 and Algorithm 4 for the implementations and Figure 1d for an illustration). Furthermore, because the sizes of trees at least half at each recursion, `init` takes $O(n \log n)$ while `query` and `update` each takes $O(\log n)$ time.

Algorithm 2: Recursive `init`$(T, s \in V)$

`d_ext` $:= 0$, `d_drop` $:= 0$;
$(d, T_0, \ldots, T_k) :=$ `tree-decompose`(T);
$\forall a \in V : $ `height`$(a) := \sum_{e \in P_{(s,a)} \cap P_{(s,d)}} r_e$;
if $|V| > 2$ **then**
$\quad \mid \quad \forall i \in \{0, 1, \ldots, k\} : D_i :=$ `init`(T_i, d);
end

Algorithm 3: Recursive `query`$(a \in V)$

---

if $a = d$ **then return** `d_drop` ;
else if $|V| = 2$ **then return** 0;
else if $a \in T_0$ **then**
$\quad \mid \quad$ **return** `d_ext` $\cdot$ `height`$(a) +$ `query`$_{D_0}(a)$;
else
$\quad \mid \quad$ let T_i be unique tree containing a;
$\quad \mid \quad$ **return** `d_drop` $+$ `query`$_{D_i}(a)$;
end

Algorithm 4: Recursive `update`$(a \in V, \alpha \in \mathbb{R})$

`d_drop` $:=$ `d_drop` $+ \alpha \cdot$ `height`(a) ;
if $|V| = 2$ **then return**;
if $a \notin T_0$ **then** `d_ext` $:=$ `d_ext` $+ \alpha$;
if $a \neq d$ **then**
$\quad \mid \quad$ let T_i be unique tree containing a;
$\quad \mid \quad$ `update`$_{D_i}(a, \alpha)$;
end

5.3   Linear Algebra Solution

Here we unpack the recursion in the previous section to provide a linear algebraic view of our data structure that may be useful in both theory and practice. We show that the `init` procedure in the previous section computes for all $a \in V$ a query vector $\vec{q}(a)$ and an update vector $\vec{u}(a)$ each of support size $O(\log |V|)$, such that the entire state of the data structure is an $O(|V|)$ dimensional vector $\vec{x}$ initialized to $\vec{0}$ allowing `query` and `update` to be as simple as the following,

$$\begin{aligned} \texttt{query}(a) &: \quad \text{return } \vec{q}(a) \cdot \vec{x} \\ \texttt{update}(a, \alpha) &: \quad \vec{x} := \vec{x} + \alpha \vec{u}(a) \ . \end{aligned} \tag{3}$$

To see this, first note that `init` recursively creates a total of $N = O(|V|)$ subtrees $T_1, \ldots, T_N$. Letting `d_ext`$_i$, `d_drop`$_i$, `height`$_i$ denote `d_ext`, `d_drop`, and `height` associated with tree T_i, the state of the data structure is completely represented by a vector $\vec{x} \in \mathbb{R}^{\{e,d\} \times N}$ defined as follows

$$\vec{x}_{c,i} \stackrel{\text{def}}{=} \begin{cases} \texttt{d\_ext}_i & \text{if } c = e; \\ \texttt{d\_drop}_i & \text{if } c = d. \end{cases}$$

12

It is easy to see that given vertex $a \in V$, in order for $\mathtt{query}(a)$ or $\mathtt{update}(a, \alpha)$ to affect tree T_i in the decomposition, it is necessary that a is a vertex in T_i and a is not the root s_i of T_i. Accordingly, we let $T(a) \overset{\text{def}}{=} \{i \in [N] \mid a \in T_i \text{ and } a \neq s_i\}$ be the indices of such trees that a may affect.

Next, in order to fully specify the query vector $\vec{q}(a)$ and the update vector $\vec{u}(a)$, we further refine $T(a)$ by defining $T_0(a) \overset{\text{def}}{=} \{i \in T(a) \mid a \text{ is in } T_0 \text{ of } \mathtt{tree\text{-}decompose}(T_i)\}$ and $T_+(a) \overset{\text{def}}{=} T(a) \setminus T_0(a)$. Then, one can carefully check that the following definitions of $\vec{q}(a), \vec{u}(a) \in \mathbb{R}^{\{e,d\} \times N}$ along with their $\mathtt{query}$ and $\mathtt{update}$ in (3), exactly coincide with our recursive definitions in Algorithm 3 and Algorithm 4 respectively.

$$\vec{q}(a)_{(c,i)} \overset{\text{def}}{=} \begin{cases} \mathtt{height}_i(a) & c = e \text{ and } i \in T_0(a) \\ 1 & c = d \text{ and } i \in T_+(a) \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad \vec{u}(a)_{(c,i)} \overset{\text{def}}{=} \begin{cases} 1 & c = e \text{ and } i \in T_+(a) \\ \mathtt{height}_i(a) & c = d \text{ and } i \in T(a) \\ 0 & \text{otherwise} \end{cases}$$

Furthermore, the properties of $\mathtt{init}$ and $\mathtt{tree\text{-}decompose}$ immediately implies that each vertex a can appear in at most $O(\log n)$ trees where it is not the root, so $|T(a)| = O(\log n)$. As a consequence, $\vec{q}(a)$ and $\vec{u}(a)$ each has at most $O(\log n)$ non-zero entries. It is also not hard to see that we can pre-compute these vectors in $O(n \log n)$ time using our recursive $\mathtt{init}$.

Applying these insights to the definition of a cycle update we see that this data structure allows each cycle update of $\mathtt{SimpleSolver}$ to be implemented as a single dot product and a single vector addition where one vector in each operation has $O(\log n)$ non-zero entries. Since these vectors can be precomputed each cycle update can be implemented by iterating over a fixed array of $O(\log n)$ pointers and performing simple arithmetic operations. We hope this may be fast in practice.

6 Simple Algorithm Running-Time Analysis

Here we give the remaining analysis needed to prove the correctness of $\mathtt{SimpleSolver}$ and prove Theorem 3.2. First we bound the energy of $\vec{f}_0$.

Lemma 6.1 (Initial Energy). [9] $\xi_r(\vec{f}_0) \leq \mathrm{st}(T) \cdot \xi_r(\vec{f}_{\mathrm{opt}})$.

Proof. Recall that $\vec{f}_0 \in \mathbb{R}^E$ is the *unique* vector that meets the demands and is nonzero only on T. Now, if for every $e \in E$ we sent $\vec{f}_{\mathrm{opt}}(e)$ units of flow along P_e we achieve a vector with the same properties. Therefore, $\vec{f}_0 = \sum_{e \in E} \vec{f}_{\mathrm{opt}}(e) \vec{p}_e$ and by applying the Cauchy-Schwarz inequality we get

$$\xi_r(\vec{f}_0) = \sum_{e \in T} r_e \left(\sum_{e' \in E | e \in P_{e'}} \vec{f}_{\mathrm{opt}}(e') \right)^2 \leq \sum_{e \in T} \left[\left(\sum_{e' \in E | e \in P_{e'}} \frac{r_e}{r_{e'}} \right) \left(\sum_{e' \in E | e \in P_{e'}} r_{e'} \vec{f}_{\mathrm{opt}}(e')^2 \right) \right] .$$

Applying the crude bound that $\sum_{e' \in E | e \in P_{e'}} r_{e'} \vec{f}_{\mathrm{opt}}(e')^2 \leq \xi_r(\vec{f}_{\mathrm{opt}})$ for all edges $e \in E$ and noting that by the definition of stretch $\sum_{e \in T} \sum_{e' \in E | e \in T_{e'}} \frac{r_e}{r_{e'}} = \sum_{e' \in E} \sum_{e \in T_e} \frac{r_e}{r_{e'}} = \mathrm{st}(T)$ the result follows immediately. $\square$

Next, we show how approximate optimality is preserved within polynomial factors when rounding from an ε-approximate electric flow to tree induced voltages.

Lemma 6.2 (Tree Voltage Rounding). *Let $\vec{f} \in \mathbb{R}^E$ be a primal feasible flow with $\xi_r(\vec{f}) \leq (1 + \varepsilon) \cdot \xi_r(\vec{f}_{\mathrm{opt}})$ for $\varepsilon > 0$ and let $\vec{v} \in \mathbb{R}^V$ be the tree induced voltages. Then the following holds*

$$\left\| \vec{v} - \mathbf{L}^\dagger \vec{\chi} \right\|_{\mathbf{L}} \leq \sqrt{\varepsilon \cdot \tau} \left\| \mathbf{L}^\dagger \vec{\chi} \right\|_{\mathbf{L}} .$$

---

[9]This lemma follows immediately from the well known fact that T is a $\mathrm{st}(T)$-spectral sparsifier of G, cf. [ST12, Lemma 9.2], but a direct proof is included here for completeness.

Proof. By Lemma 4.5 we know that one random cycle update from `SimpleSolver` is expected to decrease $\xi_r(\vec{f})$ by $\mathrm{gap}(\vec{f},\vec{v})/\tau$. Therefore by the optimality of $\vec{f}_{\mathrm{opt}}$ we have

$$\xi_r(\vec{f}) - \frac{\mathrm{gap}(\vec{f},\vec{v})}{\tau} \geq \xi_r(\vec{f}_{\mathrm{opt}})$$

and since $\xi_r(\vec{f}) \leq (1+\varepsilon) \cdot \xi_r(\vec{f}_{\mathrm{opt}})$ and $\xi_r(\vec{f}) \leq \xi_r(\vec{f}_{\mathrm{opt}})$ we have $\mathrm{gap}(\vec{f}_{\mathrm{opt}}, \vec{v}) \leq \varepsilon \cdot \tau \cdot \xi_r(\vec{f}_{\mathrm{opt}})$. Finally, using that $\vec{v}_{\mathrm{opt}} = \mathbf{L}^\dagger \vec{\chi}$, $\vec{f}_{\mathrm{opt}} = \mathbf{R}^{-1}\mathbf{B}\vec{v}_{\mathrm{opt}}$, and $\mathbf{L} = \mathbf{B}^T \mathbf{R}^{-1}\mathbf{B}$ it is straightforward to check

$$\mathrm{gap}(\vec{f}_{\mathrm{opt}}, \vec{v}) = \left\| \vec{v} - \mathbf{L}^\dagger \vec{\chi} \right\|_{\mathbf{L}}^2 \quad \text{and} \quad \xi_r(\vec{f}_{\mathrm{opt}}) = \left\| \mathbf{L}^\dagger \vec{\chi} \right\|_{\mathbf{L}}^2 \ .$$

$\square$

Finally, we have everything we need to prove the correctness of `SimpleSolver`.

Proof of Theorem 3.2. Applying Theorem 4.1, the definition of K, and Lemma 6.1 we get

$$\mathbb{E}\left[\xi_r(\vec{f}_K)\right] - \xi_r(\vec{f}_{\mathrm{opt}}) \leq \left(1 - \frac{1}{\tau}\right)^{\tau \log(\varepsilon^{-1}\mathrm{st}(T)\tau)} \left(\mathrm{st}\,(T)\,\xi_r(\vec{f}_{\mathrm{opt}}) - \xi_r(\vec{f}_{\mathrm{opt}})\right) \ .$$

Using several crude bounds and applying Lemma 6.2 we get

$$\mathbb{E}\left[\xi_r(\vec{f}_K)\right] \leq \left(1 + \frac{\varepsilon}{\tau}\right)\xi_r(\vec{f}_{\mathrm{opt}}) \quad \text{and} \quad \mathbb{E}\left\|\vec{v}_K - \mathbf{L}^\dagger \vec{\chi}\right\|_{\mathbf{L}} \leq \sqrt{\varepsilon} \cdot \left\|\mathbf{L}^\dagger \vec{\chi}\right\|_{\mathbf{L}}$$

which is stronger than what we need to prove for the theorem.

All that remains is to bound the running time. To do this, we implement `SimpleSolver` using the latest low-stretch spanning tree construction (Theorem 2.11) and construct a spanning tree T of stretch $\mathrm{st}(T) = O(m \log n \log\log n)$ in time $O(m \log n \log\log n)$ yielding $\tau = O(m \log n \log\log n)$.

Next, to compute $\vec{f}_0$ we note that given any demand vector $\vec{\chi} \in \mathbb{R}^V$ with $\sum_i \vec{\chi}_i = 0$, the quantity $\vec{f}_0(e)$ on a tree edge $e \in T$ is uniquely determined by the summation of $\vec{\chi}_v$ where v is over the vertices on one side of e. Therefore, $\vec{f}_0$ can be computed via a DFS in $O(n)$ time.

To compute R_e for each off-tree edge $e \in E \setminus T$ we could either use Tarjan's off-line LCA algorithm [Tar79] that runs in a total of $O(m)$ time, or simply use our own data structure in $O(m \log n)$ time. In fact, one can initiate a different instance of our data structure on T, and for each off-tree edge $(a,b) \in E \setminus T$, one can call `update`$(b,1)$ and `update`$(a,-1)$, so that $R_e = $ `query`$(b) - $ `query`$(a) + r_e$.

Finally, we can initialize our data structure in $O(n \log n)$ time, set the initial flow values in $O(n \log n)$ time, perform each cycle update in $O(\log n)$ time, and compute all the tree voltages in $O(n \log n)$ time using the work in section Section 5. Since the total number of iterations of our algorithm is easily seen to be $O(m \log n \log(n\varepsilon^{-1}) \log\log n)$ we get the desired total running time. $\square$

7 Improved Running Time

In this section we show how to improve our algorithm's running time to $O\left(m \log^2 n \log\log n \log(\varepsilon^{-1})\right)$ (i.e. improve our algorithm's dependence on ε from $O(\log(\varepsilon^{-1}n))$ to $O(\log \varepsilon^{-1})$). The improvement is gained by careful application of two techniques to the standard `SimpleSolver`: (1) by changing the resistances of the edges in the T and applying `SimpleSolver`, we can quickly compute an initial feasible $\vec{f}_0 \in \mathbb{R}^E$ of better quality, and (2) by changing the stopping criterion for cycle updates, we can achieve a stronger guarantee on the quality of flow induced tree voltages.

In Section 7.1, we introduce these concepts by presenting a straightforward two-step algorithm that improves the error dependence to $\log(\varepsilon^{-1} \log n)$. In Section 7.2, we present the full algorithm that by carefully applying these techniques recursively $O(\log^* n)$ times obtains an improved error dependence of $O(\log \varepsilon^{-1})$ and the best asymptotic running time proven in this paper.

Algorithm 5: ExampleSolver

Input : $G = (V, E, r)$, $\vec{\chi} \in \mathbb{R}^V$, $\varepsilon \in \mathbb{R}^+$
Output: $\vec{f}$ and $\vec{v}$

1 $T :=$ low-stretch spanning tree of G;
2 $(\vec{f_0}, \star) := \texttt{SimpleSolver}(G - T + \frac{T}{\log n}, \vec{\chi}, 1)$;

   (the simple solver will use $T' := \frac{T}{\log n}$ as the spanning tree)

3 $\vec{p_e} := \frac{1}{\tau(T)} \cdot \frac{R_e}{r_e}$ for all $e \in E \setminus T$;
4 $K := \left\lceil \tau(T) \log\left(\frac{2 \log n}{\varepsilon} \right) \right\rceil$;
5 $K' :=$ an integer chosen uniformly at random in $\{0, 1, \ldots, \lceil \tau(T) \rceil - 1\}$;
6 for $i = 1$ *to* $K + K'$ **do**
7 $\quad$ Pick random $e_i \in E \setminus T$ by probability distribution $\vec{p}$;
8 $\quad$ $\vec{f_i} = \vec{f_{i-1}} - \frac{\Delta_{c_e}(\vec{f_{i-1}})}{R_e} \vec{c_e}$;
9 end
10 return $\vec{f}_{K+K'}$ *and its tree induced voltages* $\vec{v}_{K+K'}$

7.1 An Example Algorithm with $O\big(m \log^2 n \log \log n \log(\varepsilon^{-1} \log n)\big)$ Running Time

Here we present an ExampleSolver (see Algorithm 5), an algorithm that illustrates the techniques we will use in Section 7.2 to obtain the fastest running times presented in this paper. This algorithm modifies the resistances of tree edges and applies SimpleSolver once to compute feasible $\vec{f_0}$ that is $O(\log n)$ factor from optimal (rather than an $\tilde{O}(m)$ factor as in Lemma 6.1) and then uses this $\vec{f_0}$ as a warm start to perform cycle updates. By also randomizing over the number of cycle updates to perform, we can show that we get an improved running time as stated in the following theorem.

Theorem 7.1 (ExampleSolver)**.** *The output of ExampleSolver satisfies*

$$\mathbb{E}\big[\xi_r(\vec{f})\big] \leq (1 + \varepsilon) \cdot \xi_r\big(\vec{f}_{\text{opt}}\big) \quad and \quad \mathbb{E}\big\|\vec{v} - \mathbf{L}^\dagger \vec{\chi}\big\|_{\mathbf{L}} \leq \sqrt{\varepsilon} \cdot \big\|\mathbf{L}^\dagger \vec{\chi}\big\|_{\mathbf{L}}$$

and ExampleSolver can be implemented to run in time $O\big(m \log^2 n \log \log n \log(\varepsilon^{-1} \log n)\big)$.

We prove the above theorem in two steps, introducing one new technique for each step.

Step #1: Tree Scaling. Given graph G, recall that by Theorem 2.11 one can obtain a spanning tree T with condition number $\tau(T) = O(m \log n \log \log n)$. For $\kappa \geq 1$ let T/κ denote T where every $e \in T$ has its resistance decreased by a multiplicative κ. Now replace T with T/κ and denote this new graph G', the new resistances by r', and the new tree by T'. This technique is similar to one used in [KMP10, KMP11], and one can easily check that

$$\tau(T') = \tau(T)/\kappa + (m - n + 1) < \tau(T)/\kappa + m ,$$

i.e., decreasing the resistances of T improves the tree condition number. Furthermore, since replacing T by T/κ only changes resistances, $\vec{f} \in \mathbb{R}^E$ is feasible in G if and only if it is feasible in G', and therefore one could compute an approximate flow in G' and then use the results in G. The following lemma bounds the loss in approximation of such an operation.

Lemma 7.2 (Tree Scaling)**.** *Let G' be graph G with T replaced by T/κ for $\kappa \geq 1$, let $\vec{f'}$ be an ε-approximate electrical flow in G', and let $\vec{f}_{\text{opt}}$ be the optimal electrical flow in G. Then,*

$$\xi_r\big(\vec{f'}\big) \leq \kappa(1 + \varepsilon) \xi_r\big(\vec{f}_{\text{opt}}\big) .$$

Proof. Since $\vec{f'}$ is feasible in G' it is feasible in G. Letting $\vec{f'}_{\text{opt}}$ be the optimal electrical flow in G' we can then bound $\xi_r(\vec{f'})$ as follows

$$\xi_r(\vec{f'}) \leq \kappa \cdot \xi_{r'}(\vec{f'}) \qquad\qquad\qquad \text{(Since } r_e \leq \kappa \cdot r'_e \text{ for any edge } e)$$
$$\leq \kappa(1+\varepsilon) \cdot \xi_{r'}(\vec{f'}_{\text{opt}}) \qquad\qquad\qquad \text{(Since } \vec{f'} \text{ is } \varepsilon\text{-approximate in } G')$$
$$\leq \kappa(1+\varepsilon) \cdot \xi_{r'}(\vec{f}_{\text{opt}}) \qquad\qquad\qquad \text{(By optimality of } \vec{f'}_{\text{opt}} \text{ in } G')$$
$$\leq \kappa(1+\varepsilon) \cdot \xi_r(\vec{f}_{\text{opt}}) \;. \qquad\qquad\qquad \text{(Since } r'_e \leq r_e \text{ for any edge } e)$$

$\square$

We can apply Lemma 7.2 directly to the analysis of `ExampleSolver`. Choosing $\kappa = \log n$ in line 2, we see that `ExampleSolver` first computes feasible $\vec{f}_0 \in \mathbb{R}^E$ that is 1-approximate electric flow in G'. By having the recursive `SimpleSolver` invocation in this line use spanning tree T' (which has $\tau(T') \leq \tau(T)/\kappa + m$), this step takes $O\left(\left(\frac{\tau(T)}{\log n} + m\right) \log n \log n\right) = O(m \log^2 n \log\log n)$ time and by Lemma 7.2 guarantees that

$$\mathbb{E}\left[\xi_r(\vec{f}_0)\right] = (1 + \log n) \cdot \xi_r(\vec{f}_{\text{opt}}) \leq 2\log n \cdot \xi_r(\vec{f}_{\text{opt}}) \;.$$

This is an improvement compared to what was previously guaranteed by Lemma 6.1 and lets us apply Theorem 4.1 with $i = K = \left\lceil \tau(T) \log\left(\frac{2\log n}{\varepsilon}\right) \right\rceil$ to conclude that:

$$\mathbb{E}\left[\xi_r(\vec{f}_K)\right] \leq (1+\varepsilon)\xi_r(\vec{f}_{\text{opt}}) \;. \tag{4}$$

Step #2: Randomized Stopping Time. While (4) provides an (expected) guarantee on the primal energy for $\vec{f}_K$, converting this to a dual guarantee on its tree induced voltage vector $\vec{v}_K$ using Lemma 6.2 causes us to lose a factor of $\tau = \tilde{O}(m)$ on the dependency of ε, which translates into an extra factor of $O(\log m)$ in the running time. This motivates us to study a more efficient conversion as follows.

Lemma 7.3. *For feasible $\vec{f}_K \in \mathbb{R}^E$ satisfying $\xi_r(\vec{f}_K) \leq (1+\varepsilon)\xi_r(\vec{f}_{\text{opt}})$, if one applies K' cycle updates, where K' is an integer chosen uniformly at random from $\{0, 1, \ldots, \lceil\tau\rceil - 1\}$, then we have*

$$\mathbb{E}\left[\text{gap}(\vec{f}_{K+K'}, \vec{v}_{K+K'})\right] \leq \varepsilon \cdot \xi_r(\vec{f}_{\text{opt}}) \;.$$

Proof. Following the notation of Lemma 4.6, let random variable $D_i \stackrel{\text{def}}{=} \xi_r(\vec{f}_i) - \xi_r(\vec{f}_{\text{opt}})$ for $i \in \{K, K+1, \ldots, K + \lceil\tau\rceil\}$. By Lemma 4.5 we have

$$\forall i = K+1, \ldots, K + \lceil\tau\rceil, \quad \mathbb{E}[D_i - D_{i-1}] = -\frac{1}{\tau}\mathbb{E}[\text{gap}(\vec{f}_{i-1}, \vec{v}_{i-1})] \;.$$

Now, by assumption $D_K \leq \varepsilon \cdot \xi_r(\vec{f}_{\text{opt}})$ and since $D_i \geq 0$ we can also sum over the K' yielding

$$\varepsilon \cdot \xi_r(\vec{f}_{\text{opt}}) \geq \mathbb{E}[D_K] \geq \mathbb{E}[D_K - D_{K+\lceil\tau\rceil}] \geq \frac{1}{\tau}\mathbb{E}\left[\sum_{K'=K}^{K+\lceil\tau\rceil-1} \text{gap}(\vec{f}_{K'}, \vec{v}_{K'})\right] \;.$$

Therefore, if one picks $K' \in \{0, 1, \ldots, \lceil\tau\rceil - 1\}$ at random, the desired inequality is obtained. $\square$

Now in `ExampleSolver`, our output is $\vec{f} = \vec{f}_{K+K'}$ and $\vec{v} = \vec{v}_{K+K'}$ so combining (4) and Lemma 7.3, we immediately have that $\mathbb{E}\left[\text{gap}(\vec{f}, \vec{v})\right] \leq \varepsilon \cdot \xi_r(\vec{f}_{\text{opt}})$. This simultaneously implies the following

$$\mathbb{E}[\xi_r(\vec{f}) - \xi_r(\vec{f}_{\text{opt}})] = \mathbb{E}\left[\text{gap}(\vec{f}, \vec{v}_{\text{opt}})\right] \leq \mathbb{E}\left[\text{gap}(\vec{f}, \vec{v})\right] \leq \varepsilon \cdot \xi_r(\vec{f}_{\text{opt}}) \;, \text{ and}$$
$$\mathbb{E}\left[\|\vec{v} - \mathbf{L}^\dagger \vec{\chi}\|_\mathbf{L}\right]^2 \leq \mathbb{E}\left[\|\vec{v} - \mathbf{L}^\dagger \vec{\chi}\|_\mathbf{L}^2\right] = \mathbb{E}\left[\text{gap}(\vec{f}_{\text{opt}}, \vec{v})\right] \leq \mathbb{E}\left[\text{gap}(\vec{f}, \vec{v})\right] \leq \varepsilon \cdot \xi_r(\vec{f}_{\text{opt}}) = \varepsilon\|\mathbf{L}^\dagger \vec{\chi}\|_\mathbf{L}^2 \;.$$

Here the second line has used the fact that $\text{gap}(\vec{f}_{\text{opt}}, \vec{v}) = \|\vec{v} - \mathbf{L}^\dagger \vec{\chi}\|_\mathbf{L}^2$ and $\xi_r(\vec{f}_{\text{opt}}) = \|\mathbf{L}^\dagger \vec{\chi}\|_\mathbf{L}^2$ whose proof can be found in the proof of Lemma 6.2. This finishes the proof of Theorem 7.1.

7.2 The Full Algorithm With $O\big(m\log^2 n\log\log n\log(\varepsilon^{-1})\big)$ Running Time

Here we present `FullSolver` (see Algorithm 6), an algorithm that achieves the best asymptotic running time presented this paper by recursively applying the tree scaling technique of the previous section $\log^*(n)$ number of times with careful chosen scaling parameters.

Theorem 7.4 (`FullSolver`). *The output of* `FullSolver` *satisfies*

$$\mathbb{E}\big[\xi_r(\vec{f})\big] \leq (1+\varepsilon)\cdot \xi_r(\vec{f}_{\mathrm{opt}}) \quad and \quad \mathbb{E}\big\|\vec{v} - \mathbf{L}^\dagger \vec{\chi}\big\|_{\mathbf{L}} \leq \sqrt{\varepsilon}\cdot \big\|\mathbf{L}^\dagger \vec{\chi}\big\|_{\mathbf{L}}$$

and `FullSolver` *can be implemented to run in time* $O\big(m\log^2 n\log\log n\log(\varepsilon^{-1})\big)$.

Algorithm 6: `FullSolver`

Input : $G = (V, E, r)$, $\vec{\chi} \in \mathbb{R}^V$, $\varepsilon \in \mathbb{R}^+$
Output: $\vec{f}$ and $\vec{v}$

1 $T :=$ low-stretch spanning tree of G;
2 $\vec{f}^{(0)} :=$ unique flow on T such that $\mathbf{B}^T \vec{f}_0 = \vec{\chi}$;
3 Define $\kappa^{(i)}$ and $K^{(i)}$ according to (5) and (6);
4 **for** $i = 1$ *to* $c+1$ **do**
5 $\quad \vec{f}_0 := \vec{f}^{(i-1)}$;
6 $\quad G^{(i)} := G - T + T^{(i)}$ where $T^{(i)} := \frac{T}{\prod_{j=i}^c \kappa_j}$;
$\qquad\qquad$ *(the symbols $R_e, r_e, \Delta_{c_e}(\cdot)$ below are with respect to this new graph $G^{(i)}$ and tree $T^{(i)}$)*
7 $\quad \vec{p}_e := \frac{1}{\tau(T^{(i)})} \cdot \frac{R_e}{r_e}$ for all $e \in E \setminus T$;
8 $\quad$ **for** $j = 1$ *to* $K^{(i)}$ **do**
9 $\quad\quad$ Pick random $e_i \in E \setminus T$ by probability distribution $\vec{p}$;
10 $\quad\quad \vec{f}_i = \vec{f}_{i-1} - \frac{\Delta_{c_e}(\vec{f}_{i-1})}{R_e}\vec{c}_e$;
11 $\quad$ **end**
12 $\quad \vec{f}^{(i)} := \vec{f}_{K^{(i)}}$;
13 **end**
14 **return** $\vec{f}^{(c+1)}$ *and its tree induced voltages* $\vec{v}^{(c+1)}$

We begin our proof of the above theorem by first making the parameters in `FullSolver` explicit. Again, T is a low-stretch spanning tree of G with $\tau(T) = O(m\log n\log\log n)$. Let us define a sequence $(\kappa_1, \ldots, \kappa_c)$ where

$$\kappa_1 = \log n, \quad \kappa_i = \log \kappa_{i-1}, \quad \text{and } c \text{ is chosen so that } \kappa_c \leq 2. \tag{5}$$

For each $i \in [c+1]$, we define $T^{(i)}$ to be the same spanning tree T but with resistances on each edge decreased by a factor of $\prod_{j=i}^c \kappa_j$, and let $G^{(i)}$ be the graph G with its spanning tree T replaced by $T^{(i)}$. Notice that $T^{(c+1)} = T$ and $G^{(c+1)} = G$. By a similar observation as in Section 7.1, we have that $\tau(T^{(i)}) \leq \frac{\tau(T)}{\prod_{j=i}^c \kappa_j} + m$ due to the scaling of the tree edges.

We now choose the number of iterations

$$K^{(i)} = \begin{cases} \left\lceil \big(\frac{\tau(T)}{\prod_{j=1}^c \kappa_j} + m\big) \log(\mathrm{st}(T^{(1)}))\right\rceil, & i = 1; \\[2mm] \left\lceil \big(\frac{\tau(T)}{\prod_{j=i}^c \kappa_j} + m\big) \log(2\kappa_{i-1}-1)\right\rceil, & i = 2, 3, \ldots, c; \\[2mm] \left\lceil \big(\tau(T) + m\big) \log\big(\frac{2\kappa_c-1}{\varepsilon}\big)\right\rceil + K', & i = c+1. \end{cases} \tag{6}$$

where K' is chosen from $\{0, 1, \ldots, \lceil\tau\rceil - 1\}$ uniformly at random just like that in Section 7.1. Those choices induce the following property on flow vectors $\vec{f}^{(i)}$ constructed in `FullSolver`:

Claim 7.5. *For each $i \in [c]$, $\vec{f}^{(i)}$ is (in expectation) a 1-approximate electrical flow for $G^{(i)}$ with demand vector $\vec{\chi}$, and also (in expectation) a $(2\kappa_i - 1)$-approximate electrical flow for $G^{(i+1)}$.*

Proof. Let $r^{(i)} \in \mathbb{R}^E$ denote the resistance vector in graph $G^{(i)}$, and let $\vec{f}^{(i)}_{\text{opt}} \in \mathbb{R}^E$ denote the electrical flow satisfying $\vec{\chi} \in \mathbb{R}^V$ in graph $G^{(i)}$. We prove the claim by induction on i.

In the base case ($i = 1$), since $\vec{f}^{(0)}$ is the unique flow on the tree meeting the demands Lemma 6.1 implies that $\xi_{r^{(1)}}(\vec{f}^{(0)}) \leq \text{st}(T^{(1)})\xi_{r^{(1)}}(\vec{f}^{(1)}_{\text{opt}})$. By our choice of $K^{(1)} \geq \tau(T^{(1)}) \log(\text{st}(T^{(1)}))$ and Theorem 4.1, we then see that $\mathbb{E}[\xi_{r^{(1)}}(\vec{f}^{(1)})] \leq (1+1)\xi_{r^{(1)}}(\vec{f}^{(1)}_{\text{opt}})$ and therefore the computed $\vec{f}^{(1)}$ is in expectation 1-approximate for $G^{(1)}$. Furthermore, since the resistances in $T^{(1)}$ and $T^{(2)}$ are off by a multiplicative $\kappa_1 = \log n$, Lemma 7.2 implies that $f^{(1)}$ satisfies $\mathbb{E}[\xi_{r^{(2)}}(\vec{f}^{(1)})] \leq 2\kappa_1 \xi_{r^{(2)}}(\vec{f}^{(2)}_{\text{opt}})$ so is in expectation $(2\kappa_1 - 1)$-approximate for $G^{(2)}$.

For $i \in \{2, 3, \ldots, c\}$, we have by our inductive hypothesis that $\vec{f}^{(i-1)}$ is in expectation $(2\kappa_{i-1} - 1)$-approximate for $G^{(i)}$. Therefore, by our choice of $K^{(i)} \geq \tau(T^{(i)}) \log(2\kappa_{i-1} - 1)$ and Theorem 4.1, we immediately see that the constructed $\vec{f}^{(i)}$ satisfies $\mathbb{E}[\xi_{r^{(i)}}(\vec{f}^{(i)})] \leq (1+1)\xi_{r^{(i)}}(\vec{f}^{(i)}_{\text{opt}})$ and is in expectation a 1-approximate for $G^{(i)}$. Furthermore, since the resistances in $T^{(i)}$ and $T^{(i+1)}$ are off by a multiplicative κ_i, Lemma 7.2 implies that $f^{(i)}$ satisfies $\mathbb{E}[\xi_{r^{(i+1)}}(\vec{f}^{(i)})] \leq 2\kappa_i \xi_{r^{(i+1)}}(\vec{f}^{(i+1)}_{\text{opt}})$ so is in expectation $(2\kappa_i - 1)$-approximate for $G^{(2)}$. $\qquad\square$

As a direct consequence of the above lemma we see that in an execution of `FullSolver`, $\vec{f}^{(c)}$ is in expectation a $(2\kappa_c - 1)$-approximate electrical flow in graph $G = G^{(c+1)}$. Therefore, when $\vec{f}^{(c)}$ is used as an initial, $\vec{f}_0$, for the last iteration ($i = c+1$), after $K := K^{(c+1)} - K' \geq \tau(T) \log(\frac{2\kappa_c - 1}{\varepsilon})$ cycle updates we have $\mathbb{E}[\xi_r(\vec{f}_K)] \leq (1+\varepsilon)\xi_r(\vec{f}_{\text{opt}})$. Therefore, by the same analysis of randomized stopping time as in Section 7.1, after running the final K' cycle updates the final flow $\vec{f}^{(c+1)} = \vec{f}_{K+K'}$ and $\vec{v}^{(c+1)} = \vec{v}_{K+K'}$ satisfy the error guarantee of Theorem 7.4.

$$\mathbb{E}[\xi_r(\vec{f}^{(c+1)})] \leq (1+\varepsilon) \cdot \xi_r(\vec{f}_{\text{opt}}) \quad \text{and} \quad \mathbb{E}\|\vec{v}^{(c+1)} - \mathbf{L}^\dagger \vec{\chi}\|_{\mathbf{L}} \leq \sqrt{\varepsilon} \cdot \|\mathbf{L}^\dagger \vec{\chi}\|_{\mathbf{L}}$$

To prove Theorem 7.4 all that remains is to bound the running time of `FullSolver`. As in `SimpleSolver` each cycle update takes $O(\log n)$ time so the total running time of `FullSolver` is

$$O\left(\sum_{i=1}^{c+1} K^{(i)} \log n\right) = O\left(m \log^2 n \log\log n \log(\varepsilon^{-1})\right) + O\left(\sum_{i=2}^{c} K^{(i)} \log n\right) .$$

To upper bound the second term we note the following

$$\begin{aligned}
O\left(\sum_{i=2}^{c} K^{(i)} \log n\right) &= O\left(\sum_{i=2}^{c} \left(\frac{\tau(T)}{\prod_{j=i}^{c} \kappa_j} + m\right) \log(2\kappa_{i-1} - 1) \log n\right) \\
&= O\left(\sum_{i=2}^{c} \left(\frac{\tau(T)}{\prod_{j=i}^{c} \kappa_j} + m\right) \kappa_i \log n\right) \qquad\qquad \text{(using } \kappa_i = \log \kappa_{i-1}) \\
&= O\left(\sum_{i=2}^{c} \frac{\tau(T)}{\prod_{j=i+1}^{c} \kappa_j} \log n + \sum_{i=2}^{c} m\kappa_i \log n\right) \\
&= O\left(\tau(T) \log n + m\kappa_2 \log n\right) = O(m \log^2 n \log\log n) .
\end{aligned}$$

The last step used that all $\kappa_j \geq 2$ so $\sum_{i=2}^{c} \frac{1}{\prod_{j=i+1}^{c} \kappa_j} \leq 1 + \frac{1}{2} + \frac{1}{4} + \cdots = O(1)$ and $\sum_{i \geq 2} \kappa_i = O(\kappa_2)$. This ends the proof of Theorem 7.4 with the desired running time $O(m \log^2 n \log\log n \log(\varepsilon^{-1}))$.

18

8 Numerical Stability

Up until this point our analysis has assumed that all arithmetic operations are exact. In this section, we show that our algorithm is numerically stable and achieves the same convergence guarantees when implemented with finite-precision arithmetic. We start with the analysis of `SimpleSolver`.

For simplicity of exposition, we assume that the resistances r_e and the coordinates of the demand vector $\vec{\chi}$ are all represented as b-bit integers with absolute values bounded by $N = 2^b$. We show that `SimpleSolver` works when arithmetic operations are implemented with $O(\max(b, \log n, \log 1/\varepsilon))$ bits of precision (which is necessary simply to guarantee we can represent an answer meeting the required error bound). In particular, if the entries of the input and ε can be stored in $\log n$-bit words, our running time guarantees hold in the standard unit cost RAM model, which only allows arithmetic operations on $O(\log n)$-bit numbers to be done in constant time.

We start our algorithm by multiplying the demand vector $\vec{\chi}$ by $\lceil 4mN^2/\varepsilon \rceil$, to ensure that in $\vec{f}_{\mathrm{opt}}$ there exist at least $\lceil 4mN^2/\varepsilon \rceil$ total units of flow on the edges. Since the resistances are at least 1, this guarantees that $\xi_r(\vec{f}_{\mathrm{opt}}) \geq \lceil 4mN^2/\varepsilon \rceil$. Next, we ensure that throughout our algorithm all flow vectors $\vec{f}_i$ are integer vectors. At each iteration of `SimpleSolver`, when we are given a primal feasible flow $\vec{f}$ and want to compute the optimal cycle update $\alpha^* \overset{\mathrm{def}}{=} \frac{\vec{f}^T \mathbf{R} \vec{c}_e}{\vec{c}_e^T \mathbf{R} \vec{c}_e}$, we round this fraction to the nearest integer and suppose we pick some $\tilde{\alpha} \in \mathbb{Z}$ such that $|\alpha^* - \tilde{\alpha}| \leq \frac{1}{2}$.

Based on the modifications above, it is easy to see that our algorithm can be run on a RAM machine with word size $O(\max(b, \log n, \log 1/\varepsilon))$, since all flow values are integers bounded above by $O(\mathsf{poly}(N, n, 1/\varepsilon))$. All we have left to show is the convergence analysis for such integral updates.

We do so by strengthening our cycle update lemma, Lemma 4.3. For each cycle update, if we add $\tilde{\alpha} = \alpha^*(1 + \delta)$ units of flow on the cycle instead of α^*, the corresponding energy decrease is

$$
\begin{aligned}
\xi_r(\vec{f} - \alpha^*(1+\delta)\vec{c}_e) - \xi_r(\vec{f}) &= (\vec{f} - \alpha^*(1+\delta)\vec{c}_e)^T \mathbf{R}(\vec{f} - \alpha^*(1+\delta)\vec{c}_e) - \vec{f}^T \mathbf{R} \vec{f} \\
&= -2\alpha^*(1+\delta)\vec{f}^T \mathbf{R}\vec{c}_e + (\alpha^*(1+\delta))^2 \vec{c}_e^T \mathbf{R}\vec{c}_e = -\frac{\Delta_{c_e}(\vec{f})^2}{R_e}(1 - \delta^2) \; ,
\end{aligned}
$$

where the last equality has used the fact that $R_e = \vec{c}_e^T \mathbf{R}\vec{c}_e$. We have $|\delta| \leq \frac{1}{2\alpha^*}$, so, as long as $\alpha^* \geq 1$, the decrease in the energy decrease is at least $\frac{3}{4}$ of what it would be for $\delta = 0$. We call an off-tree edge "good" if its $\alpha^* \geq 1$, and "bad" otherwise. We can rewrite the duality gap as

$$
\begin{aligned}
\mathrm{gap}(\vec{f}, \vec{v}) = \sum_{e \in E \backslash T} \frac{\Delta_{c_e}(\vec{f})^2}{r_e} &= \sum_{e \in E \backslash T} \frac{\Delta_{c_e}(\vec{f})^2}{R_e} \frac{R_e}{r_e} = \sum_{\substack{e \in E \backslash T \\ e \text{ is bad}}} \frac{\Delta_{c_e}(\vec{f})^2}{R_e^2} \frac{R_e^2}{r_e} + \sum_{\substack{e \in E \backslash T \\ e \text{ is good}}} \frac{\Delta_{c_e}(\vec{f})^2}{R_e} \frac{R_e}{r_e} \\
&\leq mN^2 + \sum_{\substack{e \in E \backslash T \\ e \text{ is good}}} \frac{\Delta_{c_e}(\vec{f})^2}{R_e} \frac{R_e}{r_e} \; .
\end{aligned}
$$

As a consequence, if one samples[10] each tree cycle $\vec{c}_e$ with probability to $\frac{R_e}{r_e \tau}$, then the expected energy decrease is at least:

$$
\begin{aligned}
\mathbb{E}\left[\xi_r(\vec{f}_i) - \xi_r(\vec{f}_{i-1}) \Big| \mathrm{gap}(\vec{f}_{i-1}, \vec{v}_{i-1})\right] &\leq \sum_{\substack{e \in E \backslash T \\ e \text{ is good}}} \left(\frac{R_e}{r_e \tau}\right)\left(-\frac{\Delta_{c_e}(\vec{f})^2}{R_e} \frac{3}{4}\right) \\
&\leq -\frac{\left(\mathrm{gap}(\vec{f}_{i-1}, \vec{v}_{i-1}) - mN^2\right)}{4\tau/3} \leq -\frac{\left(\xi_r(\vec{f}_{i-1}) - \xi_r(\vec{f}_{\mathrm{opt}}) - mN^2\right)}{4\tau/3} \; .
\end{aligned}
$$

---

[10]Recall that *exact* sampling can be done in *expected* constant time on a machine with $O(\log n)$-sized words.

If one defines a random variable $D_i \stackrel{\text{def}}{=} \xi_r(\vec{f_i}) - \xi_r(\vec{f}_{\text{opt}}) - mN^2$, then

$$\mathbb{E}[D_i|D_{i-1}] \leq \left(1 - \frac{1}{4\tau/3}\right) D_{i-1} \ .$$

Therefore, using the same analysis as before, after $K = \frac{4\tau}{3} \log \frac{2\text{st}(T)}{\varepsilon \cdot p}$, we have with probability at least $1 - p$ that:

$$\xi_r(\vec{f_K}) \leq \left(1 + \frac{\varepsilon}{2}\right) \left(\xi_r(\vec{f}_{\text{opt}}) + mN^2\right) \leq (1+\varepsilon)\xi_r(\vec{f}_{\text{opt}}) \ .$$

We note that the above analysis extends to `ExampleSolver` and `FullSolver`. In fact, one can always round those scaling factor κ's to their nearest integers and verify that the same analysis for running time and convergence holds. This ensures that throughout our algorithm each graph $G^{(i)}$ has its edge resistances all being rational numbers with the same integer denominator that is bounded above by $\tilde{O}(\log n)$. Therefore, when solving the approximate electrical flow for each graph $G^{(i)}$, we can simply scale up all resistances by their common denominator, and this reduces the analysis on numerical stability to the earlier case of `SimpleSolver`.

9   A Geometric Interpretation via Alternating Projections

In this section, we present a geometric interpretation of our algorithm and provide an alternative analysis of its convergence based on the Method of Alternating Projections [BV04, ER11]. This method provides a generic framework for iteratively finding a point in the intersection of a number of convex constraint sets by sequentially projecting the current solution onto a set at the time. The specialization of this method to systems of linear equations is known as the Kaczmarz method [Kac37] and is commonly used in practice, especially in the field of Computerized Tomography [Nat01] to solve large linear systems efficiently.

An important contribution towards a better understanding of the Kaczmarz method was given by Strohmer and Vershynin [SV09], who showed that a randomized version of such method converges exponentially with a rate equal to the scaled condition number of the system. We remark that `SimpleSolver` can be interpreted as applying this randomized Kaczmarz method to the scaled system of KPL equations:

$$\forall e \in E \setminus T \ , \ \frac{1}{\sqrt{r_e}} \vec{f}^T \mathbf{R} \vec{c}_e = 0 \ .$$

Our analysis effectively shows that the tree condition number τ plays exactly the role of the scaled condition number of the system in the randomized Kaczmarz method.

9.1   Geometric View

Given the low-stretch spanning tree T and the corresponding basis $\{\vec{c}_e\}_{e \in E \setminus T}$ of the cycle space of G, let $P_e \stackrel{\text{def}}{=} \{\vec{f} \in \mathbb{R}^E : \vec{f}^T \mathbf{R} \vec{c}_e = 0\}$ be the hyperplane of flows $\vec{f}$ respecting the KPL condition over circulation $\vec{c}_e$. Then, the optimality condition with respect to a basis of the cycle space can be viewed as requiring the flow $\vec{f}$ to be in the intersection of the $(m - n + 1)$ hyperplanes $\{P_e\}_{e \in E \setminus T}$.

From this geometric perspective, at every iteration, our algorithm picks a hyperplane P_e associated with a basis vector $\vec{c}_e$, and projects the current flow $\vec{f_i}$ onto P_e. Formally, we can check that, at iteration i, our algorithms applies to the current flow the projection Π_{e_i} onto P_{e_i}:

$$\Pi_{e_i} \vec{f_i} = \left(\mathbf{I} - \frac{\vec{c}_{e_i} \vec{c}_{e_i}^T \mathbf{R}}{\|\vec{c}_{e_i}\|_{\mathbf{R}}^2}\right) \vec{f_i} = \vec{f_i} - \frac{\vec{f_i}^T \mathbf{R} \vec{c}_{e_i}}{\|\vec{c}_{e_i}\|_{\mathbf{R}}^2} \vec{c}_{e_i} = \vec{f}_{i+1} \ . \tag{7}$$

Notice that, as this update adds a circulation to $\vec{f_i}$, the resulting flow $\vec{f}_{i+1}$ meets the demands $\vec{\chi}$.

Our analysis shows that by iteratively projecting $\vec{f_i}$ onto different hyperplanes in $\{P_e\}_{e \in E \setminus T}$, the final flow can be made arbitrarily close to the intersection $\bigcap_{e \in E \setminus T} P_e$, i.e. the unique electrical flow.

9.2 Alternating Projections

The geometric interpretation above casts our algorithm as an instance of the Method of Alternating Projections [BV04, ER11]. This method is an algorithmic paradigm used to solve the following generic problem. Given k constraints in the form of convex sets $S_1, \ldots, S_k \in \mathbb{R}^n$ such that $\bigcap_{i=1}^{k} S_i$ is non-empty, and we have the ability to project onto each individual S_i, i.e. given $x \in \mathbb{R}^n$ we can efficiently compute the projected point $\Pi_i(x) \in S_i$ that is closest to x. The goal is to find (or come arbitrarily close to) a point in $\bigcap_{i=1}^{k} S_i$.

A generic alternating-projection algorithm solves this problem by iteratively applying a sequence of projections $\Pi_{i_1}, \Pi_{i_2}, \ldots$ to a starting point $x_0 \in \mathbb{R}^n$. Given the current solution x_t, the next iterate is defined as $x_{t+1} = \Pi_{i_t}(x_t)$. In the case of our electrical-flow algorithm, the convex sets $\{S_i\}$ consist of the hyperplanes $\{P_e\}_{e \in E \setminus T}$ and the notion of distance is given by the resistance norm $\|x\|_{\mathbf{R}} = \sqrt{x^T \mathbf{R} x}$.

The analysis of the convergence of alternating-projection algorithms exploits the convexity of the sets to argue that, at each iteration t, for any x^* in the target set $\bigcap_{i=1}^{n} S_i$,

$$\|x_{t+1} - x^*\|^2 \le \|x_t - x^*\|^2 - \|x_{t+1} - x_t\|^2 \ . \tag{8}$$

In words, the additive progress in the direction of x^* in one iteration is at least the squared distance between x_{t+1} and x_t, i.e. the amount our solution has moved as a result of the projection. Hence, the convergence is fastest when we are able to pick the next projection to be onto a constraint that is substantially violated.

9.3 A Randomized Kaczmarz Algorithm

When the convex sets are hyperplanes, the alternating-projection algorithm specializes to the Kaczmarz method [Kac37]. Different variants of this method have proved effective at solving systems of sparse linear equations in computerized tomography [Nat01] and image reconstruction [Her09].

The convergence analysis still relies on (8). For all $e \in E \setminus T$, define the normalized vector $\hat{c}_e \overset{\text{def}}{=} \frac{\vec{c}_e}{\|\vec{c}_e\|_{\mathbf{R}}}$. For the specific case of our algorithm, (8) yields

$$\|\vec{f}_t - \vec{f}_{\text{opt}}\|_{\mathbf{R}}^2 - \|\vec{f}_{t+1} - \vec{f}_{\text{opt}}\|_{\mathbf{R}}^2 \ge \|\vec{f}_{t+1} - \vec{f}_t\|_{\mathbf{R}}^2 = (\vec{f}_t^T \mathbf{R} \hat{c}_{e_t})^2 \ . \tag{9}$$

Equivalently, we make fast progress at iteration t if we project over P_{e_t} such that $(\vec{f}_t^T \mathbf{R} \hat{c}_{e_t})^2$ is large with respect to our current distance from optimal.

Recalling that, by Lemma 2.5 at optimality $\vec{f}_{\text{opt}}^T \mathbf{R} \vec{c} = 0$ for all circulations $\vec{c}$, we can also rephrase our task as finding a basis direction $\hat{c}_e$, $e \in E \setminus T$, such that $\left((\vec{f}_t - \vec{f}_{\text{opt}})^T \mathbf{R} \hat{c}_e\right)^2$ is a large fraction of $\|\vec{f}_t - \vec{f}_{\text{opt}}\|_{\mathbf{R}}^2$, i.e. $\hat{c}_e$ captures a large component of the current error $\vec{f}_t - \vec{f}_{\text{opt}}$.

This observation raises two main challenges. First, there may be no basis direction $\hat{c}_e$ that is well correlated with the error. Second, even if such a basis direction exists, the computation required to find it may be too slow for our purposes, e.g. it might require looking at $\vec{c}_e$ for all $e \in E \setminus T$.

Our algorithm solves both these issues by a randomized approach, that is to sample a direction $\hat{c}_e$ for $e \in E \setminus T$ with probability p_e proportional to $\frac{R_e}{r_e}$, and then apply the corresponding projection Π_e. We prove that this makes sufficient progress in expectation by exploiting the fact that the basis $\{\vec{c}_e\}_{e \in E \setminus T}$ comes from a low-stretch spanning tree. In particular, the following property of the basis $\{\vec{c}_e\}_{e \in E \setminus T}$ is crucial to our argument, and its proof is similar to that of Lemma 4.5.

Lemma 9.1. *For any circulation $\vec{g}$, we have*

$$\sum_{e \in E \setminus T} p_e (\vec{g}^T \mathbf{R} \hat{c}_e)^2 \ge \frac{\|\vec{g}\|_{\mathbf{R}}^2}{\tau} \ . \tag{10}$$

Proof. Recall that, for $e = (a, b) \in E \setminus T$, $\vec{p}_e \in \mathbb{R}^E$ is the vector corresponding to the unique path from a to

b by T. By the definition of $\vec{c}_e$, we have:

$$\sum_{e \in E \setminus T} p_e (\vec{g}^T \mathbf{R} \vec{c}_e)^2 = \frac{1}{\tau} \sum_{e \in E \setminus T} \frac{(g_e r_e + \vec{g}^T \mathbf{R}(-\vec{p}_e))^2}{r_e}$$

$$\geq \frac{1}{\tau} \left(\sum_{e \in E \setminus T} r_e g_e^2 - 2 \sum_{e \in E \setminus T} g_e \cdot (\vec{g}^T \mathbf{R} \vec{p}_e) \right) = \frac{1}{\tau} \left(\sum_{e \in E \setminus T} r_e g_e^2 + 2 \vec{g}^T \mathbf{R} \left(- \sum_{e \in E \setminus T} g_e \vec{p}_e \right) \right).$$

Since $\vec{g}$ is a circulation, its values on the off-tree edges determines its values on the tree edges. Consequently,

$$\forall e' \in E \; : \; \left(- \sum_{e \in E \setminus T} g_e \vec{p}_e \right)_{e'} = \begin{cases} g_{e'} & e' \in T \\ 0 & e' \notin T \end{cases}.$$

This yields:

$$\sum_{e \in E \setminus T} p_e (\vec{g}^T \mathbf{R} \hat{c}_e)^2 \geq \frac{1}{\tau} \left(\sum_{e \in E \setminus T} r_e g_e^2 + 2 \sum_{e \in T} r_e g_e^2 \right) \geq \frac{\|\vec{g}\|_{\mathbf{R}}^2}{\tau}.$$

$\square$

The geometric meaning of this lemma is that, in expectation under distribution $\vec{p}$, any circulation $\vec{c}$ has correlation at least $\frac{1}{\tau}$ with the basis directions $\{\vec{c}_e\}_{e \in E \setminus T}$. We remark here that the uniform distribution over an arbitrary orthonormal basis of the cycle space would satisfy (10) with τ replaced by the number of dimensions $(m - n + 1)$, and would yield the best possible correlation bound for our randomized approach. However, it is not known how to produce such a basis efficiently enough. Moreover, we also rely on the compact representation of our basis $\{\vec{c}_e\}_{e \in E \setminus T}$ as a spanning tree to design our data structure in Section 5. By Lemma 9.1, we obtain that our expected progress is

$$\mathbb{E}_{e_t \leftarrow \vec{p}} \left[\left((\vec{f}_t - \vec{f}_{\mathrm{opt}})^T \mathbf{R} \hat{c}_{e_t} \right)^2 \right] = \sum_{e \in E \setminus T} p_e \left((\vec{f}_t - \vec{f}_{\mathrm{opt}})^T \mathbf{R} \hat{c}_e \right)^2 \geq \frac{1}{\tau} \cdot \| \vec{f}_t - \vec{f}_{\mathrm{opt}} \|_{\mathbf{R}}^2 .$$

Hence, by (9), the expected distance of our current solution from optimal decreases as

$$\mathbb{E} \left[\| \vec{f}_t - \vec{f}_{\mathrm{opt}} \|_{\mathbf{R}}^2 \right] \leq \left(1 - \frac{1}{\tau} \right)^t \| \vec{f}_0 - \vec{f}_{\mathrm{opt}} \|_{\mathbf{R}}^2 . \tag{11}$$

To bound our initial distance from optimum, we we note that for any feasible $\vec{f} \in \mathbb{R}^E$ the optimality of $\vec{f}_{\mathrm{opt}}$ implies

$$\| \vec{f} \|_{\mathbf{R}}^2 = \| \vec{f}_{\mathrm{opt}} \|_{\mathbf{R}}^2 + \| \vec{f} - \vec{f}_{\mathrm{opt}} \|_{\mathbf{R}}^2 .$$

Then, by Lemma 6.1, it must be the case that $\| \vec{f}_0 - \vec{f}_{\mathrm{opt}} \|_{\mathbf{R}}^2 \leq \mathrm{st}(T) \cdot \| \vec{f}_{\mathrm{opt}} \|_{\mathbf{R}}^2$. Combined with (11), this shows that after $K = \tau \log \frac{\mathrm{st}(T)}{\varepsilon}$ iterations, $\mathbb{E} \left[\| \vec{f}_K - \vec{f}_{\mathrm{opt}} \|_{\mathbf{R}}^2 \right] \leq \varepsilon \| \vec{f}_{\mathrm{opt}} \|_{\mathbf{R}}^2$.

10 An Operator View: Linearity and Approximating $\mathbf{L}^\dagger$

In this section, we depart from the electrical perspective of Section 3 and the geometric one of Section 9 to present an interpretation of the `SimpleSolver` algorithm as a composition of simple linear operators. The main results of this section are Theorem 10.1, which shows the linearity of `SimpleSolver`, and Theorem 10.2 and its corollaries, which prove that the linear operator corresponding to an execution of `SimpleSolver` is an approximation of the Laplacian pseudoinverse with large probability.

10.1 Linearity of `SimpleSolver`

In Section 9 we showed that each cycle update of `SimpleSolver` is the application of projector Π_{e_i} to $\vec{f}_i$ where e_i is the off-tree edge sample in iteration i and the projectors are defined by

$$\forall e \in E \setminus T \ : \ \Pi_e \stackrel{\text{def}}{=} \left(\mathbf{I} - \frac{\vec{c}_e \vec{c}_e^T \mathbf{R}}{\|\vec{c}_e\|_{\mathbf{R}}^2} \right) \ .$$

The projectors $\{\Pi_e\}_{e \subset E \setminus T}$ as well as the incidence matrix for the spanning tree T, denoted $\mathbf{B}_T \in \mathbb{R}^{E \times V}$, and the Laplacian of T, denoted $\mathbf{L}_T \stackrel{\text{def}}{=} \mathbf{B}_T^T \mathbf{R}^{-1} \mathbf{B}_T$, give us the necessary building blocks to describe `SimpleSolver` as a linear operator.

Theorem 10.1. *Consider an execution of `SimpleSolver` on demand vector $\vec{\chi}$ for K iterations of cycle updates. Let $e_1, \ldots, e_K$ be the random off-tree edges picked from the probability distribution $\{p_e\}$ by the algorithm. Then, the final flow $\vec{f}_K$ and tree induced voltages $\vec{v}_K$ are given by*

$$\vec{f}_K = \mathbf{F}_K \vec{\chi} \ , \ \ \vec{v}_K = \mathbf{V}_K \vec{\chi},$$

where the flow operator $\mathbf{F}_K$ *and the* voltage operator $\mathbf{V}_K$ *are defined as:*

$$\mathbf{F}_K \stackrel{\text{def}}{=} \left(\prod_{j=1}^K \Pi_{e_j} \right) \mathbf{R}^{-1} \mathbf{B}_T \mathbf{L}_T^\dagger \ , \ \ \mathbf{V}_K \stackrel{\text{def}}{=} \mathbf{L}_T^\dagger \mathbf{B}_T^T \left(\prod_{j=1}^K \Pi_{e_j} \right) \mathbf{R}^{-1} \mathbf{B}_T \mathbf{L}_T^\dagger.$$

Proof. The initial flow $\vec{f}_0$ is the electrical flow over the tree T and therefore $\vec{f}_0 = \mathbf{R}^{-1} \mathbf{B}_T \mathbf{L}_T^\dagger \vec{\chi}$. Since applying Π_e corresponds to a cycle update we have $\vec{f}_K = \left(\prod_{j=1}^K \Pi_{e_j} \right) \vec{f}_0$. Furthermore, we see that the final tree induced voltages for $\vec{f}_K$ can be obtained by $\vec{v}_K = \mathbf{L}^\dagger \mathbf{B}_T^T \vec{f}_K$. Combining these facts yields the result. $\qquad \square$

The ideas of Theorem 10.1 apply to `FullSolver` and one can show that for a fixed choice of sampled edges `FullSolver` is also linear operator. Furthermore, for `SimpleSolver` we see that we could make the flow and voltage operators symmetric without a loss in asymptotic running time or error guarantees by simply applying the projections a second time in reverse.

10.2 Approximation of Laplacian Pseudoinverse

In this section, we show that the flow and voltage operators $\mathbf{F}_K$ and $\mathbf{V}_K$ arising from `SimpleSolver` yield an approximate representation of the pseudoinverse $\mathbf{L}^\dagger$ and related operators as the composition of a number of simple linear operators. Our notion of approximation will be with respect to the Frobenius norm. We denote the Frobenius norm of a matrix $\mathbf{X}$ by $\|\mathbf{X}\|_F$ and recall that $\|\mathbf{X}\|_F^2 \stackrel{\text{def}}{=} \text{Tr}(\mathbf{X}^T \mathbf{X})$.

For the purpose of this section, the natural notion of inner product in $\mathbb{R}^E$ is given by the resistance operator $\mathbf{R}$. To make the proof simpler to follow, we scale some of the operators defined so far as to be able to use the standard notion of inner product. We define

$$\forall e \in E \setminus T \ : \ \tilde{\Pi}_e \stackrel{\text{def}}{=} \mathbf{R}^{1/2} \Pi_e \mathbf{R}^{-1/2} = \left(\mathbf{I} - \frac{\mathbf{R}^{1/2} \vec{c}_e \vec{c}_e^T \mathbf{R}^{1/2}}{\|\vec{c}_e\|_{\mathbf{R}}^2} \right) \ .$$

This shows that $\tilde{\Pi}_e$ is an orthogonal projection. Equivalently, before the scaling, the operator Π_e is an orthogonal projection with respect to the inner product defined by $\mathbf{R}$.

The main theorem of this section involves showing a relation between the operator $\left(\prod_{j=1}^K \tilde{\Pi}_{e_j} \right)$, which describes the cycle updates performed by `SimpleSolver`, and the projection operator $\Pi_G \in \mathbb{R}^{E \times E}$, defined as:

$$\Pi_G \stackrel{\text{def}}{=} \mathbf{R}^{-1/2} \mathbf{B} \mathbf{L}^\dagger \mathbf{B}^T \mathbf{R}^{-1/2} \ .$$

The projection Π_G takes any flow $\mathbf{R}^{1/2}\vec{f}$ and projects it orthogonally onto the subspace of KPL-respecting flows, i.e. electrical flows. The operator Π_G plays an important role in the spectral study of graphs and is the main object of study in spectral sparsification [SS11].

Theorem 10.2. *Let $e_1, \ldots, e_K$ be random off-tree edges picked independently from the $\{p_e\}$ probability distribution. Let $K = \lceil \tau \log \frac{n}{\varepsilon p} \rceil$. Then, with probability at least $1 - p$,*

$$\left\| \left(\prod_{j=1}^{K} \tilde{\Pi}_{e_j} \right) - \Pi_G \right\|_F^2 \leq \varepsilon \ .$$

We defer the proof to the end of this section and first show how two useful bounds involving the flow and voltage operators follow directly from Theorem 10.2.

Theorem 10.3. *Let $e_1, \ldots, e_K$ be random off-tree edges picked independently from the $\{p_e\}$ probability distribution. Let $K = \lceil \tau \log \frac{n}{\varepsilon p} \rceil$. Then, with probability at least $1 - p$,*

$$(\mathbf{F}_K - \mathbf{R}^{-1}\mathbf{B}\mathbf{L}^{\dagger})^T \mathbf{R}(\mathbf{F}_K - \mathbf{R}^{-1}\mathbf{B}\mathbf{L}^{\dagger}) \preceq \varepsilon \mathbf{L}_T^{\dagger} \preceq \varepsilon \text{st}(T) \mathbf{L}^{\dagger} \tag{12}$$

and

$$(\mathbf{V}_K - \mathbf{L}^{\dagger})^T \mathbf{L}(\mathbf{V}_K - \mathbf{L}^{\dagger}) \preceq \varepsilon (\text{st}(T))^2 \mathbf{L}^{\dagger} \ .$$

In particular, the first equation implies that, for $\delta > 0$ with probability at least $1 - p$, a random sequence of $K' = \lceil \tau \log \frac{n \cdot \text{st}(T)}{\delta p} \rceil$ cycle updates yield a flow operator $\mathbf{F}_{K'}$ which is a multiplicative δ-approximation to the electrical-flow operator $\mathbf{R}^{-1}\mathbf{B}\mathbf{L}^{\dagger}$. In practice, this means that, if we choose K' cycles at random according to $\{p_e\}$, the resulting sequence of updates will yield a flow whose energy is at most a factor of $(1 + \delta)$ away from optimum, *for all* possible initial demand vectors $\vec{\chi}$.

Similarly, the second equation implies that $K' = \lceil \tau \log \frac{n \cdot \text{st}(T)^2}{\delta p} \rceil$ randomly selected cycle updates yields with probability at least $1 - p$ a voltage operator that when applied to any demand $\vec{\chi}$ results in a $\sqrt{\delta}$-approximate solution to $\mathbf{L}\vec{v} = \vec{\chi}$.

Proof of Theorem 10.3. Theorem 10.2 implies:

$$\left(\left(\prod_{j=1}^{K} \tilde{\Pi}_{e_j} \right) - \Pi_G \right)^T \left(\left(\prod_{j=1}^{K} \tilde{\Pi}_{e_j} \right) - \Pi_G \right) \preceq \varepsilon \mathbf{I} \ .$$

It suffices to left- and right-multiply this inequality by $\mathbf{R}^{-1/2}\mathbf{B}_T^T \mathbf{L}_T^{\dagger}$ to obtain:

$$(\mathbf{F}_K - \mathbf{R}^{-1}\mathbf{B}\mathbf{L}^{\dagger})^T \mathbf{R}(\mathbf{F}_K - \mathbf{R}^{-1}\mathbf{B}\mathbf{L}^{\dagger}) \preceq \varepsilon \mathbf{L}_T^{\dagger}$$

as required. To obtain the inequality involving $\mathbf{L}^{\dagger}$, we can directly apply the fact that $\mathbf{L} \preceq \text{st}(T) \mathbf{L}_T$ (cf. [ST12, Lemma 9.2]).

We now derive the bound for the voltage operator. This requires the following chain of inequalities:

$$\mathbf{R} \succeq \mathbf{R}^{1/2}(\mathbf{R}^{-1/2}\mathbf{B}_T \mathbf{L}_T^{\dagger} \mathbf{B}_T^T \mathbf{R}^{-1/2})\mathbf{R}^{1/2} = \mathbf{B}_T \mathbf{L}_T^{\dagger} \mathbf{B}_T^T \succeq \frac{1}{\text{st}(T)} \mathbf{B}_T \mathbf{L}_T^{\dagger} \mathbf{L} \mathbf{L}_T^{\dagger} \mathbf{B}_T^T \ .$$

The first inequality follows as $\mathbf{R}^{-1/2}\mathbf{B}_T \mathbf{L}_T^{\dagger} \mathbf{B}_T^T \mathbf{R}^{-1/2}$ is an orthogonal projection. The second one is again a consequence of the fact that $\mathbf{L} \preceq \text{st}(T) \mathbf{L}_T$. It now suffices to replace $\mathbf{R}$ by $\frac{1}{\text{st}(T)}\mathbf{B}_T \mathbf{L}_T^{\dagger} \mathbf{L} \mathbf{L}_T^{\dagger} \mathbf{B}_T^T$ in (12) to obtain the required bound. $\qquad\square$

The rest of the section is dedicated to proving Theorem 10.2. This could be achieved by slightly modifying the proof of Section 4. However, in the following, we prefer to briefly give an operator-based proof of the theorem. We believe that this complements the view of Section 4 and helps to convey its geometric flavor.

24

10.2.1 Proof of Theorem 10.2

The spanning tree T allows us to construct a projector Π_T that can be used to relate Π_G and $\left(\prod_{j=1}^{K} \tilde{\Pi}_{e_j}\right)$. The projector Π_T is defined as

$$\Pi_T \stackrel{\text{def}}{=} \mathbf{R}^{-1/2}\mathbf{B}\mathbf{L}_T^{\dagger}\mathbf{B}_T^T\mathbf{R}^{-1/2} \ .$$

Here, it is important to notice that Π_T is not symmetric, i.e. it is not an orthogonal projector, but an oblique one. The range of Π_T is the range of $\mathbf{R}^{-1/2}\mathbf{B}^T$ i.e. the space of KPL-respecting flows. The following lemma relates Π_T to the tree cycle vectors $\{\vec{c}_e\}_{e \in E \setminus T}$.

Lemma 10.4. *Let $\vec{1}_e \in \mathbb{R}^E$ be the standard unit vector associated to $e \in E$. Then:*

$$(\mathbf{I} - \Pi_T)^T\mathbf{R}^{1/2}\vec{1}_e = \begin{cases} \mathbf{R}^{1/2}\vec{c}_e, & e \in E \setminus T \ , \\ 0, & e \in T \ . \end{cases}$$

Proof. For $e \in T$, we have:

$$(\mathbf{I} - \Pi_T)^T\mathbf{R}^{1/2}\vec{1}_e = \mathbf{R}^{1/2}\vec{1}_e - \mathbf{R}^{-1/2}\mathbf{B}^T\mathbf{L}_T^{\dagger}\mathbf{B}_T\vec{1}_e = \mathbf{R}^{1/2}(\vec{1}_e - \vec{1}_e) = 0 \ .$$

For $e \in E \setminus T$:

$$(\mathbf{I} - \Pi_T)^T\mathbf{R}^{1/2}\vec{1}_e = \mathbf{R}^{1/2}\vec{1}_e - \mathbf{R}^{-1/2}\mathbf{B}^T\mathbf{L}_T^{\dagger}\mathbf{B}_T\vec{1}_e = \mathbf{R}^{1/2}(\vec{1}_e - \vec{p}_e) = \mathbf{R}^{1/2}\vec{c}_e \ .$$

$\square$

In words, the rows corresponding to off-tree edges in the matrix representation of $\mathbf{R}^{1/2}(\mathbf{I} - \Pi_T)\mathbf{R}^{-1/2}$ in the standard basis are exactly the tree cycle vectors $\{\vec{c}_e\}_{e \in E \setminus T}$. The remaining rows are 0.

In the next lemma, we relate Π_T to the cycle projections $\{\tilde{\Pi}_e\}_{e \in E \setminus T}$ used in `SimpleSolver`.

Lemma 10.5. *Let $e \in E \setminus T$ be sampled according to probability $\{p_e\}$, as in* ***SimpleSolver***. *Then:*

$$\mathbb{E}[\tilde{\Pi}_e] = \mathbf{I} - \frac{1}{\tau}(\mathbf{I} - \Pi_T)^T(\mathbf{I} - \Pi_T) \ .$$

Proof. This follows from the definition of $\{p_e\}$ and Lemma 10.4:

$$\mathbb{E}[\tilde{\Pi}_e] = \mathbf{I} - \frac{1}{\tau} \sum_{e \in E \setminus T} \frac{\mathbf{R}^{1/2}\vec{c}_e\vec{c}_e^T\mathbf{R}^{1/2}}{r_e} = \mathbf{I} - \frac{1}{\tau}(\mathbf{I} - \Pi_T)^T(\mathbf{I} - \Pi_T) \ .$$

$\square$

Next, we show how to relate $(\mathbf{I} - \Pi_T)^T(\mathbf{I} - \Pi_T)$ to $\mathbf{I} - \Pi_G$. We will show that, for any flow $\mathbf{R}^{1/2}\vec{f}$, the distance $\left\|\mathbf{R}^{1/2}\vec{f} - \Pi_G\mathbf{R}^{1/2}\vec{f}\right\|_2$ to its Π_G-projection is smaller than $\left\|\mathbf{R}^{1/2}\vec{f} - \Pi_T\mathbf{R}^{1/2}\vec{f}\right\|_2$, the distance to its Π_T-projection. It is easy to see that this should be the case, because Π_G and Π_T project onto the same subspace and Π_G is an orthogonal projection. We prove it formally here. This is analogous to Lemma 9.1.

Lemma 10.6.
$$(\mathbf{I} - \Pi_T)^T(\mathbf{I} - \Pi_T) \succeq (\mathbf{I} - \Pi_G)^T(\mathbf{I} - \Pi_G) = \mathbf{I} - \Pi_G \ .$$

Proof. Algebraically, we can check that

$$(\Pi_G - \Pi_T)^T(\Pi_G - \Pi_T) + (\mathbf{I} - \Pi_G) = \Pi_G - \Pi_T^T\Pi_G - \Pi_G\Pi_T + \Pi_T^T\Pi_T + \mathbf{I} - \Pi_G =$$
$$\mathbf{I} - \Pi_T^T - \Pi_T + \Pi_T^T\Pi_T = (\mathbf{I} - \Pi_T)^T(\mathbf{I} - \Pi_T) \ .$$

In the second equality, we used the fact that $\Pi_G\Pi_T = \Pi_T$ as Π_G and Π_T project onto the same subspace. As $(\Pi_G - \Pi_T)^T(\Pi_G - \Pi_T) \succeq 0$, the lemma follows. $\square$

We are now ready to prove Theorem 10.2.

Proof of Theorem 10.2. For any iteration t, $0 \leq t \leq K$, let

$$\Phi_t \stackrel{\text{def}}{=} \left\| \left(\prod_{j=1}^{t} \tilde{\Pi}_{e_j} \right) - \Pi_G \right\|_F^2 .$$

We will show that $\Phi_K \leq \varepsilon$ with probability at least $1-p$, by bounding the expected reduction in Φ_t at every iteration. Notice that for any $e \in E \setminus T$, we have $\Pi_G \tilde{\Pi}_e = \tilde{\Pi}_e \Pi_G = \Pi_G$, as Π_G and $\tilde{\Pi}_e$ are both orthogonal projectors and the range of Π_G is contained into that of $\tilde{\Pi}_e$. Hence, we can simplify the expression for Φ_t as

$$\Phi_t = \text{Tr} \left(\left(\prod_{j=1}^{t} \tilde{\Pi}_{e_j} \right)^T \left(\prod_{j=1}^{t} \tilde{\Pi}_{e_j} \right) - \Pi_G \right) .$$

Now, we are ready to bound the change in Φ_t in one iteration. For any t, $t \leq 0 \leq K-1$, assume that $e_1, \cdots, e_t$ have been fixed and notice:

$$\mathbb{E}[\Phi_{t+1}] = \mathbb{E}\left[\text{Tr} \left(\left(\prod_{j=1}^{t+1} \tilde{\Pi}_{e_j} \right)^T \left(\prod_{j=1}^{t+1} \tilde{\Pi}_{e_j} \right) - \Pi_G \right) \right] = \text{Tr} \left(\left(\prod_{j=1}^{t} \tilde{\Pi}_{e_j} \right)^T \mathbb{E}[\tilde{\Pi}_{e_{t+1}}] \left(\prod_{j=1}^{t} \tilde{\Pi}_{e_j} \right) - \Pi_G \right) .$$

By Lemma 10.5, we obtain:

$$\mathbb{E}[\Phi_{t+1}] = \Phi_t - \frac{1}{\tau} \text{Tr} \left(\left(\prod_{j=1}^{t} \tilde{\Pi}_{e_j} \right)^T (\mathbf{I} - \Pi_T)^T (\mathbf{I} - \Pi_T) \left(\prod_{j=1}^{t} \tilde{\Pi}_{e_j} \right) \right) .$$

We now apply Lemma 10.6:

$$\mathbb{E}[\Phi_{t+1}] \leq \Phi_t - \frac{1}{\tau} \text{Tr} \left(\left(\prod_{j=1}^{t} \tilde{\Pi}_{e_j} \right)^T (\mathbf{I} - \Pi_G) \left(\prod_{j=1}^{t} \tilde{\Pi}_{e_j} \right) \right) .$$

This yields that $\mathbb{E}[\Phi_{t+1}] \leq \left(1 - \frac{1}{\tau} \right) \Phi_t$ and by taking expectation over all the choices of sampled edges, we have $\mathbb{E}[\Phi_K] \leq \left(1 - \frac{1}{\tau} \right)^K \Phi_0$. Now, by definition, $\Phi_0 = \|\Pi_G\|_F^2$ and because Π_G is a projector onto an $(n-1)$ dimensional space, we have $\Phi_0 = n - 1$. Hence, for our choice of $K = \lceil \tau \log \frac{n}{\varepsilon p} \rceil$, we have:

$$\mathbb{E}[\Phi_K] \leq \left(1 - \frac{1}{\tau} \right)^K (n - 1) \leq \varepsilon p .$$

By Markov's Inequality, with probability at least $1 - p$, it must be the case that $\Phi_K \leq \varepsilon$. $\square$

11 Solving without Low Stretch Spanning Trees

In the algorithms presented in this paper the spanning tree, T, provides two essential functions. (1) The T induces a well conditioned basis for cycle space and (2) the spanning tree allows for a compact representation of cycles that allow efficient querying and updating. It is natural to ask, can a combinatorial object other than a low-stretch spanning subgraph be used to achieve these goals? To the best of our knowledge all previous nearly-linear-time SDD-system solvers use such low-stretch spanning trees; can a nearly-linear running time be achieved without them?

All these questions can be answered in the affirmative and in fact a particular *decomposition tree* (low-stretch tree approximation of G that are not necessarily subgraphs)[Bar96, Bar98, FRT03] construction of Bartal can used precisely for this purpose. Several steps are needed. First, we need to show how decomposition trees in general can be used to satisfy property (1). Next we need to show how we can replicate edges to create an equivalent problem without hurting the property (1) too much. Finally, we need to show how this replication can be done to achieve just enough path disjointness that we can apply the data structure from Section 5 and achieve property (2). Ultimately, this can all be accomplished while only losing several log factors in the run time (as compared to the most efficient algorithm presented in this paper). *(We plan on including the details in a later version of this paper.)*

12 Acknowledgments

References

[ABN08] Ittai Abraham, Yair Bartal, and Ofer Neiman. Nearly tight low stretch spanning trees. *CoRR*, abs/0808.2017, 2008. 2, 7

[AKPW95] Noga Alon, Richard M. Karp, David Peleg, and Douglas West. A graph-theoretic game and its application to the k-server problem. *SIAM J. Comput.*, 24:78–100, February 1995. 2, 7

[AN12] Ittai Abraham and Ofer Neiman. Using petal-decompositions to build a low stretch spanning tree. In *Proceedings of the 44th symposium on Theory of Computing*, STOC '12, pages 395–406, New York, NY, USA, 2012. ACM. 2, 3, 7

[Bar96] Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *In 37th Annual Symposium on Foundations of Computer Science*, pages 184–193, 1996. 27

[Bar98] Yair Bartal. On approximating arbitrary metrices by tree metrics. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98, pages 161–168, New York, NY, USA, 1998. ACM. 27

[BCHT04] E.G. Boman, D. Chen, B. Hendrickson, and S. Toledo. Maximum-weight-basis preconditioners. *Numerical linear algebra with applications*, 11(8-9):695–721, 2004. 2

[BGH+06] M. Bern, J. Gilbert, B. Hendrickson, N. Nguyen, and S. Toledo. Support-graph preconditioners. *SIAM Journal on Matrix Analysis and Applications*, 27(4):930–951, 2006. 2

[BH03] Erik G. Boman and Bruce Hendrickson. Support theory for preconditioning. *SIAM J. Matrix Anal. Appl.*, 25:694–717, March 2003. 2

[BHM00] William L. Briggs, Van Emden Henson, and Steve F. McCormick. *A multigrid tutorial: second edition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000. 2

[BHV08] E.G. Boman, B. Hendrickson, and S. Vavasis. Solving elliptic finite element systems in near-linear time with support preconditioners. *SIAM Journal on Numerical Analysis*, 46(6):3264–3284, 2008. 1

[BKMT10] Guy E. Blelloch, Ioannis Koutis, Gary L. Miller, and Kanat Tangwongsan. Hierarchical diagonal blocking and precision reduction applied to combinatorial multigrid. In *SC*, pages 1–12, 2010. 2

[Bol98]   Bela Bollobas. *Modern Graph Theory.* Springer, 1998. 6

[BV04]   Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, New York, NY, USA, 2004. 20, 21

[CKM+11]   Paul Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, STOC '11, pages 273–282, New York, NY, USA, 2011. ACM. 2

[DS08]   Samuel I. Daitch and Daniel A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pages 451–460, 2008. 2

[EEST05]   Michael Elkin, Yuval Emek, Daniel A. Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, STOC '05, pages 494–503, 2005. 2, 7

[ER11]   Ren Escalante and Marcos Raydan. *Alternating Projection Methods.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2011. 20, 21

[FRT03]   Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, STOC '03, pages 448–455, New York, NY, USA, 2003. ACM. 27

[GMZ95]   Keith D. Gremban, Gary L. Miller, and Marco Zagha. Performance evaluation of a new parallel preconditioner. In *IPPS*, pages 65–69, 1995. 2

[Gre96]   Keith Gremban. *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems.* PhD thesis, Carnegie Mellon University, Pittsburgh, October 1996. CMU CS Tech Report CMU-CS-96-123. 31

[Her09]   Gabor T. Herman. *Fundamentals of Computerized Tomography: Image Reconstruction from Projections.* Springer Publishing Company, Incorporated, 2nd edition, 2009. 21

[HK11]   Luqman Hodgkinson and Richard Karp. Algorithms to detect multiprotein modularity conserved during evolution. In Jianer Chen, Jianxin Wang, and Alexander Zelikovsky, editors, *Bioinformatics Research and Applications*, volume 6674 of *Lecture Notes in Computer Science*, pages 111–122. Springer Berlin / Heidelberg, 2011. 1

[Jor69]   C. Jordan. Sur les assemblages de lignes. *J. Reine Angew Math*, 70:185–190, 1869. 11

[Kac37]   Stefan Kaczmarz. Angenäherte auflösung von systemen linearer gleichungen. *Bull. Internat. Acad. Polon.Sci. Lettres A*, page 335357, 1937. 4, 20, 21

[KM09]   Jonathan A. Kelner and Aleksander Madry. Faster generation of random spanning trees. In *Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '09, pages 13–21, Washington, DC, USA, 2009. IEEE Computer Society. 2

[KM11]   Jonathan Kelner and Petar Maymounkov. Electric routing and concurrent flow cutting. *Theor. Comput. Sci.*, 412(32):4123–4135, July 2011. 2

[KMP10]   Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving SDD systems. In *Proceedings of the 51st Annual Symposium on Foundations of Computer Science*, 2010. 2, 7, 15

[KMP11]   Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-m log n time solver for sdd linear systems. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 590 –598, oct. 2011. 2, 3, 7, 15

[KMP12]   Jonathan A. Kelner, Gary L. Miller, and Richard Peng. Faster approximate multicommodity flow using quadratically coupled flows. In *Proceedings of the 44th symposium on Theory of Computing*, STOC '12, pages 1–18, New York, NY, USA, 2012. ACM. 2

[KMT11]   Ioannis Koutis, Gary L. Miller, and David Tolliver. Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing. *Computer Vision and Image Understanding*, 115(12):1638–1646, 2011. 1

[LLB⁺09] Chung-Shou Liao, Kanghao Lu, Michael Baym, Rohit Singh, and Bonnie Berger. Isorankn: spectral methods for global alignment of multiple protein networks. *Bioinformatics*, 25(12):i253–i258, 2009. 1

[LLDM08] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Statistical properties of community structure in large social and information networks. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 695–704, New York, NY, USA, 2008. ACM. 1

[Nat01]   Frank Natterer. *The mathematics of computerized tomography.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001. 20, 21

[OSV12]   Lorenzo Orecchia, Sushant Sachdeva, and Nisheeth K. Vishnoi. Approximating the exponential, the lanczos method and an $\tilde{O}(m)$-time spectral algorithm for balanced separator. In *Proceedings of the 44th symposium on Theory of Computing*, STOC '12, pages 1141–1160, New York, NY, USA, 2012. ACM. 2

[She09]   Jonah Sherman. Breaking the multicommodity flow barrier for $O\left(\sqrt{\log n}\right)$-approximations to sparsest cut. In *Proceedings of the 50th Annual Symposium on Foundations of Computer Science*, 2009. 2

[Spi12]   Daniel A. Spielman. Algorithms, graph theory, and the solution of laplacian linear equations. In *Proceedings of the 39th international colloquium conference on Automata, Languages, and Programming - Volume Part II*, ICALP'12, pages 24–26, Berlin, Heidelberg, 2012. Springer-Verlag. 2

[SS11]   D.A. Spielman and N. Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011. 2, 24

[ST83]   Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. 11

[ST04]   Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 81–90, New York, NY, USA, 2004. ACM. 1, 2, 7

[ST08a]   Daniel A. Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly-linear time graph partitioning. *CoRR*, abs/0809.3232, 2008. 2

[ST08b]   Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *CoRR*, abs/0808.4134, 2008. 2

[ST12]   Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *CoRR*, abs/cs/0607105v5, 2012. 2, 3, 7, 13, 24, 31

[SV09]   Thomas Strohmer and Roman Vershynin. A randomized kaczmarz algorithm with exponential convergence. *Journal of Fourier Analysis and Applications*, 15:262–278, 2009. 4, 20

[Tar79]   Robert Endre Tarjan. Applications of path compression on balanced trees. *J. ACM*, 26(4):690–715, October 1979. 14

[Ten10]   Shang-Hua Teng. The laplacian paradigm: Emerging algorithms for massive graphs. In *TAMC*, pages 2–14, 2010. 2

[Vai90]   Pravin M. Vaidya. Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. Unpublished manuscript UIUC 1990. A talk based on the manuscript was presented at the IMA Workshop on Graph Theory and Sparse Matrix Computation, October 1991, Minneapolis., 1990. 2

[Vis]   Nisheeth Vishnoi. *Lx = b*. Monograph, available at http://research.microsoft.com/en-us/um/people/nvishno/site/Lxb-Web.pdf. 2

[VTX09]   Konstantin Voevodski, Shang-Hua Teng, and Yu Xia. Finding local communities in protein networks. *BMC Bioinformatics*, 10(1):297, 2009. 1

[Wil12]   Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *STOC*, pages 887–898, 2012. 1

A   Reduction from SDD Systems to Laplacian Systems

Reductions from SDD system solving to Laplacian system solving can be traced back to [Gre96] where the problem of solving a linear system in a *symmetric diagonally dominant* (SDD) matrix of dimension n was reduced to that of solving a Laplacian matrix of dimension $2n + 1$. This reduction contains two steps. First, a SDD matrix is reduced to a SDDM matrix of dimension $2n$, where a SDDM matrix is an SDD matrix with non-positive off-diagonals.[11]. Second, this SDDM matrix is augmented with an extra dimension yielding a Laplacian matrix of dimension $2n + 1$. Following the interpretation of a Laplacian system as an electrical circuit system (see Section 2.1.1) this second step can also be viewed as adding an extra ground vertex with fixed zero voltage, and connecting it to all other vertices whose corresponding rows do not sum up to zero.

The first step of the above reduction is often referred to as *Gremban's reduction*, and has been used in previous nearly-linear-time SDD system solvers (see for instance [ST12]). Although this reduction is stated for exact solutions, it also carries approximate solutions from one to the other [ST12].

Below we present a variant of Gremban's reduction, that in a single step reduces from a linear system of a SDD matrix directly to that of a Laplacian matrix and we prove that this reduction preserves approximate solutions. Since for Laplacians of disconnected graphs each component can be solved independently we see that it suffices to study Laplacian matrices for connected graphs in the main body of our paper.

A.1   Our Reduction

Given an arbitrary SDD matrix $\mathbf{A}$, we can always decompose it into $\mathbf{A} = \mathbf{D}_1 + \mathbf{A}_p + \mathbf{A}_n + \mathbf{D}_2$, where
 - $\mathbf{A}_p$ is the matrix containing all positive off-diagonal entries of $\mathbf{A}$,
 - $\mathbf{A}_n$ is the matrix containing all negative off-diagonal entries of $\mathbf{A}$,
 - $\mathbf{D}_1$ is the diagonal matrix where $\mathbf{D}_1(i, i) \stackrel{\text{def}}{=} \sum_{j=1}^{n} |\mathbf{A}(i, j)|$, and
 - $\mathbf{D}_2 = \mathbf{A} - \mathbf{A}_p - \mathbf{A}_n - \mathbf{D}_1$ is the excess diagonal matrix.

Given that $\mathbf{A}$ is SDD it is a simple exercise to check that $\mathbf{D}_1 + \mathbf{A}_n - \mathbf{A}_p$ is Laplacian and the entries of $\mathbf{D}_2$ are non-negative. Now, consider the following linear system

$$\boxed{\tilde{\mathbf{A}} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b \\ -b \end{pmatrix}} \quad \text{where } \tilde{\mathbf{A}} = \begin{pmatrix} \mathbf{D}_1 + \mathbf{D}_2/2 + \mathbf{A}_n & -\mathbf{D}_2/2 - \mathbf{A}_p \\ -\mathbf{D}_2/2 - \mathbf{A}_p & \mathbf{D}_1 + \mathbf{D}_2/2 + \mathbf{A}_n \end{pmatrix} \ .$$

It is not hard to verify that $\tilde{\mathbf{A}}$ is Laplacian and that given an exact solution (x_1, x_2) to the above linear system (and notice that there may be many such solutions) we have

$$\left. \begin{array}{l} (\mathbf{D}_1 + \mathbf{D}_2/2 + \mathbf{A}_n)x_1 + (-\mathbf{D}_2/2 - \mathbf{A}_p)x_2 = b \\ (-\mathbf{D}_2/2 - \mathbf{A}_p)x_1 + (\mathbf{D}_1 + \mathbf{D}_2/2 + \mathbf{A}_n)x_2 = -b \end{array} \right\} \text{ implies } \mathbf{A}\left(\frac{x_1 - x_2}{2} \right) = b \ .$$

So $x = \frac{x_1 - x_2}{2}$ is also an exact solution to the original linear system $\mathbf{A}x = b$.

Now let us verify that this reduction also holds for approximate solutions. Let $(\hat{x}_1, \hat{x}_2)$ denote an arbitrary ε-approximate solution to the new Laplacian system, and let $\hat{e}_1 \stackrel{\text{def}}{=} \hat{x}_1 - x_1$ and $\hat{e}_2 \stackrel{\text{def}}{=} \hat{x}_2 - x_2$. By definition we have:

$$\left\| \begin{pmatrix} \hat{e}_1 \\ \hat{e}_2 \end{pmatrix} \right\|_{\tilde{\mathbf{A}}} = \left\| \begin{pmatrix} \hat{x}_1 \\ \hat{x}_2 \end{pmatrix} - \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \right\|_{\tilde{\mathbf{A}}} \leq \varepsilon \left\| \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \right\|_{\tilde{\mathbf{A}}}$$

and by expanding it out we have:

$$\begin{pmatrix} \hat{e}_1 \\ \hat{e}_2 \end{pmatrix}^T \tilde{\mathbf{A}} \begin{pmatrix} \hat{e}_1 \\ \hat{e}_2 \end{pmatrix} \leq \varepsilon^2 \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}^T \tilde{\mathbf{A}} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$\Leftrightarrow \quad \begin{pmatrix} \hat{e}_1 \\ \hat{e}_2 \end{pmatrix}^T \begin{pmatrix} \mathbf{D}_1 + \mathbf{D}_2/2 + \mathbf{A}_n & -\mathbf{D}_2/2 - \mathbf{A}_p \\ -\mathbf{D}_2/2 - \mathbf{A}_p & \mathbf{D}_1 + \mathbf{D}_2/2 + \mathbf{A}_n \end{pmatrix} \begin{pmatrix} \hat{e}_1 \\ \hat{e}_2 \end{pmatrix} \leq \varepsilon^2 \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}^T \begin{pmatrix} b \\ -b \end{pmatrix} = 2\varepsilon^2 x^T b \qquad (13)$$

---

[11] An SDDM matrix is not necessarily Laplacian since its rows (or columns) may not sum up to zeros. For historical reasons, Gremban [Gre96] uses the notion "generalized Laplacian matrix" for SDD matrix, and the notion "Laplacian matrix" for SDDM matrix.

Now, because $\mathbf{D}_1 + \mathbf{A}_n - \mathbf{A}_p$ is Lapacian (and therefore positive semidefinite) we have

$$\begin{pmatrix} \mathbf{D}_1 + \mathbf{A}_n - \mathbf{A}_p & \mathbf{D}_1 + \mathbf{A}_n - \mathbf{A}_p \\ \mathbf{D}_1 + \mathbf{A}_n - \mathbf{A}_p & \mathbf{D}_1 + \mathbf{A}_n - \mathbf{A}_p \end{pmatrix} \succeq 0$$

$$\Leftrightarrow \qquad \begin{pmatrix} \mathbf{D}_1 + \mathbf{A}_n - \mathbf{A}_p & \mathbf{D}_1 + \mathbf{A}_n - \mathbf{A}_p \\ \mathbf{D}_1 + \mathbf{A}_n - \mathbf{A}_p & \mathbf{D}_1 + \mathbf{A}_n - \mathbf{A}_p \end{pmatrix} + \begin{pmatrix} \mathbf{A} & -\mathbf{A} \\ -\mathbf{A} & \mathbf{A} \end{pmatrix} \succeq \begin{pmatrix} \mathbf{A} & -\mathbf{A} \\ -\mathbf{A} & \mathbf{A} \end{pmatrix}$$

$$\Leftrightarrow \qquad 2\begin{pmatrix} \mathbf{D}_1 + \mathbf{D}_2/2 + \mathbf{A}_n & -\mathbf{D}_2/2 - \mathbf{A}_p \\ -\mathbf{D}_2/2 - \mathbf{A}_p & \mathbf{D}_1 + \mathbf{D}_2/2 + \mathbf{A}_n \end{pmatrix} \succeq \begin{pmatrix} \mathbf{A} & -\mathbf{A} \\ -\mathbf{A} & \mathbf{A} \end{pmatrix}$$

and as a consequence we deduce from (13) that (by defining $\hat{x} = \frac{\hat{x}_1 - \hat{x}_2}{2}$)

$$\frac{1}{2}\begin{pmatrix} \hat{e}_1 \\ \hat{e}_2 \end{pmatrix}^T \begin{pmatrix} \mathbf{A} & -\mathbf{A} \\ -\mathbf{A} & \mathbf{A} \end{pmatrix} \begin{pmatrix} \hat{e}_1 \\ \hat{e}_2 \end{pmatrix} \le 2\varepsilon^2 x^T b \quad \Leftrightarrow \quad 2\left(\frac{\hat{e}_1 - \hat{e}_2}{2}\right)^T \mathbf{A} \left(\frac{\hat{e}_1 - \hat{e}_2}{2}\right) \le 2\varepsilon^2 x^T b$$

$$\Leftrightarrow \quad 2(\hat{x} - x)^T \mathbf{A}(\hat{x} - x) \le 2\varepsilon^2 x^T b$$

$$\Leftrightarrow \quad \|\hat{x} - x\|_{\mathbf{A}} \le \varepsilon \|x\|_{\mathbf{A}} \ .$$

Therefore, we have shown that $\hat{x} = \frac{\hat{x}_1 - \hat{x}_2}{2}$ is also an ε approximate solution to $\mathbf{A}x = b$.