A New View on Cycle Toggling Based Laplacian Solvers

Hui Han Chin

CMU-CS-17-134

December 2017

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee
Gary Miller, Chair
David Woodruff

Submitted in partial fulfillment of the requirements
for the Degree of Master of Science

Abstract

An electrical flow on a graph is a special type of flow which corresponds to the physical interpretation of an electrical current induced on a network of conductors when an electrical circuit is set up. Electrical flow has interesting algebraic properties and is a useful algorithmic primitive in modern algorithm design and is used in recent breakthroughs in long-standing problems such as solving the approximate maximum flow problem [CKM$^+$11].

The "minimum energy, demand satisfying, electrical flow" problem is to find an electrical flow of minimum energy that satisfies the demand on the vertices. This problem can be solved using Laplacian Solvers in nearly linear time and many vertex potential based solvers such as [KMP11, CKM$^+$14] have been developed. In [KOSZ13], a "cycle toggling" electrical solver was introduced. The cycle flow solver operates in the dual space of the vertex potential solvers and it is considered a simpler algorithm. Despite having a competitive asymptotic running time, the cycle toggling solver was shown to have poorer performance experimentally when compared to traditional optimization methods [DGM$^+$16].

The goal of this thesis is to give a better analysis of cycle toggling solver and to demystify its performance. First, techniques from vertex potential solvers will be used to improve the performance of cycle toggling solvers. Second, empirical results from the implementations of the solver of [KOSZ13] would be discussed. Finally, the solvers would be reanalyze using the framework of "Stochastic Dual Ascent" framework of [GR15] thereby establishing the connection of Laplacian Solvers with stochastic optimization.

Keywords: Spectral Graph Theory, Electrical Flow, Stochastic Gradient Descent, Primal Dual Methods

Acknowledgements

I would like to express my thanks for my advisor Prof Gary Miller for giving me the opportunity to work with him once more despite the circumstances.

I would like to Prof David Woodruff for giving me advice on this thesis project and as well as the introducing the world of sketching techniques.

Finally, I would like to thank the other graduate students in SCS for helping me to rekindle my interest in Computer Science. Special mention to my friends Chun Kai Ling, Paul Liang Pu, Bo Jian Han, and Timothy Chu, who have contributed to the ideas in this thesis in some way or another.

Contents

Chapter 1

Introduction

An electrical flow on a graph is a special type of flow which corresponds to the physical interpretation of an electrical current induced on a network of conductors when an electrical circuit is set up. Electrical flow has interesting algebraic properties and is a useful algorithmic primitive in modern algorithm design and is used in recent breakthroughs in long-standing problems such as solving the approximate maximum flow problem [CKM$^+$11]. A better understanding of electrical flow will lead to better algorithm designs.

1.1 Minimum Energy, Demand Satisfying Electrical Flow

Given an *undirected* weighted graph, the edges of the graph could be thought of as the resistors in a network and the edge weight, r_e would be the associated resistance. Like any electrical circuit, if a potential difference is applied to a subset of vertices, an electrical flow would be induced onto the graph. Let the flow induced on each edge to be f_e. The electrical energy of an edge is then $\xi(f_e) := r_e f_e^2$ and the energy of the graph is the sum of the electrical energy across the edges.

The "minimum energy and demand satisfying electrical flow", or in short **"Min-energy electrical flow"**, problem is to find an electrical flow of minimum energy that satisfies the given potential difference on the vertices. A more through treatment of the problem can be found in [Bol98, DS84] and the key insight is that the "Min-energy electrical flow" problem can be reformulated as problem of solving the underlying Graph Laplacian system. This connection allows for the power of linear solvers to bring to bear on the optimization problem.

1.2 Fast Laplacian Graph Solvers

Fast and robust linear system solvers are the work horse of many communities through out scientific disciplines, engineering fields and the industry. Linear systems appear in the inner operations of common routines such as solving linear programs, eigenvector calculations, convex optimization, and physical simulations. Thus, improvements in linear system solvers technology will have a huge impact.

Unfortunately, sub-quadratic time solver for general linear systems remains an open problem [Str69, CW90, Wil12]. However, there is substantial progress in the special cases of restricting to linear systems in the form, $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ where $\boldsymbol{A}$ is Symmetric Diagonally Dominant (SDD) matrix. A SDD matrix is a square matrix where each diagonal entry is greater than or equal to the sum of the absolute values of its off diagonals. A graph Laplacian matrix (or Laplacian) is a SDD matrix with the additional constraint that the off diagonal elements are non-positive, and the row sums are 0.

The graph Laplacian represents a weighted, undirected graph that has n vertex and m edges. Formally, it is a $n \times n$ matrix with entries given by

$$\boldsymbol{L}_{i,j} = \begin{cases} \sum_{j \neq i} \boldsymbol{w}_{ij} & \text{if} \quad i = j, \\ -\boldsymbol{w}_{ij} & \text{if } i \neq j \text{ and } i \text{ is incident to } j, \\ 0 & \text{otherwise.} \end{cases} \tag{1.1}$$

There is a simple $O(m)$ procedure to convert any SDD system into a Laplacian system thus the focus would be on Laplacian systems. In addition, graph Laplacians appear frequently and naturally in scientific computing, graph theory, and machine learning problem[CMMP13].

1.3 Prior Works

In a seminal paper by Spielman and Teng [ST14], they exhibited a Laplacian solver that ran in $O(m \log^c n)$ time to constant precision. Their result sparked a renaissance in faster algorithms for wide classes of problems that have not seen improvements in many years[KM09, CKM+11, Mad13], all of which involved the use of a fast linear solver. Since then, there has been substantial improvements in the time to solve Laplacian systems and the present best upper bound is now about $O(m\sqrt{\log n})$ time [KMP14, KMP11, KOSZ13, LS13, CKM+14, PS14, KLP+15].

In [KOSZ13], a "cycle toggling" electrical solver was introduced. It was the first flow based Laplacian solver and this is different from the existing Laplacian solvers such as [CKM+14]

which is vertex potential based. Though the cycle toggling solver has comparable asymptotic performance to potential solver, [BDG16b] have found that despite implementation optimizations, the cycle toggling solvers practical performance is not competitive to linear system solvers.

1.4 Overview

This gap between theoretical and practical performance motivates the investigation in this thesis. The goals of thesis are:

1. A better analysis of the cycle toggling solver and to demystify its practical performance.
2. A better understanding of the primal-dual nature of electrical flows.
3. To establish the connection between the Laplacian solvers and stochastic optimization.
4. To Incorporate methods from potential solvers into flow solvers.

Chapter 2

Preliminaries

We will define some notations and concepts that will be used in the paper. Boldface would used to denote matrices while normal font would be used to denote vectors. All the matrices and vectors would be associated with the graph in context and subscripts would be used in ambiguous situation.

2.1 Matrices Definitions

2.1.1 Graphs and Trees

Definition 2.1.1 (Graph)
Let $G = (V, E, W)$ be a weighted, undirected graph with $|V| = n$ vertices, $|E| = m$ edges, and $\forall w \in W, w > 0$ weights. For simplicity, the graph is assumed to be connected throughout the thesis.

Definition 2.1.2 (Conductance, Resistance)
The weights of graph, W, would be termed as conductance. The conductance matrix is a $|E| \times |E|$ diagonal matrix, $\boldsymbol{C} \in \mathbb{R}^{m \times m}$ where each diagonal entry is the conductance of the edge.
The inverse of the conductance matrix is called the resistance matrix, $\boldsymbol{R} \in \mathbb{R}^{m \times m}$ and $\boldsymbol{R} = \boldsymbol{C}^{-1}$. The conductance matrix can be expressed as $\boldsymbol{R}^{-1}$

Definition 2.1.3 (Spanning Tree and Non-Tree edges)
Let the spanning tree of a connected graph G, $T(G) = T$, be a connected subgraph that contains the all the vertices with minimal edges. Clearly, T has $n - 1$ edge.
Given a spanning tree, the edge set is partitioned into 2 sets: tree edges, denoted as E_t, and non-tree edges, denoted as E_n. There are $m - n + 1$ non-tree edges and for notation convenience, denote this as $m' = m - n + 1$

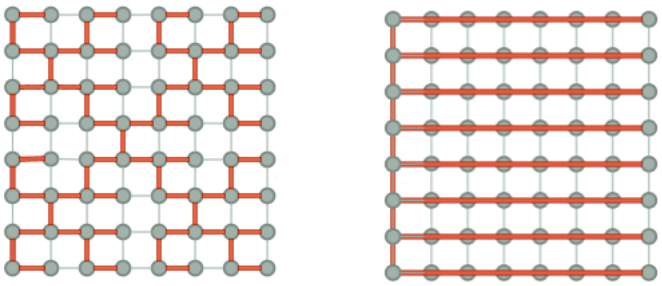


Figure 2.1: Low Stretch Spanning Tree on 2D grid

Definition 2.1.4 (Stretch and Low Stretch Spanning Tree)
The stretch of an edge $e = (a, b) \in E$ with respect to a tree T is

$$st(e) = \frac{\sum_{i \in P_e} r_i}{r_e}$$

where P_e is the unique path of the vertex a to b on the tree T.
The induced stretch of a tree on a graph is the sum of the stretch of all the edges,

$$st(T) = \sum_{e \in E} st(e)$$

A Low Stretch Spanning Tree (LSST) is a tree with low total stretch.

2.1.2 Associated Basis Matrices of the Graph

Definition 2.1.5 (Incidence Matrix)
Given an orientation of the edges of G, each edge $(a, b) \in E$ can be expressed as an ordered pair. It is convenient to orient the edges based on the tree traversal order of a given spanning tree. Let the incidence matrix $\boldsymbol{B}$ be the $|V| \times |E|$ matrix whose rows are indexed by vertices and columns are indexed by edges where

$$\boldsymbol{B}_{v,e} = \begin{cases} 1 & \text{if } e = (u, v), \text{ for some } u \in V, \\ -1 & \text{if } e = (v, u), \text{ for some } u \in V, \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

Order $\boldsymbol{B}$ such that the first $n - 1$ columns are the edges of the spanning tree. $\boldsymbol{B}$ can be represented as 2 block matrices, $\boldsymbol{B}_t$, a $n \times (n - 1)$ matrix for the tree edges, and $\boldsymbol{B}_n$, a $n \times (m - n + 1)$ matrix for the non-tree edges such that

$$\boldsymbol{B} = [\boldsymbol{B}_t | \boldsymbol{B}_n]$$

2.1.3 Laplacian and its Decomposition

Definition 2.1.6 (Laplacian)
The graph Laplacian is a $|V| \times |V|$ matrix defined as

$$\boldsymbol{L}_{i,j} = \begin{cases} \sum_{j \neq i} \boldsymbol{w}_{ij} & \text{if} \quad i = j, \\ -\boldsymbol{w}_{ij} & \text{if } i \neq j \text{ and } i \text{ is incident to } j, \\ 0 & \text{otherwise.} \end{cases} \tag{2.2}$$

Often, the Laplacian is defined as $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$ where $\boldsymbol{D}$ is the degree matrix and $\boldsymbol{A}$ is the weighted adjacency matrix. An alternative characterization is by the incidence matrix. It can be verified that

$$\boldsymbol{L} = \boldsymbol{B}\boldsymbol{C}\boldsymbol{B}^T$$

When $\boldsymbol{C}$ has non-negative edge weights, the Laplacian is a positive semi-definite (PSD) matrix. This version of the graph Laplacian is also known as the combinatorial Laplacian or Kirchhoff matrix.

Definition 2.1.7 (Cycle Matrix)
Given a spanning tree of a graph, since there is a unique tree path for every pair of vertices, every non-tree edge forms a unique cycle with the tree. Given a non-tree e, call this unique cycle, C_e, the fundamental cycle of e. It is crucial to note that the edges on the cycle is oriented and the orientation is given by $\boldsymbol{B}$
Let the cycle matrix $\boldsymbol{K}$ be the $|E| \times |E_n|$ matrix whose rows are indexed by edges and columns are indexed by non-tree edges where

$$\boldsymbol{K}_{a,b} = \begin{cases} 1 & \text{if } a \in C_b \text{ and } a, b \text{ have the same orientation in } C_b, \\ -1 & \text{if } a \in C_b \text{ and } a, b \text{ do not have the same orientation in } C_b, \\ 0 & \text{if } a \notin C_b. \end{cases} \tag{2.3}$$

It is easy to check that $\boldsymbol{B}\boldsymbol{K} = 0$ as the cycle matrix is in the nullspace of the incidence matrix.
Similar to the incidence matrix, by ordering the first $n-1$ rows of $\boldsymbol{K}$ to be the edges of the tree, we can represent $\boldsymbol{K}$ as 2 block matrices, $\boldsymbol{K}_t$, a $(n-1) \times (m-n+1)$ matrix for the tree edges, and $\boldsymbol{K}_n$, a $(m-n+1) \times (m-n+1)$ matrix for the non-tree edges

$$\boldsymbol{K} = \left[\frac{\boldsymbol{K}_t}{\boldsymbol{K}_n} \right]$$

Note that $\boldsymbol{K}_n = I$ since each fundamental cycle would contain only 1 non-tree edge.

We can express $\boldsymbol{K}$ in terms of $\boldsymbol{B}$

$$
\begin{aligned}
\boldsymbol{B}\boldsymbol{K} &= 0 \\
[\boldsymbol{B}_t|\boldsymbol{B}_n]\left[\frac{\boldsymbol{K}_t}{\boldsymbol{K}_n}\right] &= 0 \\
\boldsymbol{B}_t\boldsymbol{K}_t + \boldsymbol{B}_n I &= 0 \\
\boldsymbol{K}_t &= -\boldsymbol{B}_t^+\boldsymbol{B}_n
\end{aligned}
$$

where $\boldsymbol{B}_t^+$ is the pseudo inverse of $\boldsymbol{B}_t$

[Bol98] has a deeper discussion in chapter *II.3: Electrical Networks* on the connections between the incidence and cycle matrices.

2.2 Potentials and Flows

Definition 2.2.1 (Potential or Potential Flow)
Let the vertex potential (potential), v, be a $\mathbb{R}^{|V|}$ vector. Similar to a electrical circuit, a potential flow is an induced flow on the edges where the flow amount is proportional to the conductance and the difference between the potentials of the start and end vertices of the edge.

$$f_{potential} = \boldsymbol{R}^{-1}\boldsymbol{B}^T v$$

Definition 2.2.2 (Generator or Circulation Flow)
Let a potential difference generator (generator),u, be a $\mathbb{R}^{|E_n|}$ vector. This is setting of flows on the non-tree edges. A circulation flow is where the in flow and out flow on every vertex is 0.

$$f_{circulation} = \boldsymbol{K}u$$

Definition 2.2.3 (Flow decomposition)
Recall that $\boldsymbol{B}\boldsymbol{K} = 0$, implying that potential flows are orthogonal to the circulation flows. Thus all flows can be decompose as

$$f = f_{potential} + f_{circulation} = \boldsymbol{R}^{-1}\boldsymbol{B}^T v + \boldsymbol{K}u$$

Definition 2.2.4 (Vertex Demand, Demand Satisfying Flow)
Let the vertex demand (demand), d, be a $\mathbb{R}^{|V|}$ vector which is the required difference between the in flow and out flow of each vertex. Given a demand $\boldsymbol{d}$ a demand satisfying flow, f, is one where each vertex meets the demand. This can be expressed as $\boldsymbol{B}f = d$

2.3 Summary

This is a quick and visual representation of the required matrices and flow concepts that will be used in the subsequent chapters.

Special Matrices

$\boldsymbol{B}$: Incidence Matrix

$\boldsymbol{B}_t$: Incidence Matrix of the Spanning Tree

$\boldsymbol{K}$: Cycle Matrix

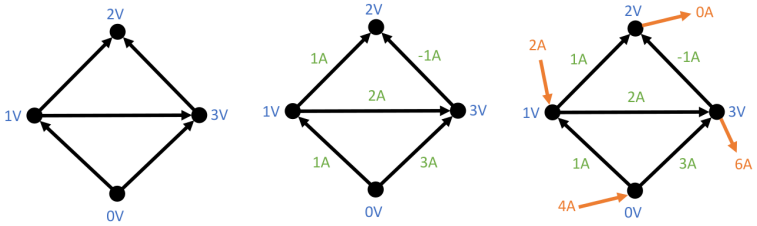
$\boldsymbol{L}$: Laplacian Matrix

Incidence Matrix

$\boldsymbol{B}^T$: potential $\rightarrow$ flow, $\boldsymbol{B}$: flow $\rightarrow$ residual

Graph Laplacian

$$\boldsymbol{L} = \boldsymbol{B}\boldsymbol{C}\boldsymbol{B}^T$$



v $\quad \boldsymbol{C}\boldsymbol{B}^T v$ $\quad \boldsymbol{B}\boldsymbol{C}\boldsymbol{B}^T v$

Incidence and Cycle Matrices

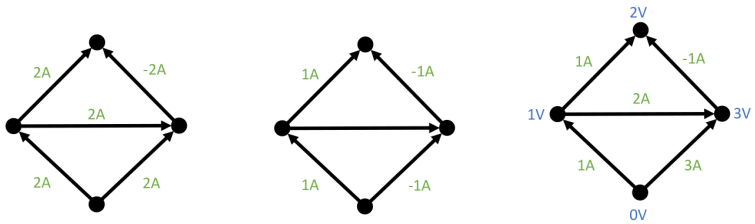
$$\boldsymbol{B} = [\boldsymbol{B}_t | \boldsymbol{B}_n],\ \boldsymbol{K} = \left[\frac{\boldsymbol{K}_t}{\boldsymbol{K}_n}\right],\ \boldsymbol{B}\boldsymbol{K} = 0 \rightarrow \boldsymbol{K}_t = -\boldsymbol{B}_t^+ \boldsymbol{B}_n$$

Potential Flow, Pure Circulations

$$f_{potential} = \boldsymbol{R}^{-1}\boldsymbol{B}^T v,\ f_{circulation} = \boldsymbol{K}u$$

Flow Decomposition

$$f = \boldsymbol{R}^{-1}\boldsymbol{B}^T v + \boldsymbol{K}u = \boldsymbol{R}^{-1}(\boldsymbol{B}^T v + \boldsymbol{R}\boldsymbol{K}u)$$



Flow = $\quad$ Pure Circulation + $\quad$ Potential Flow

Chapter 3

A Cycle Toggling Solver

3.1 KOSZ13 Cycle Toggling Solver

In [KOSZ13], the aim was to develop a minimum-energy and demand satisfying flow (Min-energy electrical flow) solver matches the demand on each vertex exactly but might have ϵ error in getting the energy of the flow. In contrast, the existing vertex potential solvers, such as [KMP11], would have ϵ error in both the energy as well as the meeting the demand.
The key idea for the KOSZ13 solver can be broken down into the following step.

1. If the graph was a tree, then there exists a unique flow satisfying the demand exactly. Call this flow as"tree flow". The tree flow be computed in linear time by a simple tree traversal.

2. Since all connected graph contains a tree, there is tree flow satisfying the demand exactly i.e. $\boldsymbol{B} f_{tree} = d$.

3. Observe that the tree flow is a potential flow since it has no cycles. Also, adding cycles does not change the residual on each vertex i.e. $\boldsymbol{B}(f_{tree} + \boldsymbol{K}u) = d$. This step is called "*Cycle toggling*" in KOSZ13.

4. However, adding cycles changes the energy of the flow.

5. The min-energy flow problem can be casted as finding the set of cycles to add to a tree flow which will result in the minimum energy.

The KOSZ13 solver uses cycle toggling as the main operation. A contribution of [KOSZ13] is to make cycle toggling efficient by using two key algorithmic tools.

1. A good Low Stretch Spanning Tree with stretch of $O(\tau) = O(m \log n \log \log n)$ can be found using using [AN12]. The Low Stretch Spanning Tree ensures that the average length of the cycles is short.

2. Cycle toggle updates with respect to the tree can be done efficiently in $O(\log n)$ using a link-cut tree data structure.

The main theorem of [KOSZ13] is

Theorem 3.1.1 *[KOSZ13] A flow based SDD system solver that gives a $1+\epsilon$ approximation in $O(m \log^2 n \log\log n \log(\epsilon^{-1}))$ time.*

To achieve the stated runtime, [KOSZ13] develop 3 solvers, `SimpleSolver`, `ExampleSolver`, and `FullSolver` that builds upon the previous solver. The core routine of the KOSZ13 solver is the `SimpleSolver` and the breakdown is as follows:

3.1.1 KOSZ13 SimpleSolver

Algorithm 1 `SimpleSolver`

Input: Graph $G = (V, E, R)$ and demand d
Output: Flow that meets demand d, f

1: Find a Low Stretch Spanning Tree, T of graph G.
2: Find the tree flow on T that meets the vertices demand by tree traversal.
3: **for** $\tau \log\left(\frac{st(T)\tau}{\epsilon}\right)$ iterations. **do**
4: Sample each non-tree edge, $e = (a, b)$, with some probability $p_e = \frac{1}{\tau}\frac{R_e}{r_e}$
5: Form cycle, C_e, consisting of e and the path of a to b on T.
6: Minimize the energy of flow on C_e by toggling a circulation of $\frac{\Delta_{C_e}}{R_{C_e}} = \frac{\sum_{i \in C_e} f_i r_i}{\sum_{i \in C_e} r_i}$
7: **return** f.

Pictorially, `SimpleSolver` can be visualized as

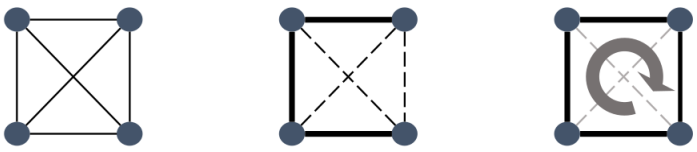


Figure 3.1: "Build Tree and Toggle Cycles"

3.1.2 KOSZ13 SimpleSolver Analysis

Reproducing the key parts of the proof of convergence from [KOSZ13], will give some insights for subsequent analysis. Using the following terms:

f_0: Initial flow from solving the tree flow
f_i: flow at round i
f_{opt}: optimal flow with the minimum energy
τ: Tree condition number of the Low Stretch Tree

Lemma 6.1 (Initial Energy Bound)
The initial energy of the tree-flow can be bounded by the tree condition number.

$$|f_0|_R^2 \leq O(\tau)(|f_{opt}|_R^2)$$

Theorem 4.1 (Convergence of `SimpleSolver`)
At each iteration i, the energy distance to optimal flow

$$E[|f_i|_R^2] - |f_{opt}|_R^2 \leq (1 - \frac{1}{\tau})(|f_0|_R^2 - |f_{opt}|_R^2)$$

Combining both parts by substitution gives a $O(\tau \log(n\epsilon^{-}1))$ iteration count to get a ϵ approximation to the optimal energy.

3.1.3 KOSZ13 FullSolver

To get the full results [KOSZ13] would require the use of 2 more solvers. It high level sketch is given here and they can be seen as parameter optimization of the `SimpleSolver` and they do not introduce new algorithmic techniques.

KOSZ13 Algorithm 5: `ExampleSolver`
Key Ideas: Tree Weight Scaling and Randomized Stopping Time on `SimpleSolve`
Running Time: $O(m \log^2 n \log\log n \log(\epsilon^{-1} \log n))$

KOSZ13 Algorithm 6: `FullSolver`
Key Ideas: Applying Tree Scaling in a sequence for $\log^*(n)$ times
Running Time: $O(m \log^2 n \log\log n \log(\epsilon^{-1}))$

3.1.4 Empirical Studies of KOSZ13

Cycle toggling solver is a much simpler algorithm compared to other solvers of the Min-energy electrical flow[KOSZ13]. However, [DGM+16], [BDG16a] have shown in practice, despite various implementation optimizations such as parallelization and efficient data

structures, the KOSZ13 cycle solver is not competitive with existing potential based techniques such as Conjugate Gradient or tree-preconditioned Conjugate Gradient. While KOSZ13 has good asymptotic performance, more analysis is needed to understand the effects of the "hidden constants" at various stages of the algorithm to understand the gap between its theoretical and practical performance.

Chapter 4

Improved Cycle Toggling

4.1 Simple Example to Examine Possible Approaches to improve Cycle Toggling.

A natural question to ask is can the convergence of KOSZ13 be improve by coming up with a better sequence of cycle toggles?

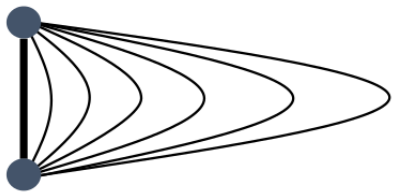


Figure 4.1: m-bond graph

Consider a simple "m-bond" graph comprising of 2 vertex, 1 tree edge, $m-1$ parallel edges and all edges have weight 1. Let the $d = [-1, 1]$ be the demand vector. It can be seen that the initial flow, f_0, of 1 on the tree edge and 0 on the non-tree edges is a feasible flow. Two views of the convergence rate for using cycle toggling to compute the Min-energy flow are examined.

KOSZ13 View
Since every edge has a stretch of 1, the tree condition number is $\tau = m * 1 + m - 4 + 2 = 2m - 2$, thus the expected convergence rate is $1 - \frac{1}{2m-2}$

Spectral Graph Theory View
Let A be the $m \times m$ random walk matrices for cycle toggling non-tree edge k such that $f_{k+1} = A_k f_k$. There are $m - 1$ possible A_k matrices, one for each non-tree edge. In each

A_k is filled with 0 except exactly 4 entries which are $\frac{1}{2}$ namely, $a_{00}, a_{0k}, a_{k0}, a_{kk}$. In the m-bond case, edge 0 is the tree edge.

$$A_k = \begin{pmatrix} \frac{1}{2} & 0 & \cdots & \frac{1}{2} & \cdots & 0 \\ 0 & \cdots & \cdots & 0 & \cdots & 0 \\ \frac{1}{2} & 0 & \cdots & \frac{1}{2} & \cdots & 0 \\ 0 & \cdots & \cdots & 0 & \cdots & 0 \\ 0 & \cdots & \cdots & 0 & \cdots & 0 \end{pmatrix}$$

Let $\bar{A}$ be the expectation of the As, where each of the A_k are chosen with probability $\frac{1}{m}$. The eigenvalues of $\bar{A}$ are:

1. $\lambda = 1$, with multiplicity 1
2. $\lambda = 1 - \frac{1}{2m-2}$, with multiplicity m-2
3. $\lambda = \frac{m-2}{2m-2}$, with multiplicity 1

The eigenvalue of 1 is when the flow has reached minimum energy meaning that regardless which cycle has been toggled, the resulting flow vector is the same. This eigenvalue analysis shows that the KOSZ13 view of the expected convergence rate is $1 - \frac{1}{2m-2}$ is correct with probability $\frac{m-2}{m}$, which is close to 1 for $m >> 2$

However, the smallest eigenvalue of $\lambda = \frac{m-2}{2m-2}$ suggests that the *actual* convergence rate can be much faster than $1 - \frac{1}{2m-2}$ if the correct cycle is picked at every round. Experimental, this can be shown for the "m-bond" case by picking the edge with the largest edge for the initial few rounds of cycle toggling.

4.2 Proposed Approach of Improving Convergence

The simple "M-bond" example have illustrated that it is possible to construct a "clever" sequence of cycle toggle to get better convergence. Some cycle selection heuristics, common in optimization and machine learning, are tested in a series of experiments.

1. **Select based on sequence of subgraphs**: A sequence of sub graphs $G_0 \subset G_1 \subset \cdots \subset G$ that has the same vertex set but are subsets of the edge set as the original graph G. This is idea from the potential solver and the intuition is that it improves sampling at the start as the graph would be sparser

2. **Change cycle selection metric**: KOSZ13 uses uniform sampling of cycles but possible alternative is sampling by stretch.

3. **Select without replacement**: Commonly known as "Shuffle" in machine learning.

4. **Mixing**: Mix between sampling with and without replacement.

5. **Greedy**: Get the edge with the best toggle value.(minimizes the most energy) Assumes that is given by some oracle or can be estimated.

4.3 Experiments Setup

The difference approaches to speed up convergence was tested on different graphs with the following setup

1. Implemented [KOSZ13] `SimpleSolver` without special data structures as the goal was to observe the number of cycle toggle iterations and not true timings. This would be reported as **KOSZ**.

2. Tested on graphs which the Low Stretch Spanning Tree is known.

3. Cycle toggling is run till error within $1e-8$ of true solution's energy. The true solution is found by doing a direct solve using a linear solver.

4. The best heuristics, except greedy, for each graph tested is reported. Greedy is tested but not reported as it would outperform all the other methods however, it takes a longer runtime than solving the original problem.

4.3.1 MChain

Figure 4.2: "Series Graph"

Number of cycle toggle iterations

m=	**KOSZ**	**SChain+mix**	**SChain+shuffle**
128	420	226	115
512	1329	778	327
2048	9449	3086	1528
4096	13586	6500	3222
8192	33838	11516	5529

4.3.2 Mbond

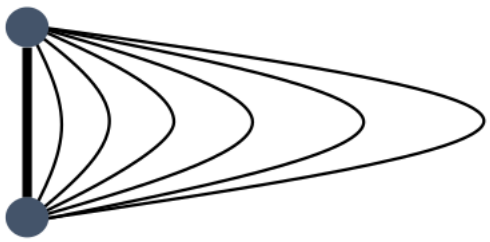


Figure 4.3: "Parallel Graph"

Number of cycle toggle iterations

m=	**KOSZ**	**SChain+mix**
64	1555	1240
256	5525	4719
1024	25809	19865
2048	53693	37688
4096	110486	76018
8192	192,948	115,050

4.3.3 2D-Grid

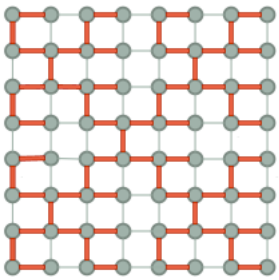


Figure 4.4: "2D Grid with Recursive C Low Stretch Spanning Tree"

Number of cycle toggle iterations

(n,m)=	**KOSZ**	**SChain+mix**
64,112	7155	7331
256,480	80145	73076
1024,1984	706,072	203,438
4096,8064	5,939,078	1,251,767

4.4 Experiments Findings

The experiments shows that **Solver Chain + Mixing** reduces the iteration count compared to **KOSZ**, showing that having a better toggle sequence does yields faster convergence for all the graphs tested. However, it was also observed that once all edges have some flow, then **Solver Chain + Mixing** does not improve significantly over **KOSZ**.

Greedy Toggle shows that there is a lower bound on cycle toggling. Even knowing the best move still requires greater than $O(m)$ iterations. A sample run on the 2D grid example illustrates this point

(n,m)=	256, 480
KOSZ	80145
SChain+Mix	73076
Greedy	30031

Mix of other methods on the 2D grid

The experiments show that might be a lower bound on the number of iterations needed for cycle toggling. While [KOSZ13] gave the expected number of iterations, it does not give the lower bound. Thus there is a need for another approach to show how much lower can the number of iterations be lower.

Chapter 5

Stochastic Dual Ascent and its Connection to Cycle Toggling

The cycle toggling framework of [KOSZ13] shares many resemblances to randomized techniques for solving linear systems. On such algorithm that has the same setup as `SimpleSolve` from KOSZ13 is the Kaczmarz Algorithm.

5.1 Randomized Kaczmarz Algorithm

The Kaczmarz's algorithm is a very simple iterative algorithm for solving linear equation systems of the form $Ax = b$. It was first introduced by Polish mathematician Stefan Kaczmarz in [Kac37] and was rediscovered many times in different area such as image reconstruction. Below is the algorithm.

Algorithm 2 Randomized Kaczmarz Algorithm

Input: Given linear system A and constraints b

Output: Output x such that $|Ax - b|_2^2$ is minimized

1: $x^0 \leftarrow 0$
2: **while** Not converged **do**
3: pick index i from rows uniformly at random.
4: Update $x^{k+1} = x^k - \frac{(A_i^T x^k - b_i)}{|A_i^T|_2^2} A_i^T$
5: **return** x^k

In the original Kaczmarz's algorithm, the row indices where chosen in a fixed order. The randomized version is where the rows are picked at random with uniform probability. This has been shown to give better empirical performance. In [SV07], Strohmer gave an insightful analysis of the Kaczmarz algorithm and showed that picking the rows based on the row norm give better convergence than picking at uniform.

In section 9.3 of [KOSZ13], the authors commented on a possible link of their algorithm to the Randomized Kaczmarz Algorithm. On initial inspection, the linear system to solve in cycle toggling was $\boldsymbol{B}f = d$. However, since the initial tree computed in step 1 will always meet the demand, i.e. $\boldsymbol{B}f_t = d$. If the Kaczmarz algorithm was applied naively, no progress would be made at each iteration, since each row norm will be 0 and no row would be picked. [KOSZ13] did not draw the conclusion that their cycle toggling algorithm was indeed a variant of the Kaczmarz Algorithm. A deep analysis of the connection between randomized Kaczmarz Algorithm and cycle toggling is needed.

5.2 Stochastic Dual Ascent

Inspired by the analysis of [SV07], Gower and Richtarik developed and analyzed a randomized iterative method for solving a consistent system of linear equations which they term as **Stochastic Dual Ascent** [GR15]. The SDA framework was very versatile and [GR15] showed the reductions for the following six common algorithms to the SDA framework:

1. Sketching viewpoint: Sketch and Project
2. Optimization viewpoint: Constrain and Approximate
3. Geometric viewpoint: Random Intersect
4. Algebraic viewpoint 1: Random Linear Solve
5. Algebraic viewpoint 2: Random Update
6. Analytic viewpoint: Random Fixed Point

The SDA framework is used to solve problems of the following primal-dual form.

5.2.1 Primal-Dual Setup

Given $A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m$, invertible $B \in \mathbb{R}^{n\times n}$, the primal and dual problems are:

Primal		**Dual**	
$\underset{x}{\text{minimize}}$	$\frac{1}{2}\lvert x\rvert_B^2$	$\underset{y}{\text{maximize}}$	$b^T y - \frac{1}{2}\lvert A^T y\rvert_{B^{-1}}^2$
subject to	$Ax = b$	subject to	$y \in \mathbb{R}^m$
	$x \in \mathbb{R}^n$		

It is assumed that system is consistent and strong duality holds, meaning that the optimal solutions x^*, y^* exists.

5.2.2 Randomization

The randomization is used into the SDA framework through the use of sampling matrics S. Let S be a $m \times k$ random matrix drawn independently from some distribution $\mathcal{D}$. The update step of most linear solver involves, $AB^{-1}A^T$. The goal in SDA is the select a family $\mathcal{D}$ such that the S matrices drawn make $(S^TAB^{-1}A^TS)^+$ easy to solve.

A simple example is the family of S that select 1 row of A at random. Now suppose if B is the identity matrix, then the update step of SDA for some $S_i \sim \mathcal{D}$ that selects row i of A is;

$$\begin{aligned} S^TAB^{-1}A^TS)^+ &= ((S^TA)(A^TS))^+ \\ &= ((A_i)((A_i^T)))^+ \\ &= \frac{1}{|A_i|^2} \text{ inverse of row norm square} \end{aligned}$$

Which gives the sampling heuristics from the randomized Kaczmarz Algorithm

$\mathcal{D}$ can be view as a family of algorithms that solves the given optimization problem thus it is no surprise that SDA covers a large family of known iterative algorithms.

5.2.3 SDA update steps

[GR15] gives for each iteration, the update rules for the primal and dual problem is as follow:

Definition 5.2.1
Primal Iterates: $x^{k+1} = x^k - B^{-1}A^TS((S^TAB^{-1}A^TS)^+)S^T(Ax^k - b)$

Definition 5.2.2
Dual Iterates: $y^{k+1} = y^k + S((S^TAB^{-1}A^TS)^+)S^T(b - AB^{-1}A^Ty^k)$

Though the update steps might look complicated, they can be simplified with the appropriate choice of S.

5.2.4 Convergence Analysis

[GR15] gave the convergence analysis in Theorem 1.1 (Convergence of primal iterates and residuals) with the following key points:

1. Only weak assumptions on $\mathcal{D}$ is needed. $\mathcal{D}$ has to be a distribution such that expectation matrix, H, below is a well defined and non-singular matrix.

$$H \coloneqq E_{S\sim\mathcal{D}}\left[S(S^TAB^{-1}A^TS)^+S^T\right]$$

The consequence is that the SDA is always able to make an update at every iteration as there is no choice of S that make the resulting update matrix ill defined.

2. Convergence Rates of Primal and Dual problems.

$$\begin{aligned}\text{Convergence Factor: } & \rho = 1 - \lambda^+_{\min}(B^{-\frac{1}{2}}A^THAB^{-\frac{1}{2}})\\ \text{Primal Rate: } & E[|x^k - x^*|_B^2] \leq \rho^k|x^0 - x^*|_B^2\\ \text{Residual Rate: } & E[|Ax^k - b|_B] \leq \rho^{k/2}|A|_B|x^0 - x^*|_B\end{aligned}$$

3. Bounds of the convergence factor.

$$1 - \frac{E[Rank(S^TA)]}{Rank(A)} \leq \rho < 1$$

5.2.5 Solving Min-Energy Flow with SDA

With these new tools, the Min-Energy electrical flow problem can be recast into the SDA framework as a primal problem.

Primal min-energy electrical flow

$$\begin{aligned}\underset{f}{\text{minimize}} \quad & |f|_{\boldsymbol{R}}^2\\ \text{subject to} \quad & \begin{bmatrix}\boldsymbol{K}^T\boldsymbol{R}\\ \boldsymbol{B}\end{bmatrix} f = \begin{bmatrix}0\\ d\end{bmatrix}\end{aligned} \tag{5.1}$$

This version of the problem incorporates the demand satisfying requirements of $\boldsymbol{B}f = d$, with the additional constraint of $\boldsymbol{K}^T\boldsymbol{R}f = 0$, which says that the flow should not have excess flow on the non-tree edge. Using the terminology introduced in section 2, this means that the solution flow can be decomposed into a circulation flow and a potential flow that satisfies the vertex demands.

5.3 Analyzing KOSZ13 Cycle Toggling in SDA

There is a an equivalent mapping for the KOSZ13 `SimpleSolver` in the SDA framework.

SDA Primal		**KOSZ13**	
$\underset{x}{\text{minimize}}$	$\frac{1}{2}\|x\|_B^2$	$\underset{f}{\text{minimize}}$	$\frac{1}{2}f^T Rf$
subject to	$Ax = b$	subject to	$\boldsymbol{K}^T\boldsymbol{R}f = 0$
	$x \in \mathbb{R}^n$		$f \in \mathbb{R}^m$
		given	$\boldsymbol{B}f_0 = d$

Matching the matrices of SDA to KOSZ
$B \to \boldsymbol{R},\ A \to \boldsymbol{K}^T\boldsymbol{R},\ x \to f,\ b \to 0$

This interpretation is a consequence of reinterpreting the Min-energy flow problem in the SDA framework. Similar, the iteration steps of cycle toggle has a similar reinterpretation.

5.3.1 KOSZ13 Cycle toggle iterations

$$\begin{aligned} x^{k+1} &= x^k - B^{-1}A^T S((S^T AB^{-1}A^T S)^+)S^T(Ax^k - b) && \text{from SDA} \\ \to f^{k+1} &= f^k - \boldsymbol{R}^{-1}\boldsymbol{R}\boldsymbol{K}S((S^T\boldsymbol{K}^T\boldsymbol{R}\boldsymbol{R}^{-1}\boldsymbol{R}\boldsymbol{K}S)^+)S^T(\boldsymbol{K}^T\boldsymbol{R}f^k) && \text{changing symbols} \\ &= f^k - (\boldsymbol{K}S)((S^T\boldsymbol{K}^T\boldsymbol{R}\boldsymbol{K}S)^+)(S^T\boldsymbol{K}^T)\boldsymbol{R}f^k && \text{simplifying} \end{aligned}$$

Using a nice family of random matrices can simplify the SDA update steps. Observe by choosing $S_i = e_i$, where e_i is ith standard basis vector in $\mathbb{R}^{m'}$, with probability $p_i = \frac{1}{\tau}\log(\frac{R_e}{r_e})$.

$$f^k - (\boldsymbol{K}S)((S^T\boldsymbol{K}^T\boldsymbol{R}\boldsymbol{K}S)^+)(S^T\boldsymbol{K}^T)\boldsymbol{R}f^k = f^k - \alpha\boldsymbol{K}_i$$

Which is chose 1 row of $\boldsymbol{K}$ and scaling it by α at every iteration. Recall that $\boldsymbol{K}$ is the cycle matrix, so the SDA interpretation of the KOSZ13 is that is performing randomized coordinate descent in the cycle matrix space. This completes the reinterpretation of the cycle toggle iteration of KOSZ13 in the SDA framework.

5.3.2 KOSZ13 Setup

Since KOSZ13 starts of with a demand satisfying flow, updating by some row of $\boldsymbol{K}^T\boldsymbol{R}$ will keep the flow at each iteration within the space of demand-satisfying flow as $\boldsymbol{B}\boldsymbol{K} = 0$.

5.3.3 Convergence Analysis of KOSZ13 using SDA

[GR15] gives the convergence factor ,ρ, to be

$$\rho \coloneqq\ 1 - \lambda^+_{\min}(B^{-\frac{1}{2}}A^T HAB^{-\frac{1}{2}})$$

To show the convergence of KOSZ under SDA, the expectation matrix H and then ρ needs to be calculated.

To calculate H

$$\begin{aligned}
H &= E_{S\sim\mathcal{D}}\left[S(S^TAB^{-1}A^TS)^+S^T\right] && \text{by definition}\\
\rightarrow &= E_{S\sim D}\left[S(S^T\boldsymbol{K}^T\boldsymbol{R}\boldsymbol{R}^{-1}\boldsymbol{R}^T\boldsymbol{K}S)^+S^T\right] && \text{substituting}\\
&= E_{S\sim D}\left[S(S^T\boldsymbol{K}^T\boldsymbol{R}^T\boldsymbol{K}S)^+S^T\right]\\
&= \sum_{i\in\text{non-tree}} p(S_i)\left(S_i(R_{c_i})^{-1}S_i^T\right) && \text{toggle only non-tree}\\
&= \sum_i \left(\frac{R_{c_i}}{\tau r_i}\right)\left(S_i(R_{c_i})^{-1}S_i^T\right) && \text{substituting}\\
&= \frac{1}{\tau}R_n^{-1}
\end{aligned}$$

To calculate $\rho = 1-\lambda^+_{\min}(B^{-\frac{1}{2}}A^THAB^{-\frac{1}{2}})$, first bound $\lambda^+_{\min}(B^{-\frac{1}{2}}A^THAB^{-\frac{1}{2}})$, which is the smallest non-zero eigenvalue.

$$\begin{aligned}
(B^{-\frac{1}{2}}A^THAB^{-\frac{1}{2}}) &= (\boldsymbol{R}^{-\frac{1}{2}}\boldsymbol{R}^T\boldsymbol{K}(\frac{1}{\tau}R_n^{-1})\boldsymbol{K}^T\boldsymbol{R}\boldsymbol{R}^{-\frac{1}{2}})\\
&= \frac{1}{\tau}\boldsymbol{R}^{\frac{1}{2}}\boldsymbol{K}\boldsymbol{R}_n^{-1}\boldsymbol{K}^T\boldsymbol{R}^{\frac{1}{2}}
\end{aligned}$$

Next is to show that $Rank(\boldsymbol{R}^{\frac{1}{2}}\boldsymbol{K}\boldsymbol{R}_n^{-1}\boldsymbol{K}^T\boldsymbol{R}^{\frac{1}{2}}) = m' = m-n+1$. Where m' is the number of non-tree edges.
Given that $Rank(A) = Rank(A^TA)$, it suffices to show $Rank(\boldsymbol{R}_n^{-\frac{1}{2}}\boldsymbol{K}^T\boldsymbol{R}^{\frac{1}{2}}) = m'$

$$\begin{aligned}
\boldsymbol{R}_n^{-\frac{1}{2}}\boldsymbol{K}^T\boldsymbol{R}^{\frac{1}{2}} &= \boldsymbol{R}_n^{-\frac{1}{2}}\left[\frac{\boldsymbol{K}_t}{\boldsymbol{I}}\right]\boldsymbol{R}^{\frac{1}{2}}\\
&= \left[\frac{\boldsymbol{R}_n^{-\frac{1}{2}}\boldsymbol{K}_t\boldsymbol{R}_t^{\frac{1}{2}}}{\boldsymbol{I}}\right]
\end{aligned}$$

Since $\boldsymbol{R}_n^{-\frac{1}{2}}\boldsymbol{K}^T\boldsymbol{R}^{\frac{1}{2}})$ has a $m'\times m'$ identity matrix in the lower m', the rank of it is m' Given $\boldsymbol{R}^{\frac{1}{2}}\boldsymbol{K}\boldsymbol{R}_n^{-1}\boldsymbol{K}^T\boldsymbol{R}^{\frac{1}{2}}$ is a $m\times m$ matrix and its rank is m', this means that it has $m-m' = n-1$, 0's as eigenvalues.

Observe that by expanding,

$$\boldsymbol{R}^{\frac{1}{2}}\boldsymbol{K}\boldsymbol{R}_n^{-1}\boldsymbol{K}^T\boldsymbol{R}^{\frac{1}{2}} = \left[\begin{array}{c|c}\boldsymbol{R}_t^{\frac{1}{2}}\boldsymbol{K}_t\boldsymbol{R}_n^{-1}\boldsymbol{K}_t^T\boldsymbol{R}_t^{\frac{1}{2}} & \boldsymbol{R}_t^{\frac{1}{2}}\boldsymbol{K}_t\boldsymbol{R}_n^{-\frac{1}{2}}\\ \hline \boldsymbol{R}_n^{-\frac{1}{2}}\boldsymbol{K}_t^T\boldsymbol{R}_t^{\frac{1}{2}} & \boldsymbol{I}\end{array}\right]$$

Remove the first $n-1$ row & columns and apply **Cauchy Interlacing Theorem** $n-1$ times. We get the bound for $\lambda^+_{\min}$

$$\lambda^+_{\min}(\boldsymbol{R}^{\frac{1}{2}}\boldsymbol{K}\boldsymbol{R}_n^{-1}\boldsymbol{K}^T\boldsymbol{R}^{\frac{1}{2}}) \geq \lambda^+_{\min}(\boldsymbol{I})$$
$$\rightarrow \lambda^+_{\min}(\boldsymbol{R}^{\frac{1}{2}}\boldsymbol{K}\boldsymbol{R}_n^{-1}\boldsymbol{K}^T\boldsymbol{R}^{\frac{1}{2}}) \geq 1$$

5.3.4 Upper and lower bounds for ρ

To complete the convergence rate analysis, substitute in the bound for $\lambda^+_{\min}$

$$\rho = 1 - \frac{1}{\tau}\left(\lambda^+_{\min}(\boldsymbol{R}^{\frac{1}{2}}\boldsymbol{K}\boldsymbol{R}_n^{-1}\boldsymbol{K}^T\boldsymbol{R}^{\frac{1}{2}})\right)$$
$$\rho \leq 1 - \frac{1}{\tau} \qquad \text{upper bound}$$
$$1 - \frac{E[Rank(S^TA)]}{Rank(A)} \leq \rho \qquad \text{From [GR15]}$$
$$1 - \frac{E[Rank(S^T\boldsymbol{K}^T\boldsymbol{R})]}{Rank(\boldsymbol{K}^T\boldsymbol{R})} \leq \rho \qquad \text{substitution}$$
$$1 - \frac{1}{m'} \leq \rho \leq 1 - \frac{1}{\tau} \qquad (S^T\boldsymbol{K}^T\boldsymbol{R}) \text{ are all rank 1}$$
$$1 - \frac{1}{m-n+1} \leq \rho \leq 1 - \frac{1}{\tau}$$

Not only does this analysis of KOSZ13 matches the upper bound in [KOSZ13], it gives a lower bound too.

5.4 Revisiting the m-bond example in the new SDA framework

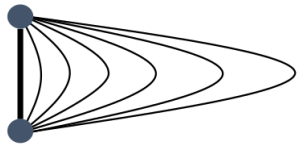


Figure 5.1: "Parallel Graph"

Recap KOSZ13 SimpleSolve Analysis
Since every edge has a stretch of 1, the tree condition number is $\tau = m * 1 + m - 4 + 2 =$

$2m-2$, thus the convergence rate is $1-\frac{1}{2m-2}$

New KOSZ13 SimpleSolve using the SDA Analysis
The convergence factor is $1-\frac{1}{m-1} \leq \rho \leq 1-\frac{1}{2(m-1)}$. Thus the converge rate is $O(1-\frac{1}{2(m-1)})$ and the total iteration count is $O(m \log m)$ for a constant approximation.

New Solver Chain Analysis using SDA Analysis
Using a simple solver chain where the number of non-tree edges double at every stage, the convergence factor at the k stage with 2^k non-tree edges is $1-\frac{1}{2^k} \leq \rho \leq 1-\frac{1}{2^k+1}$. The total iteration count is $O(m)$ for a constant approximation.

5.4.1 New insights from using the SDA analysis of cycle toggling on the m-bond

1. The lower bound of $1-\frac{1}{m-n+1} \leq \rho$ shows that single cycle updates cannot reach $O(m)$ iterations even though there might be good eigenvalues.

2. Solver Chain methods helped (initially) as the number of non-tree edges are reduced making $1-\frac{1}{m'}$ smaller. However once the non-tree edges approach $m-n+1$, the convergence rate approaches $O(\tau \log(n\epsilon^{-1}))$ in expectation as analyzed by [KOSZ13].

3. Theoretically, for a general graph, there is no good sequence of single cycle updates that improve the convergence rate. Practically for fixed family of graphs, there can be heuristics used to create good solver chains to improve initial convergence.

4. Any major improvement for cycle toggle type of algorithms would have to come from toggling more than 1 cycle at each step.

Chapter 6

Future Work and Conclusion

The first area for potential improvement is doing multiple cycle toggling. It can be shown experimentally that toggling multiple cycles simultaneously, the total iteration can be reduce. However, as of now, there is no fast data structure that supports fast update operations on multiple cycles. In [KOSZ13], a link-cut tree data structure was used to allow fast cycle updates.

The second area for potential improvement is to allow the flow vector to not meet the demand at each iteration. Cycle toggling always ensures that the current flow vector meets the demand as a cycle toggle only adds a circulation flow. Adding potential flow can perturb the demand vector thus there is a need to "fix" the flow so that it meets the demand. The challenge is to devise an efficient way to fix the flow as fix the flow can be as hard as solving the original problem.

Lastly, applying [GR15] directly to the general version of the electrical flow problem can yield more interesting algorithm. Currently, the cycle toggling algorithms operate in the primal space and there is possible techniques for electrical flow problems in the dual space. In addition, SDA allows for the primal-dual ascent style of iterations.

In conclusion, the main contributions of this work are;

1. Shown that the cycle toggling algorithm of [KOSZ13] can be analysis more efficiently in the Stochastic Gradient Ascent framework of [GR15]. This yields a much shorter proof of the convergence rate of KOSZ13, it tools gives an lower bound of the convergence rate.

2. Shown in experiments and in analysis that single cycle toggling cannot do better than $O(m)$ iterations, even with a oracle.

3. Demonstrated some practical heuristics for speeding up the convergence for doing cycle toggling as a technique in solving linear systems.

While this work did not radically improve the performance of the cycle toggle solver, it developed a deeper connection between the world of solvers and randomized algorithms. Hopefully with future inventions in data structures and new analysis techniques will the primal-dual nature of electrical flow be fully exploited to develop even faster solver algorithms.

Bibliography

[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.

[BDG16a] Erik G. Boman, Kevin Deweese, and John R. Gilbert. An empirical comparison of graph Laplacian solvers. In *Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2016, Arlington, Virginia, USA, January 10, 2016*, pages 174–188, 2016.

[BDG16b] Erik G. Boman, Kevin Deweese, and John R. Gilbert. Evaluating the dual randomized Kaczmarz Laplacian linear solver. *Informatica*, 40:95–107, 2016.

[Bol98] B. Bollobas. *Modern Graph Theory*. Graduate Texts in Mathematics. Springer New York, 1998.

[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 ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 273–282, 2011. Available at http://arxiv.org/abs/1010.2921.

[CKM+14] Michael B. Cohen, Rasmus Kyng, Gary L. Miller, Jakub W. Pachocki, Richard Peng, Anup Rao, and Shen Chen Xu. Solving SDD linear systems in nearly $m\log^{1/2} n$ time. In *STOC*, pages 343–352, 2014.

[CMMP13] Hui Han Chin, Aleksander Madry, Gary L. Miller, and Richard Peng. Runtime guarantees for regression problems. In *Proceedings of the 4^{th} conference on Innovations in Theoretical Computer Science*, ITCS '13, pages 269–282, New York, NY, USA, 2013. ACM. Available at http://arxiv.org/abs/1110.1358.

[CW90] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.

[DGM+16] Kevin Deweese, John R. Gilbert, Gary L. Miller, Richard Peng, Hao Ran Xu, and Shen Chen Xu. An empirical study of cycle toggling based laplacian solvers. *CoRR*, abs/1609.02957, 2016.

[DS84] Peter G. Doyle and J. Laurie Snell. *Random Walks and Electric Networks*, volume 22 of *Carus Mathematical Monographs*. Mathematical Association of America, 1984.

[GR15] Robert M Gower and Peter Richtarik. Stochastic Dual Ascent for Solving Linear Systems. *ArXiv e-prints*, December 2015.

[Kac37] S. Kaczmarz. Angenäherte Auflösung von Systemen linearer Gleichungen. *Bulletin International de l'Académie Polonaise des Sciences et des Lettres*, 35:355–357, 1937.

[KLP+15] Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A. Spielman. Sparsified Cholesky and multigrid solvers for connection Laplacians. *CoRR*, abs/1512.01892, 2015. Available at http://arxiv.org/abs/1512.01892.

[KM09] Jonathan A. Kelner and Aleksander Madry. Faster generation of random spanning trees. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 13–21, 2009. Available at: http://arxiv.org/abs/0908.1448.

[KMP11] Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-m log n time solver for SDD linear systems. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 590–598, Washington, DC, USA, 2011. IEEE Computer Society. Available at http://arxiv.org/abs/1102.4842.

[KMP14] I. Koutis, G. Miller, and R. Peng. Approaching optimality for solving SDD linear systems. *SIAM Journal on Computing*, 43(1):337–354, 2014. Available at http://dx.doi.org/10.1137/110845914.

[KOSZ13] Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving SDD systems in nearly-linear time. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, STOC '13, pages 911–920, New York, NY, USA, 2013. ACM. Available at http://arxiv.org/abs/1301.6628.

[LS13] Yin Tat Lee and Aaron Sidford. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. In *Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, FOCS '13, pages 147–156, Washington, DC, USA, 2013. IEEE Computer Society.

[Mad13] Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 253–262. IEEE, 2013. Available at http://arxiv.org/abs/1307.2205.

[PS14] Richard Peng and Daniel A. Spielman. An efficient parallel solver for SDD linear systems. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 333–342, New York, NY, USA, 2014. ACM. Available at http://arxiv.org/abs/1311.3286.

[ST14] D. Spielman and S. Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM Journal on Matrix Analysis and Applications*, 35(3):835–885, 2014. Available at http://arxiv.org/abs/cs/0607105.

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.

[SV07] T. Strohmer and R. Vershynin. A randomized Kaczmarz algorithm with exponential convergence. *ArXiv Mathematics e-prints*, February 2007.

[Wil12] Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 887–898. ACM, 2012. Available at: http://theory.stanford.edu/~virgi/matrixmult-f.pdf.