CSC 2240H : Graphs, Matrices, and Optimization 23 Dec 2022

Solving Laplacians using low-stretch spanning trees

Ensieh Khazaei, Danya Lette

1 Introduction

In their 2013 paper "A Simple, Combinatorial Algorithm for Solving SDD Systems in Nearly-Linear Time" [3], Kelner et al. proposed an algorithm for getting approximate solutions to SDD systems, that is: systems of the form $A\vec{x} = \vec{b}$, in which A is symmetric and diagonally dominant.

Prior to this work, in 2004, Spielman and Teng proposed their own nearly-linear time Laplacian solver [4]. This highly influential work is also highly complex, making it difficult to understand, analyze, and adapt to special cases. The algorithm in [3] is, in contrast, very simple.

In brief, the algorithm reduces SDD systems to Laplacian systems and, in the context of the electrical networks view of a Laplacian system, aims to find an approximately optimal flow vector by iteratively updating a guess of the network's flow. In each iteration, an edge is sampled from a spanning tree of the network; each edge corresponds to a cycle in the network, and the flow is updated in that iteration by enforcing a conservation of energy law on that cycle. The algorithm runs in $\mathcal{O}(m \log^2 n \log\log n \log(\epsilon^{-1}))$.

In these lecture notes, we present the necessary background materials in Section 2, the algorithm itself in Section 3. We then discuss the algorithm's correctness and convergence in Section 4 and, finally, present a geometrical interpretation in Section 5.

2 Preliminaries

2.1 The Problem

An SDD system is a system of the form $A\vec{x} = \vec{b}$, where A is *symmetric* ($A^T = A$) and A is *diagonally dominant* ($\forall i \in [n].A_{ii} \geq \sum_{j \neq i} |A_{ij}|$). The Kelner paper [3] is aimed at **approximately** solving an SDD system. In particular, if $A\vec{x}_{opt} = \vec{b}$, then $\vec{x}$ is an approximate solution whenever

$$\|\vec{x} - \vec{x}_{opt}\|_A \leq \epsilon \|\vec{x}_{opt}\|_A$$

The authors present a reduction from the SDD system $A\vec{x} = \vec{b}$ to the Laplacian system $L\vec{v} = \vec{\chi}$ satisfying $\sum_{a \in V} \vec{\chi}(a) = 0$, and their subsequent technique pertains to Laplacian solving. In these lecture notes, we omit the reduction and focus on the Laplacian solving, in keeping with the themes of this course.

If $L^\dagger \vec{\chi} = \vec{v}_{opt}$, where $L^\dagger$ is the Moore-Penrose pseudoinverse of L, then an approximate solution $\vec{v}$ to this Laplacian refers to a vector $\vec{v}$ satisfying

$$\mathbb{E}\|\vec{v} - \vec{v}_{opt}\|_L \leq \sqrt{\epsilon} \|\vec{v}_{opt}\|_L$$

2.2 Electrical Networks

We define electrical network graphs in the usual way. Let $G = (V, E, w)$ be a weighted, connected, undirected graph with $n = |V|$ vertices and $m = |E|$ edges, where edge $e \in E$ has weight w_e and *resistance* $r_e = \frac{1}{w_e}$. Edges are oriented by the convention that, for the edge incident on $a, b \in V$, one of (a, b) or (b, a) are $\in E$. Define the incidence matrix $B \in \mathbb{R}^{E \times V}$, the resistance matrix $R \in \mathbb{R}^{E \times E}$, and the Laplacian matrix $L \in \mathbb{R}^{V \times V}$:

$$B_{(a,b),c} = \begin{cases} 1 & a = c \\ -1 & b = c \\ 0 & \text{o.w.} \end{cases} \qquad R_{e_1,e_2} = \begin{cases} r_e & e = e_1 = e_2 \\ 0 & \text{o.w.} \end{cases} \qquad L_{a,b} = \begin{cases} \text{wdeg}(a) & a = b \\ -w_{a,b} & \{a, b\} \in E \\ 0 & \text{o.w.} \end{cases}$$

2.3 Electrical Flow and Duality

Definition 2.1 (Feasible)**.** A flow $\vec{f} \in \mathbb{R}^E$ is *feasible* with respect to $\vec{\chi} \in \mathbb{R}^V$ if $B^T \vec{f} = \vec{\chi}$

For $\vec{f} \in \mathbb{R}^E$, define its energy as $\xi(\vec{f}) = \vec{f}^T R \vec{f}$. Given a demand vector $\vec{\chi}$ such that $\sum_{a \in V} \vec{\chi}(a) = 0$, the *electrical flow problem* is the problem of finding the energy-minimizing feasible flow:

$$\vec{f}_{opt} = \text{argmin}_{\vec{f} \in \mathbb{R}^E : B^T \vec{f} = \vec{\chi}} \xi(\vec{f})$$

In the context of electrical networks, a solution to the system $L\vec{x} = \vec{\chi}$ concerns a *voltage* vector $\vec{x} \in \mathbb{R}^V$. However, the algorithm that we will present today searches for a *flow* vector $\vec{f} \in \mathbb{R}^E$. These two vectors are related via *duality*.

The electrical flow problem is the Lagrangian dual of solving $L\vec{x} = \vec{\chi}$, which finds the energy-maximizing voltage $\vec{x}$, where, for $\vec{v} \in \mathbb{R}^V$, voltage energy is defined as $\zeta(\vec{v}) = 2\vec{v}\vec{\chi} - \vec{v}^T L \vec{v}$.

This is a strong duality, so $\zeta(\vec{v}_{opt}) = \xi(\vec{f}_{opt})$. The optimal flow and voltage vectors $\vec{f}_{opt}$ and $\vec{v}_{opt}$ satisfy

$$\forall (a, b) \in E : \vec{f}_{opt}((a, b)) = \frac{\vec{v}_{opt}(a) - \vec{v}_{opt}(b)}{r_{(a,b)}}$$

Definition 2.2 (Circulation)**.** A flow $\vec{f} \in \mathbb{R}^E$ is a *circulation* if $B^T \vec{f} = 0$

Kirchoff's Potential Law (KPL) is one form of a conservation of energy law. This law forms the basis for the algorithm.

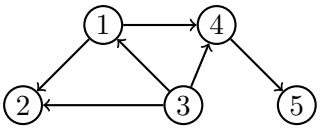
Lemma 2.3 (KPL)**.** *A feasible flow $\vec{f}$ is optimal if and only if $\vec{f}^T R \vec{c} = 0$ for all circulations $\vec{c} \in \mathbb{R}^E$.*

2.4 Low-Stretch Spanning Trees

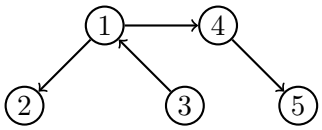
A tree is a connected, acyclic, undirected graph. Given a graph $G = (V, E, w)$, a spanning tree $T = (V_T, E_T)$ of G is a tree where $V_T = V$ and $E_T \subseteq E$. We call an edge $e \in E \setminus E_T$ an *off-tree edge*.

Remark that $E_T \cup \{e\}$, where $e = (a, b)$ is a off-tree edge, contains exactly one cycle, since there must already be a (unique) path between a and b in T. Also note that, for e, e' off-tree edges, if

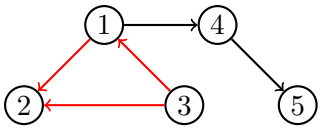
$e \neq e'$, the cycle in $E_T \cup \{e\}$ is distinct from the cycle in $E_T \cup \{e'\}$, since the former cycle contains $e \notin E_T$ and latter e'.



A graph $G = (V, E, w)$.



A spanning tree of G.



Edge $(3, 2)$'s cycle, shown in red.

Figure 1: An example graph and spanning tree with off-tree edges $(3, 2)$ and $(3, 4)$

Definition 2.4. (Tree Path) For $a, b \in V$, the tree path $P_{(a,b)} \subseteq V \times V$ is the unique path between a and b in T.

Definition 2.5. (Tree Cycle) For $a, b \in V$, the tree cycle $C_{(a,b)} = \{(a, b)\} \cup P_{(a,b)}$. The vector $\vec{c}_{(a,b)} \in \mathbb{R}^E$ assigns flow 1 to each edge $e \in C_{(a,b)}$.

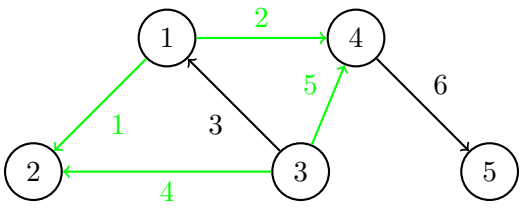
A path P_e corresponds to a vector $\vec{p}_e$ as follows: for $a, b \in V$,

$$\vec{p}_e((a, b)) = \begin{cases} 1 & (a, b) \in P_e \cap E \\ -1 & (a, b) \in P_e \wedge (b, a) \in E \\ 0 & \text{o.w.} \end{cases}$$

A cycle C_e corresponds to a vector $\vec{c}_e$, which is defined analogously to $\vec{p}_e$. The vector corresponding to a cycle is a circulation, which, recall, means that $B^T \vec{c}_e = 0$.

The set of all circulations $\{\vec{c} \in \mathbb{R}^E : B^T \vec{c} = 0\}$ is a subspace called the *cycle space* and the cycles $\{\vec{c}_e : e \in E \setminus E_T\}$ determined by the off-tree edges are a basis for this space.

Example 2.6. Continuing the example of Figure 1, the cycle basis is $\{\vec{c}_{(3,2)}, \vec{c}_{(3,4)}\}$. The cycle C' with path $\{(3, 2), (2, 1), (1, 4), (4, 3)\}$ and vector $\vec{c}'$ is not in this basis, but it can be formed from basis elements.



$$\vec{c}_{(3,2)} = \begin{bmatrix} -1 \\ 0 \\ -1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \vec{c}_{(3,4)} = \begin{bmatrix} 0 \\ -1 \\ -1 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \vec{c}' = \begin{bmatrix} -1 \\ 1 \\ 0 \\ 1 \\ -1 \\ 0 \end{bmatrix} = \vec{c}_{(3,2)} - \vec{c}_{(3,4)}$$

Figure 2: The composite cycle C', shown in green, is comprised of basis cycles.

A given graph may have several spanning trees, some of which result in faster convergence of the algorithm than others. We quantify the desirability of a spanning tree according to its *condition number*. This number represents how well the resistances of off-tree edges are approximated by the resistance along the cycle determined by that edge.

Definition 2.7 (Cycle Resistance R_e)**.** $R_e = \sum_{e' \in C_e} r_{e'} = \vec{c}_e^T R \vec{c}_e$

Definition 2.8 (Tree Condition Number τ)**.** $\tau(T) = \sum_{e\ inE\setminus E_T} \frac{R_e}{r_e}$.

Definition 2.9 (Stretch st)**.** For $e \in E$, $st(e) = \frac{\sum_{e' \in P_e} r_{e'}}{r_e}$ and $st(T) = \sum_{e \in E} st(e)$.

We remark that $\tau(T) = st(T) + m - 2n + 2$; so, a low-stretch tree has a low condition number. Other works have yielded useful techniques for producing low-stretch spanning trees.

Theorem 2.10 ([1])**.** *In $\mathcal{O}(m \log n \log\log n)$ time we can compute a spanning tree T with a total stretch $st(T) = O(m \log n \log\log n)$.*

3 The Algorithm

In this section, we present a simple version of the algorithm presented in [3], called `SimpleSolver`.

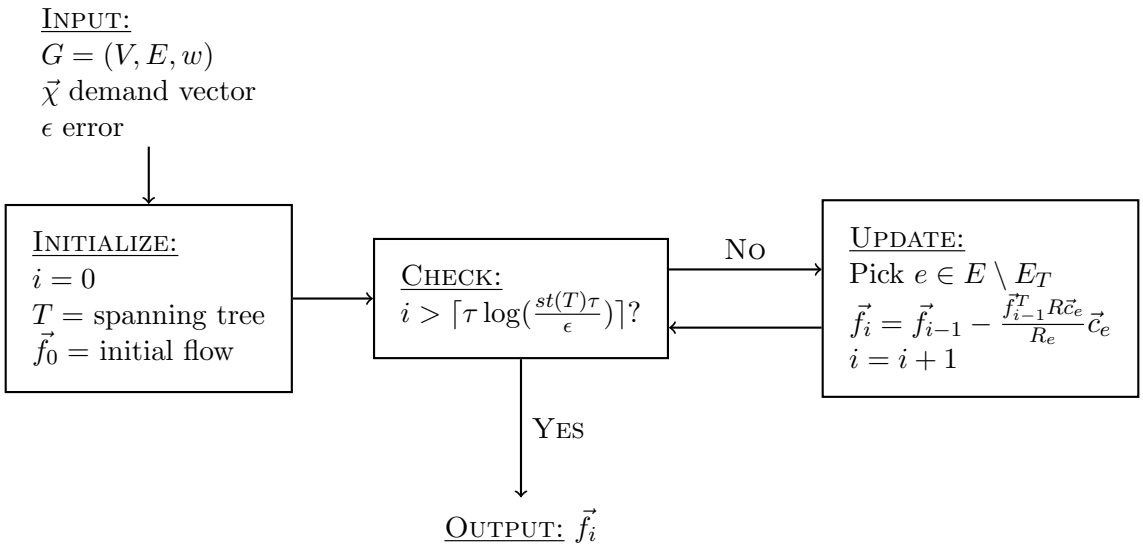


Figure 3: An outline of the `SimpleSolver` algorithm of [3]

3.1 Initialization

From Theorem 2.10, the algorithm starts by choosing a spanning tree with stretch in $\mathcal{O}(m \log n \log\log n)$ in time $\mathcal{O}(m \log n \log\log n)$.

An initial flow $\vec{f_0}$ is computed; this flow should be feasible, that is, satisfy $B^T \vec{f} = \vec{\chi}$. A feasible flow may be found in $\mathcal{O}(m+n)$ using Depth-First Search (DFS). Recalling that $\sum_{a \in V} \vec{\chi}(a) = 0$, we can make the reasonable simplifying assumption that, for some $s, t \in V$, $\vec{\chi} = \mathbf{1}_s - \mathbf{1}_t$. $P_{(s,t)} \subseteq V \times V$ is the path between s and t in T. Now consider the flow $\vec{f_0}$ defined as

$$\vec{f_0}(a) = \begin{cases} 1 & a \in P_{(s,t)} \\ 0 & \text{o.w.} \end{cases}$$

By definition of the incidence matrix B, for each edge $e = (a, b) \in E$, the eth row of B is $\mathbf{1}_a^T - \mathbf{1}_b^T$.

So,

$$B^T \vec{f_0} = \sum_{(a,b)\in P_{(s,t)}} \mathbf{1}_a - \mathbf{1}_b = \vec{\chi},$$

. which is feasible. So, a feasible initial flow $\vec{f_0}$ is computed in linear time using DFS.

3.2 Cycle Update

In order to choose a cycle in the cycle basis, an edge is selected from $E \setminus E_T$. These edges are sampled with probability distribution $P(e) = \frac{1}{\tau(T)} \cdot \frac{R_e}{r_e}$. Once an edge e is selected, the current flow $\vec{f}_{i-1}$ is updated:

$$\vec{f_i} = \vec{f}_{i-1} - \frac{\vec{f}_{i-1}^T R \vec{c_e}}{R_e} \vec{c_e}$$

The new flow satisfies KPL on the cycle C_e, that is: $\vec{f_i}^T R \vec{c_e} = 0$, since $\vec{c_e}^T R \vec{c_e} = R_e$. In addition, because $\vec{c_e}$ is a circulation, $B^T \vec{c_e} = 0$, which implies that the new flow is feasible, meaning $B^T \vec{f_i} = \vec{\chi}$.

3.3 Runtime

Initialization runs in $\mathcal{O}(m \log n \log\log n)$. Cycle updates, which are repeated $\lceil \tau \log(\frac{st(T)\tau}{\epsilon}) \rceil$ times, can be performed in $\mathcal{O}(\log n)$, using a data structure presented in [3] which is out of scope of these lecture notes. In all, algorithm runs in $\mathcal{O}(m \log^2 n \log\log n \log(\epsilon^{-1}))$.

4 Convergence Analysis

In this section we analyze the convergence rate of `SimpleSolver` algorithm. In other words, we investigate whether this algorithm can converge to the optimal solution after finite iterations or not. For this purpose, we should show that after finite iterations, the algorithm's solution approach to the optimal solution with ratio ϵ or mathematically, the following equation should be satisfied. For $\forall \;\; 0 < \epsilon < 1, \exists \; i$ that the Equation 1 is satisfied.

$$\|X_i - X^*\|_2 \leq \epsilon \|X_0 - X^*\|_2 \tag{1}$$

where X_i and X_0 are the algorithm's solution in the i-th and first iterations, respectively. ϵ is lower than 1, but our desired value is very close to zero like 0.001 or 10^{-5}. When we can find an i such that the Equation 1 is established in that iteration, then we can conclude the algorithm can converge to the optimal solution with ratio ϵ. Theorem 4.1 is equivalent to Equation 1 in the context of `SimpleSolver` algorithm.

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

$$\mathbb{E}[\xi_r(\vec{f_i})] - \xi_r(\vec{f_{opt}}) \leq (1 - \tfrac{1}{\tau})^i (\xi_r(\vec{f_0}) - \xi_r(\vec{f_{opt}}))$$

where $\xi_r(\vec{f_i})$, $\xi_r(\vec{f_{opt}})$, and $\xi_r(\vec{f_0})$ are the energy of i-th iteration, optimal energy, and energy of primary current flow, respectively. τ is the condition number of spanning tree of T which is fixed in all iterations and it is positive. Therefore, $(1 - \frac{1}{\tau})$ would be lower than 1 and as i grows, the value of $(1 - \frac{1}{\tau})^i$ becomes closer to zero. The proof of Theorem 4.1 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

In Section 3.2, we observed that the current flow vector in each iteration is updated with $-\frac{\Delta_{c_e}(\vec{f_i})}{R_e}\vec{c}_e$ vector, where $\Delta_{c_e}(\vec{f_i}) = \vec{f_i}^\top \mathbf{R}\vec{c}_e$ and $R_e = \vec{c}_e^\top \mathbf{R}\vec{c}_e$. In this section, from a mathematical perspective, we investigate why this update vector approaches the algorithm to the optimal solution.

The first idea that comes to our mind when we want to calculate the optimal update value is derivation. Therefore, we consider the $\xi_r(\vec{f}+\alpha\vec{c})$ and find the best α that can minimize the energy.

$$\frac{\partial \xi_r(\vec{f}+\alpha\vec{c})}{\partial \alpha} = \frac{\partial (\vec{f}+\alpha\vec{c})^\top \mathbf{R}(\vec{f}+\alpha\vec{c})}{\partial \alpha} = 2\vec{f}\mathbf{R}\vec{c} + 2\alpha\vec{c}^\top\mathbf{R}\vec{c}$$

$$2\vec{f}\mathbf{R}\vec{c} + 2\alpha\vec{c}^\top\mathbf{R}\vec{c} = 0 \Rightarrow \alpha^* = -\frac{\vec{f}\mathbf{R}\vec{c}}{\vec{c}^\top\mathbf{R}\vec{c}} \tag{2}$$

In the second stage, we verify whether this α^* can decrease the energy of the electrical network in each iteration or not. Therefore, we calculate the energy difference between two sequential iterations by substitution of α^*.

$$\xi_r(\vec{f}+\alpha^*\vec{c}) - \xi_r(\vec{f}) = (\vec{f}+\alpha^*\vec{c})^\top\mathbf{R}(\vec{f}+\alpha^*\vec{c}) - \vec{f}^\top\mathbf{R}\vec{f} =$$

$$\alpha^{*2}\vec{c}\mathbf{R}\vec{c} + 2\alpha^*\vec{f}^\top\mathbf{R}\vec{c} = -\frac{(\vec{f}^\top\mathbf{R}\vec{c})^2}{\vec{c}^\top\mathbf{R}\vec{c}} \tag{3}$$

According to Equation 3, it can be seen the energy in two sequential iterations decreases if we update the flow vector with the corresponding α. We summarize the update value and energy decrease in Lemma 4.2.

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

$$argmin_{\alpha\in\mathbb{R}}\ \xi_r(\vec{f}+\alpha\vec{c}) = -\frac{\vec{f}\mathbf{R}\vec{c}}{\vec{c}^\top\mathbf{R}\vec{c}} \qquad \text{and} \qquad \xi_r(\vec{f}+\alpha^*\vec{c}) - \xi_r(\vec{f}) = -\frac{(\vec{f}^\top\mathbf{R}\vec{c})^2}{\vec{c}^\top\mathbf{R}\vec{c}}$$

If we define $\vec{c} = \vec{c}_e$ is a tree cycle for some off-tree edge $e \in E \setminus T$, since $R_e = \vec{c}_e^\top\mathbf{R}\vec{c}_e$ and $\Delta_{c_e}(\vec{f}) = \vec{f}^\top\mathbf{R}\vec{c}_e$, this procedure is precisely the iterative step of `SimpleSolver`, i.e. a cycle update. Lemma 4.3 states 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. *For feasible $\vec{f} \in \mathbb{R}^E$ and $e \in E \setminus T$ we have*

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

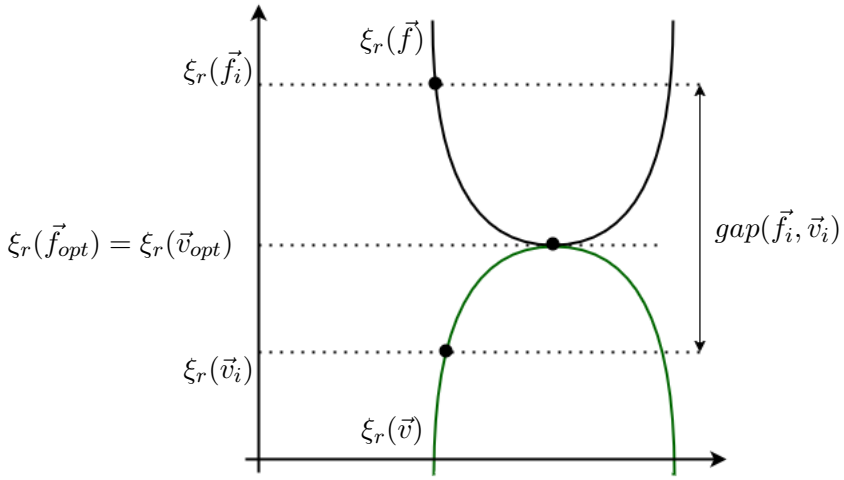


Figure 4: Visualization for primal and dual functions in electrical network energy

4.2 Distance to Optimality

In this section, we calculate the gap between $\vec{f}$ and $\vec{v}$ in the dual problem. Since the energy function of $\vec{f}$ is convex and Slater's condition is satisfied, the duality gap between the primal and dual problems is zero. Figure 4 shows the gap between the primal and dual problems.

Figure 4 makes smoother understanding the distance between $\vec{f}$, $\vec{f}_{opt}$, $\vec{v}$, and $\vec{v}_{opt}$. Here we derive a simple expression for the duality gap between $\vec{f}$ and its tree-induced voltages $\vec{v}$ in terms of cycle potentials. Lemma 4.4 states this quantity in terms of cycle potentials.

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

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

Proof. We defined the primal and dual energy in Section 2.3, therefore, we have

$$gap(\vec{f}, \vec{v}) = \vec{f}^\top \mathbf{R} \vec{f} - (2\vec{v}^\top \vec{\chi} - \vec{v}^\top \mathbf{L} \vec{v}) \tag{4}$$

If we substitute $\mathbf{B}^\top \vec{f} = \vec{\chi}$ and $\mathbf{L} = \mathbf{B}^\top \mathbf{R}^{-1} \mathbf{B}$ in Equation 4, we get

$$gap(\vec{f}, \vec{v}) = \vec{f}^\top \mathbf{R} \vec{f} - 2\vec{v}^\top \mathbf{B}^\top \vec{f} + \vec{v}^\top \mathbf{B}^\top \mathbf{R}^{-1} \mathbf{B} \vec{v}) = (\mathbf{R}\vec{f} - \mathbf{B}\vec{v})^\top \mathbf{R}^{-1} (\mathbf{R}\vec{f} - \mathbf{B}\vec{v})$$

$$= \sum_{e \in E} \frac{1}{r_e} (\vec{f}(e) r_e - \Delta_{\vec{v}}(e))^2 \tag{5}$$

In this stage, we used the definition of tree voltages in Equation 6 to simplify $gap(\vec{f}, \vec{v})$.

$$\forall\ a, b \in V \quad : \quad \Delta_{\vec{v}}(a, b) = \vec{v}(a) - \vec{v}(b) = \sum_{e \in P_{as}} \vec{f}(e) r_e + \sum_{e \in P_{sb}} \vec{f}(e) r_e = \sum_{e \in P_{ab}} \vec{f}(e) r_e \tag{6}$$

We know that

$$\vec{f}(e)r_e - \Delta_{\vec{v}}(e) = \begin{cases} 0 & \forall\ e \in T \\ \Delta_{c_e}(\vec{f}) & \forall\ e \in E \setminus T \end{cases} \tag{7}$$

is satisfied during the algorithm. Figure 5 shows an electrical network to smooth understanding of Equation 7.

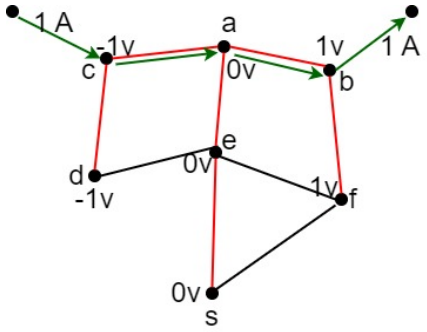


Figure 5: An example of an electrical network with induced current 1A between nodes b and c (the resistance of all edges is equal to 1Ω)

1A current is induced between nodes b and c in the electrical network in Figure 5. Assume one edge from the spanning tree, which is shown with red edges in the graph, like edge (c, a), and investigate the relation $\vec{f}(e)r_e - \Delta_{\vec{v}}(e)$ in it (the resistance of all edges in the network is equal to 1Ω)).

$$\vec{f}((c,a))r_{(c,a)} = (1A) \times (1\Omega) = 1v$$

$$\begin{gathered} \Delta_{\vec{v}}(c,a) = \vec{v}(c) - \vec{v}(a) = \textstyle\sum_{e \in P_{cs}} \vec{f}(e)r_e + \sum_{e \in P_{sa}} \vec{f}(e)r_e \\ = [\vec{f}((e,s))r_{(e,s)} + \vec{f}((a,e))r_{(a,e)} + \vec{f}((c,a))r_{(c,a)}] - [\vec{f}((e,s))r_{(e,s)} + \vec{f}((a,e))r_{(a,e)}] \\ = \vec{f}((c,a))r_{(c,a)} = (1A) \times (1\Omega) = 1v \end{gathered}$$

$$\Rightarrow \vec{f}((c,a))r_{(c,a)} - \Delta_{\vec{v}}(c,a) = 0$$

Now, consider an edge outside of the spanning tree such as (e, f) and investigate the relation $\vec{f}(e)r_e - \Delta_{\vec{v}}(e)$ for this edge.

$$\vec{f}((e,f))r_{(e,f)} = (0A) \times (1\Omega) = 0v$$

$$\begin{gathered} \Delta_{\vec{v}}(e,f) = \vec{v}(e) - \vec{v}(f) = \textstyle\sum_{e \in P_{es}} \vec{f}(e)r_e + \sum_{e \in P_{sf}} \vec{f}(e)r_e \\ = [\vec{f}((e,s))r_{(e,s)}] - [\vec{f}((e,s))r_{(e,s)} + \vec{f}((a,e))r_{(a,e)} + \vec{f}((b,a))r_{(b,a)} + \vec{f}((f,b))r_{(f,b)}] \\ = -\vec{f}((a,e))r_{(a,e)} - \vec{f}((b,a))r_{(b,a)} - \vec{f}((f,b))r_{(f,b)} \\ = -(0A) \times (1\Omega) - (1A) \times (1\Omega) - (0A) \times (1\Omega) = -1v \end{gathered}$$

$$\begin{gathered} \Rightarrow \vec{f}((e,f))r_{(e,f)} - \Delta_{\vec{v}}(e,f) = \\ \vec{f}((a,e))r_{(a,e)} + \vec{f}((b,a))r_{(b,a)} + \vec{f}((f,b))r_{(f,b)} \\ = \textstyle\sum e \in P_{ef} \vec{f}(e)r_e = \Delta_{c_{(e,f)}}(\vec{f}) \end{gathered}$$

We showed the correctness of Equation 7 with an electrical network example. Therefore, if we substitute Equation 7 in the simplified definition of $gap(\vec{f}, \vec{v})$

$$\begin{aligned} gap(\vec{f}, \vec{v}) &= \sum_{e \in E} \frac{1}{r_e} (\vec{f}(e) r_e - \Delta_{\vec{v}}(e))^2 \\ &= \sum_{e \in T} \frac{1}{r_e} (\vec{f}(e) r_e - \Delta_{\vec{v}}(e))^2 + \sum_{e \in E \setminus T} \frac{1}{r_e} (\vec{f}(e) r_e - \Delta_{\vec{v}}(e))^2 \\ &= 0 + \sum_{e \in E \setminus T} \frac{1}{r_e} \Delta_{c_e}(\vec{f})^2 = \sum_{e \in E \setminus T} \frac{1}{r_e} \Delta_{c_e}(\vec{f})^2 \end{aligned}$$

The proof in this step is complete and Lemma 4.4 is proved for each electrical network. □

4.3 Convergence Proof

In this section, we find an interpretation of energy decrease in two sequential iterations according to the duality gap in order to bound the convergence of `SimpleSolver` algorithm.
In the first step, we state the expectation of energy decrease in two sequential iterations in Lemma 4.5.

Lemma 4.5. *For iteration i of `SimpleSolver` algorithm, we have*

$$\mathbb{E}[\xi_r(\vec{f_i}) - \xi_r(\vec{f_{i-1}}) |\ gap(\vec{f_{i-1}}, \vec{v}_{i-1})] = -\frac{gap(\vec{f}_{i-1}, \vec{v}_{i-1})}{\tau}$$

Proof. In each iteration i, `SimpleSolver` algorithm picks a random $e_i \in E \setminus T$ with probability p_{e_i}. Therefore, we can calculate the expectation as follows.

$$\mathbb{E}[\xi_r(\vec{f_i}) - \xi_r(\vec{f_{i-1}}) |\ gap(\vec{f_{i-1}}, \vec{v}_{i-1})] = \sum_{e \in E} p_e [\xi_r(\vec{f_i}) - \xi_r(\vec{f_{i-1}})] \tag{8}$$

According to the `SimpleSolver` algorithm, $p_e = \frac{1}{\tau} \frac{R_e}{r_e}$ and using Lemma 4.3, the energy decreas between two sequential iterations for $\forall\ e \in E \setminus T$ is equal to $-\frac{\Delta_{c_e}(\vec{f})^2}{R_e}$ and it is zero for edges on spanning tree. By substitution of these quantities in Equation 8, we get

$$\begin{aligned} \mathbb{E}[\xi_r(\vec{f_i}) - \xi_r(\vec{f_{i-1}}) |\ gap(\vec{f_{i-1}}, \vec{v}_{i-1})] &= \sum_{e \in E} p_e [\xi_r(\vec{f_i}) - \xi_r(\vec{f_{i-1}})] \\ &= \sum_{e \in E \setminus T} \frac{1}{\tau} \frac{R_e}{r_e} (-\frac{\Delta_{c_e}(\vec{f}_{i-1})^2}{R_e}) \end{aligned} \tag{9}$$

In Lemma 4.4, we observed that $\sum_{e \in E \setminus T} \frac{\Delta_{c_e}(\vec{f})^2}{r_e} = gap(\vec{f}, \vec{v})$. By using Lemma 4.4, Equation 9 can be simplified as follows.

$$\begin{aligned} \mathbb{E}[\xi_r(\vec{f_i}) - \xi_r(\vec{f_{i-1}}) |\ gap(\vec{f_{i-1}}, \vec{v}_{i-1})] &= \sum_{e \in E \setminus T} \frac{1}{\tau} \frac{R_e}{r_e} (-\frac{\Delta_{c_e}(\vec{f}_{i-1})^2}{R_e}) \\ &= -\frac{1}{\tau} \sum_{e \in E \setminus T} \frac{\Delta_{c_e}(\vec{f}_{i-1})^2}{r_e}) = -\frac{gap(\vec{f}_{i-1}, \vec{v}_{i-1})}{\tau} \end{aligned}$$

Hence, the proof of Lemma 4.5 is finished. □
Next, we try to show that each iteration decreases the expected energy difference between the current flow and the optimal flow by a ratio $(1 - \frac{1}{\tau})$. Lemma 4.6 expresses this.

Lemma 4.6. *For all $i \geq 0$, we define a random variable $D_i \overset{def}{=} \xi_r(\vec{f_i}) - \xi_r(\vec{f_{opt}})$. Then for all iterations $i \geq 1$, we have*

$$\mathbb{E}[D_i] \leq (1 - \tfrac{1}{\tau})\mathbb{E}[D_{i-1}]$$

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 the law of total expectation we have

$$\mathbb{E}[D_i] = \sum_e \mathbb{E}[D_i | D_{i-1} = c] Pr[D_{i-1} = c] \tag{10}$$

We try to simplify $\mathbb{E}[D_i | D_{i-1} = c]$.

$$\begin{gathered}\mathbb{E}[D_i | D_{i-1} = c] = \mathbb{E}[\xi_r(\vec{f_i}) - \xi_r(\vec{f_{opt}}) | D_{i-1} = c] \\ = \mathbb{E}[(\xi_r(\vec{f_i}) - \xi_r(\vec{f_{i-1}})) + (\xi_r(\vec{f_{i-1}}) - \xi_r(\vec{f_{opt}})) | D_{i-1} = c] \\ = \mathbb{E}[(\xi_r(\vec{f_i}) - \xi_r(\vec{f_{i-1}})) + D_{i-1} | D_{i-1} = c]\end{gathered}$$

$$= c + \mathbb{E}[(\xi_r(\vec{f_i}) - \xi_r(\vec{f_{i-1}})) | D_{i-1} = c] \tag{11}$$

From Figure 4, it can be easily observed that $D_{i-1} \leq gap(\vec{f_{i-1}}, \vec{v}_{i-1})$ is always satisfied. Using this fact and Lemma 4.5, we will have

$$\mathbb{E}[(\xi_r(\vec{f_i}) - \xi_r(\vec{f_{i-1}})) | D_{i-1} = c] = -\frac{gap(\vec{f_{i-1}}, \vec{v}_{i-1})}{\tau} \leq -\frac{D_{i-1}}{\tau} = -\frac{c}{\tau} \tag{12}$$

By substitution of Equation 12 in Equation 11, we can obtain that $\mathbb{E}[D_i | D_{i-1} = c] \leq c - \frac{c}{\tau}$. Therefore,

$$\mathbb{E}[D_i] = \textstyle\sum_e \mathbb{E}[D_i | D_{i-1} = c] Pr[D_{i-1} = c] \leq (1 - \frac{1}{\tau}) \sum_e c \; Pr[D_{i-1} = c] = (1 - \tfrac{1}{\tau})\mathbb{E}[D_{i-1}].$$

□

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

$$\begin{gathered}\mathbb{E}[D_i] \leq (1 - \tfrac{1}{\tau})\mathbb{E}[D_{i-1}] \leq (1 - \tfrac{1}{\tau})^2\mathbb{E}[D_{i-2}] \leq \cdots \leq (1 - \tfrac{1}{\tau})^i\mathbb{E}[D_0] \\ \mathbb{E}[\xi_r(\vec{f_i})] - \xi_r(\vec{f_{opt}}) = \mathbb{E}[D_i] \leq (1 - \tfrac{1}{\tau})^i\mathbb{E}[D_0] = (1 - \tfrac{1}{\tau})^i[\xi_r(\vec{f_0}) - \xi_r(\vec{f_{opt}})]\end{gathered}$$

$$\Rightarrow \mathbb{E}[\xi_r(\vec{f_i})] - \xi_r(\vec{f_{opt}}) \leq (1 - \tfrac{1}{\tau})^i[\xi_r(\vec{f_0}) - \xi_r(\vec{f_{opt}})]$$

As a result, we showed that `SimpleSolver` converges to the optimal solution after finite iterations.

5 Geometric Interpretation

In this section, we present a geometric interpretation of the algorithm based on the Kaczmarz method [2]. In Section 5.1, we explain the Kaczmarz method [2] to solve linear systems. Then in Section 5.2, we provide a geometric interpretation of `SimpleSolver` algorithm based on the Kaczmarz method [2]. In particular, the `SimpleSolver` algorithm can be recast as an instance of the randomized Kaczmarz method of Strohmer and Vershynin [5].

5.1 Kaczmarz Method

The original Kaczmarz algorithm is an iterative algorithm that solves the linear system $Ax = b$, where $\mathbf{A} \in \mathbb{R}^{n \times d}$, $x \in \mathbb{R}^d$, $b \in \mathbb{R}^n$. If we find out a vector x which satisfies $\mathbf{A}x = b$, this vector will simultaneously satisfy all the following constraints.

$$\begin{cases} a_1 x = b_1 \\ a_2 x = b_2 \\ \vdots \\ a_n x = b_n \end{cases} \tag{13}$$

where a_i is the i-th row of matrix $\mathbf{A}$. Each constraint can be shown with a hyperplane in $\mathbb{R}^d$ space and the solution for the linear system should be in the intersection of these hyperplanes.

$$\begin{cases} H_1 = \{x : a_1 x = b_1\} \\ H_2 = \{x : a_2 x = b_2\} \\ \vdots \\ H_n = \{x : a_n x = b_n\} \end{cases} \tag{14}$$

In the Kaczmarz method, we initialize from a random point in $\mathbb{R}^d$ space, and in each iteration, we project the point on one of the hyperplanes. Algorithm 1 summarizes the Kaczmarz method.

Algorithm 1 Kaczmarz Algorithm for linear system $\mathbf{A}x = b$

Require: x_0

1: **while** $k \leq C$ **do**

2: $\quad x_{k+1} = x_k + \frac{b_{i(k)} - <a_{i(k)}, x_k>}{\|a_{i(k)}\|^2} a_{i(k)}$

3: $\quad i(k) = (k \mod n) + 1$

4: $\quad k = k + 1$

Return x_{C+1}

Figure 6 shows eight iteration of Kaczmarz algorithm with $\mathbf{A} \in \mathbb{R}^{4 \times 2}$. Therefore, we will have four hyperplanes in two dimensional space. As can be seen in Figure 6, we start from a random point x_0, project it on the first hyperplane H_1, and obtain x_1. In the second iteration, we project x_0 to the second hyperplane H_2 and achieve x_2. We continue this method until we find out the optimal solution x^* or we become very close to the optimal solution.
From a mathematical perspective, it can be shown that the Kaczmarz algorithm can approach the optimal solution very fast. Equation 15 states that in each iteration we approach the optimal solution x^* more than $\|x_{t+1} - x_t\|$.

$$\forall x^* \in \cap_{i=1}^{n} H_i, \qquad \|x_{t+1} - x^*\|^2 \leq \|x_t - x^*\|^2 - \|x_{t+1} - x_t\|^2 \tag{15}$$

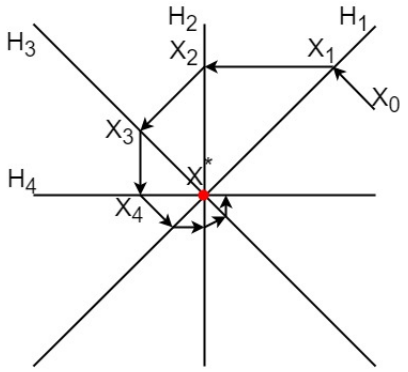


Figure 6: An example of Kaczmarz method for $\mathbf{A} \in \mathbb{R}^{4\times 2}$

5.2 Geometric View

Given the low-stretch spanning tree T, we define the cycle basis of G as $\{\vec{c}_e\}_{e\in E\setminus T}$. The cycle basis includes independent cycles, which correspond to each edge of the graph outside of the spanning tree. In other words, each cycle has just one edge outside of the tree and other edges are on the spanning tree.

Using the cycle basis $\{\vec{c}_e\}_{e\in E\setminus T}$, we can define the hyperplanes $P_e \stackrel{\text{def}}{=} \{\vec{f} \in \mathbb{R}^E : \vec{f}^\top R\vec{c_e} = 0\}$ that respect 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, the `SimpleSolver` 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 as follows.

$$\Pi_{e_i}\vec{f_i} = (I - \frac{\vec{c_{e_i}}\vec{c_{e_i}}^\top R}{\|\vec{c_{e_i}}\|_R^2})\vec{f_i} = \vec{f_i} - \frac{\vec{f_i}^\top R\vec{c_{e_i}}}{\|\vec{c_{e_i}}\|_R^2}\vec{c_{e_i}} = \vec{f_{i+1}} \tag{16}$$

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

Generally speaking, given an inner product $\langle ., . \rangle$, the orthogonal projection of x along y (resp. $y^\perp$) in that inner product space is $Proj_y(x) = \frac{\langle y,x\rangle}{\langle y,y\rangle}y$ (resp. $Proj_{y^\perp}(x) = x - \frac{\langle y,x\rangle}{\langle y,y\rangle}y$). Noting that $\langle \vec{a}, \vec{b}\rangle = \vec{a}^T R\vec{b}$ is an inner product, denote S the inner product space with this inner product. Since $\|\vec{a}\|_R^2 = \langle \vec{a}, \vec{a}\rangle$, we can see that the above expression $\vec{f_i} - \frac{\vec{f_i}^T R\vec{c_{e_i}}}{\|\vec{c_{e_i}}\|_R^2}c_{e_i}$ is an orthogonal projection of $\vec{f_i}$ along $\vec{c}_{e_i}^\perp$ in S, in the ordinary sense of projection, meaning that this projection operator Π_{e_i} is symmetric and idempotent.

References

[1] Ittai Abraham and Ofer Neiman. "Using Petal-Decompositions to Build a Low Stretch Spanning Tree". In: *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*. STOC '12. New York, New York, USA: Association for Computing Machinery, 2012, pp. 395–406. ISBN: 9781450312455. DOI: 10.1145/2213977.2214015. URL: https://doi.org/10.1145/2213977.2214015.

[2] Stefan Kaczmarz. “Angenaherte auflosung von systemen linearer gleichungen”. In: Bull. Internat. Acad. Polon.Sci. Lettres A, 1937, pp. 335–357.

[3] Jonathan A. Kelner et al. “A Simple, Combinatorial Algorithm for Solving SDD Systems in Nearly-Linear Time”. In: *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*. STOC ’13. Palo Alto, California, USA: Association for Computing Machinery, 2013, pp. 911–920. ISBN: 9781450320290. DOI: 10.1145/2488608.2488724. URL: https://doi.org/10.1145/2488608.2488724.

[4] Daniel A. Spielman and Shang-Hua Teng. “Nearly-Linear Time Algorithms for Graph Partitioning, Graph Sparsification, and Solving Linear Systems”. In: *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*. STOC ’04. Chicago, IL, USA: Association for Computing Machinery, 2004, pp. 81–90. ISBN: 1581138520. DOI: 10.1145/1007352.1007372. URL: https://doi.org/10.1145/1007352.1007372.

[5] Thomas Strohmer and Roman Vershynin. “A Randomized Kaczmarz Algorithm with Exponential Convergence”. In: *Journal of Fourier Analysis and Applications* 15 (2007), pp. 262–278.