# Almost-Linear-Time Algorithms for Markov Chains and New Spectral Primitives for Directed Graphs*

Michael B. Cohen
MIT
Cambridge, Massachusetts 02139
USA
micohen@mit.edu

Jonathan Kelner
MIT
Cambridge, Massachusetts 02139
USA
kelner@mit.edu

John Peebles
MIT
Cambridge, Massachusetts 02139
USA
jpeebles@mit.edu

Richard Peng
Georgia Tech
Atlanta, Georgia 30332, USA
rpeng@cc.gatech.edu

Anup B. Rao
Georgia Tech
Atlanta, Georgia 30332, USA
anup.rao@gatech.edu

Aaron Sidford
Stanford University
Stanford, California 94305, USA
sidford@stanford.edu

Adrian Vladu
MIT
Cambridge, Massachusetts 02139
USA
avladu@mit.edu

## ABSTRACT

In this paper, we begin to address the longstanding algorithmic gap between general and reversible Markov chains. We develop directed analogues of several spectral graph-theoretic tools that had previously been available only in the undirected setting, and for which it was not clear that directed versions even existed.

In particular, we provide a notion of approximation for directed graphs, prove sparsifiers under this notion always exist, and show how to construct them in almost linear time. Using this notion of approximation, we design the first almost-linear-time directed Laplacian system solver, and, by leveraging the recent framework of [Cohen-Kelner-Peebles-Peng-Sidford-Vladu, FOCS'16], we also obtain almost-linear-time algorithms for computing the stationary distribution of a Markov chain, computing expected commute times in a directed graph, and more. For each problem, our algorithms improve the previous best running times of $O((nm^{3/4} + n^{2/3}m)\log^{O(1)}(n\kappa\epsilon^{-1}))$ to $O((m + n2^{O(\sqrt{\log n \log\log n})})\log^{O(1)}(n\kappa\epsilon^{-1}))$ where $n$ is the number of vertices in the graph, $m$ is the number of edges, $\kappa$ is a natural condition number associated with the problem, and $\epsilon$ is the desired accuracy.

We hope these results open the door for further studies into directed spectral graph theory, and that they will serve as a stepping stone for designing a new generation of fast algorithms for directed graphs.

## CCS CONCEPTS

• **Theory of computation** → **Sparsification and spanners**; **Preconditioning**; **Random walks and Markov chains**; • **Mathematics of computing** → *Continuous optimization*;

## KEYWORDS

Markov chains, stationary distribution, linear system solver, directed graphs, sparsification, preconditioning

**ACM Reference format:**
Michael B. Cohen, Jonathan Kelner, John Peebles, Richard Peng, Anup B. Rao, Aaron Sidford, and Adrian Vladu. 2017. Almost-Linear-Time Algorithms for Markov Chains
and New Spectral Primitives for Directed Graphs. In *Proceedings of 49th Annual ACM SIGACT Symposium on the Theory of Computing, Montreal, Canada, June 2017 (STOC'17),* 10 pages.
DOI: 10.1145/3055399.3055463

## 1 INTRODUCTION

In the analysis of Markov chains, there has been a longstanding algorithmic gap between the general case, corresponding to random walks on directed graphs, and the special case of reversible chains, for which the corresponding graph can be taken to be undirected. This gap begins with the most basic computational task—computing the stationary distribution—and persists for many of the fundamental problems associated with random walks, such as computing hitting and commute times, escape probabilities, and personalized PageRank vectors. In the undirected case, there are algorithms for all of these problems that run in linear or nearly-linear time. In the directed case, however, the best algorithms have historically been much slower. Specifically, the best running times were given by a recent precursor to the present paper [13], which showed that one

*A full version of this paper is available as [12].


*STOC'17, Montreal, Canada*

978-1-4503-4528-6/17/06…$15.00
DOI: 10.1145/3055399.3055463

could solve these problems on a graph with $n$ vertices and $m$ edges in time $\widetilde{O}(nm^{3/4} + n^{2/3}m)$.[1] Prior to that work, it was unknown whether one could solve any of them faster than the time needed to solve an arbitrary linear system with the given size and sparsity, i.e. $\Theta(\min(mn, n^\omega))$ time, where $\omega < 2.3729$ is the exponent for matrix multiplication.

This gap has its origins in a broader discrepancy between the state of algorithmic spectral graph theory in undirected and directed settings. While the undirected case has a richly developed theory and a powerful collection of algorithmic tools, similar results have remained somewhat elusive for directed graphs. In particular, the problems mentioned above can be expressed in terms of the linear algebraic properties of the Laplacian matrix of a graph, and it was shown in [13] how to reduce all these problems to the solution of a small number of Laplacian linear systems. In the undirected case, there has been a tremendously successful line of research on how to use the combinatorial properties of graphs to accelerate the solution of such systems, culminating in algorithms that can solve them in nearly-linear time [14, 22–27, 32, 36]. Unfortunately, these solvers relied heavily on several features that seemed intrinsic to the undirected case and did not appear to be available for directed graphs, thereby precluding an analogous solver for directed Laplacians. In particular, the undirected solvers relied on:

**Knowledge of the kernel/stationary distribution:** Up to a simple rescaling by the vertex degrees, vectors in the kernel of a Laplacian correspond to stationary distributions of the corresponding random walk. For undirected graphs, the kernel is spanned by the all-ones vector on each of the connected components, so it and the space of stationary distributions can be easily computed in linear time. For directed graphs, however, this is no longer the case, and finding the stationary distribution does not seem to be any easier than the original problem of solving Laplacian linear systems. In fact, while stationary distributions of random walks on directed graphs have been studied for over 100 years [3], and computing them has been extensively investigated in both theory and practice (see e.g. [33, 37]), the $\widetilde{O}(nm^{3/4} + n^{2/3}m)$ result in [13] was the first to find them in less time than is required to find the kernel of a general matrix.

**Symmetry and positive semidefiniteness:** Undirected Laplacians are symmetric and positive semidefinite. Essentially every aspect of algorithmic spectral graph theory uses this symmetry to treat the Laplacian as a quadratic form and relies on its expression as a sum of positive semidefinite contributions from each of the edges to analyze its properties. This includes the Laplacians' connection to the graph's cut structure, their relationship to electrical circuits and effective resistances, the notion of graph inequalities and spectral approximation, the combinatorial construction of preconditioners, and the iterative methods used to solve Laplacian systems. On the other hand, directed Laplacians are asymmetric matrices, and their naive symmetrizations are not typically positive semidefinite.

**Sparsification** One of the most powerful algorithmic tools in the undirected setting is the ability to construct *sparsifiers* [5, 6, 35]. These allow one to approximate an arbitrarily dense graph by a sparse graph that has only a slightly super-linear number of edges. The classical notion of cut sparsification requires that the value of every cut in the original graph be approximately preserved in the sparsifier; the more recent notion of spectral sparsification is stronger, and also implies the former property. For directed graphs, it can be shown that, even for the weaker notion of cut sparsification, such sparsifiers do not generally exist. One simple example is the complete bipartite graph. (See Section 1.1.2.) In fact, it was not known how to define any other useful notion of sparsification for which this would not be the case.

In this paper, we show how to cope with these fundamental differences, and begin to address the algorithmic gap between general and reversible Markov chains. Our core technical result is the first almost-linear-time solver for directed Laplacian systems. Using the work from [13], this yields the first almost-linear-time algorithms for computing a host of fundamental objects and quantities associated with a random walk on a directed graph, including the stationary distribution, hitting and commute times, escape probabilities, and personalized PageRank vectors.

More broadly, constructing our solver required the development of directed versions of several foundational tools and techniques from undirected algorithmic spectral graph theory. Most notably, and perhaps surprisingly, we show that it is possible to develop a useful notion of spectral approximation and sparsification of directed graphs, and that sparsifiers under this definition exist and can be constructed efficiently.

In addition to their direct application to the analysis of Markov chains, we hope that both the solver itself and the sparsification machinery will prove to be useful tools in the further development of fast graph algorithms. In the undirected case, sparsifiers have been a core algorithmic tool since the early 1990s [1, 6, 17, 20, 21], and fast solvers for undirected Laplacian solvers have recently led to an explosion of algorithms operating in the so-called "Laplacian Paradigm" [38], in both cases leading to asymptotic improvements for many of the core algorithmic problems for undirected graphs. Given the success these methods have enjoyed in the case of undirected graphs, we hope that their directed analogues will spark similar progress in the directed setting.

## 1.1 Previous Work

In this section, we briefly review some of the previous work related to our results and techniques. Given the extensive prior research on Markov chains, spectral graph theory, sparsification, solving general and Laplacian linear systems, and computing PageRank, we do not attempt to give a comprehensive overview of the literature; instead we simply describe the work that most directly relates to or motivates this paper.

*1.1.1 Directed Laplacian Systems, Stationary Distributions, and PageRanks.* The most direct precursor to this work is a recent paper by a subset of the authors [13]. As mentioned above, it showed that, by exploiting linear algebraic properties of directed Laplacians, one could obtain faster algorithms for a wide range of problems involving directed random walks. Prior to this paper, it seemed

[1] We use $\widetilde{O}$ notation to suppress terms that are polylogarithmic in $n$, the natural condition number of the problem $\kappa$, and the desired accuracy $\epsilon$. We use the term "nearly linear" to refer to algorithms whose running time is $\widetilde{O}(m) = m \log^{O(1)}(n\kappa\epsilon^{-1})$ and "almost linear" to refer to algorithms that are linear up to sub-polynomial (but possibly super-logarithmic) factors, i.e., whose running time is $O(m(n\kappa\epsilon^{-1})^{o(1)})$.

quite possible that the similarities between directed and undirected Laplacians were largely syntactic, and that there was no way to use the structure of directed Laplacians or random walks to obtain asymptotically faster algorithms. In particular, despite extensive theoretical and applied work in computer science, mathematics, statistics, and numerical scientific computing, all algorithms that we are aware of prior to [13] for obtaining high-quality[2] solutions for directed Laplacian systems, stationary distributions, or personalized PageRank vectors either have a polynomial dependence on a condition number or related parameter (such as a random walk's mixing time or PageRank's restart probability), or they apply a general-purpose linear algebra routine and thus run in at least the $\Omega(\min(mn, n^\omega))$ time currently required to solve arbitrary linear systems.

By showing that this was not the case, [13] provided the first indication that one could actually use the structure of directed Laplacian systems to accelerate their solution, which provided a strong motivation to see how much of an improvement was possible. It also created hope that the recently successful research program in building and applying fast algorithms for solving (symmetric) Laplacian systems [22–25, 27, 32, 36] could be applied to give more direct improvements to running times for solving combinatorial optimization problems on directed graphs.

In addition to motivating the search for faster Laplacian solvers, [13] provided a set of reductions that we will directly apply in this paper. In order to prove its results, [13] showed how to reduce a range of algorithmic questions about directed walks, such as computing the stationary distribution, hitting and commute times, escape probabilities, and personalized PageRank vectors, to solving a small number of linear systems in directed Laplacians.

It turns out that it is easier to work with such systems in the special case where the graph is Eulerian. One of the main technical tools in [13] is a reduction to this special case. They did this by giving an iterative method that solved a general Laplacian system by solving a small number of systems in which the graph is Eulerian. Together, this showed that to solve the aforementioned problems, it suffices to give a solver for Eulerian graphs, and that this only incurs a factor of $\widetilde{O}(1)$ overhead. It then obtained all of its results by constructing an Eulerian solver that runs in time $\widetilde{O}(m^{3/4}n + mn^{2/3})$. In this paper we construct an Eulerian solver that runs in time $m^{1+o(1)}$ and then just directly apply these reductions to obtain our other results.

However, while [13] opened the door for further algorithmic improvements in analyzing Markov chains, the arguments in it provided little evidence that the running time could be improved to anything approaching what was known in the undirected case. Indeed, while the techniques in it suggested that it might be possible to obtain further improvements, even the most optimistic interpretations of the structural results in [13] only gave hope for achieving running times of roughly $\widetilde{O}(m\sqrt{n})$. This would make it no faster than some of the existing algorithms that use undirected Laplacian solvers to solve problems on directed graphs, such as the $\widetilde{O}(m^{10/7})$ algorithms for unit cost maximum flow [30, 31] and shortest path with negative edge lengths [15], or the $\widetilde{O}(m\sqrt{n})$ type bounds for minimum cost flow [28]. As such, while this would provide better results for the applications to Markov chains, it would rule out the hope of obtaining improved results for these directed problems by replacing the undirected solver with a directed one.

Intuitively, the solver in [13] worked by showing how one could use the existence of a fast undirected solver to solve directed Laplacians. For a directed Eulerian Laplacian $\mathcal{L}$, it showed that the symmetrized matrix $\mathbf{U} = (\mathcal{L} + \mathcal{L}^\top)/2$ is the Laplacian of an undirected graph, and that the symmetric matrix $\mathcal{L}^\top \mathbf{U}^+ \mathcal{L}$ was, in a certain sense, reasonably well approximated by $\mathbf{U}$. Given a linear system $\mathcal{L}\vec{x} = \vec{b}$, one could then form the equivalent system $\mathcal{L}^\top \mathbf{U}^+ \mathcal{L}\vec{x} = \mathcal{L}^\top \mathbf{U}^+ \vec{b}$ and use a fast undirected Laplacian solver to apply $\mathbf{U}^+$. One could then hope that the fact that the matrix on the left is somewhat well-approximated by $\mathbf{U}$ would imply that $\mathbf{U}^+$ is a sufficiently good preconditioner for it to yield an improved running time. It turned out that, while this would actually be the case in exact arithmetic, numerical issues provided a legitimate obstruction. This necessitated a more involved scheme, which gave a slightly slower running time of $\widetilde{O}(m^{3/4}n + mn^{2/3})$, rather than the roughly $\widetilde{O}(n^{2/3}m)$ running time that what would have been achieved by exact arithmetic.

The way this algorithm works provides a good intuitive explanation for why one would not expect it to give a solver yielding substantial improvements for combinatorial "Laplacian Paradigm" algorithms that rely on undirected solvers. At its root, the solver from [13] works by trying to find the right way to ignore the directed structure and solve the system with an undirected solver; thus it is on essentially the same footing as the algorithms it would hope to improve. The obstructions it faces are rooted in the fact that directed Laplacians are fundamentally not very well-approximated by undirected ones. In essence, the difference between the solver in this paper and the one presented in [13] is that, instead of figuring out how to properly neglect the directed structure, the solver we present here intrinsically works with asymmetric (directed) objects, and redevelops the theory from the ground up to properly capture them.

#### 1.1.2 *Directed Graph Sparsification and Approximation.*

While sparsification of undirected graphs has been extensively studied [4–6, 17, 29, 34, 35, 39], there has been very little success extending the notion to directed graphs. In fact, it was not even clear that there should exist a useful definition under which directed graphs should have sparsifiers with a subquadratic number of edges, and for many of the natural definitions one would propose, sparsification is provably impossible.

For instance, a natural first attempt would be to try to generalize the classical notion of cut sparsification for undirected graphs [6, 19]. Given any weighted undirected graph $G$, Benczur and Karger showed that one could construct a new graph $H$ with at most $O(n \log n/\epsilon^2)$ edges such that the value of every cut in $G$ is within a multiplicative factor of $1 \pm \epsilon$ of its value in $H$. While this definition makes sense for directed graphs as well, there is no analogous existence result. Indeed it is not hard to construct directed graphs for which any such approximation must have $\Omega(n^2)$ edges.

[2] By high-quality, we mean that the algorithm should be able to find a solution with error $\epsilon$ in time that is sub-polynomial in $1/\epsilon$, i.e. $(1/\epsilon)^{o(1)}$. For PageRank there were some known techniques for achieving better dependence on $n$ and $m$ at the expense of a polynomial dependence on $1/\epsilon$ [2, 9–11].

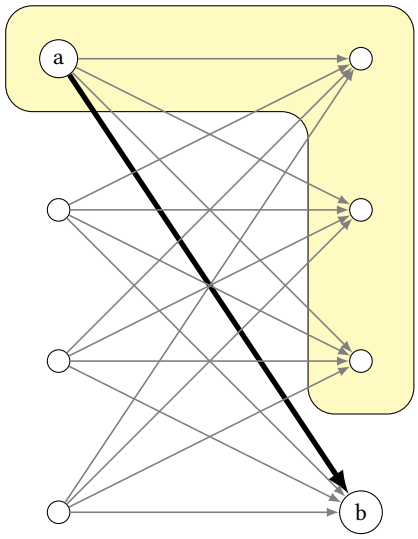


**Figure 1: An example of the family of cuts described in Equation (1.1). The only edge leaving the highlighted set is $a \to b$, so any sparsifier that omits it will fail to approximate the corresponding cut.**

For example, consider the directed complete bipartite graph $K$ on the vertex set $A \cup B$ with all edges directed from $A$ to $B$. For each pair of $a \in A$ and $b \in B$, the directed cut

$$E(\{a\} \cup B \setminus \{b\}, \{b\} \cup A \setminus \{a\}) \tag{1.1}$$

contains only the edge $a \to b$. (See Figure 1.) Removing this edge from the graph would change the value of this cut from 1 to 0, resulting in an infinite multiplicative error.

Any graph that multiplicatively approximates the cuts in $K$ must have $|E(B, A)| = 0$, so it must be supported on a subset of the edges of $K$, and the above then shows that such a graph must contain the edge $a \to b$ for every $a \in A$ and $b \in B$. It thus follows that any graph that approximately preserves every cut in $K$ must contain all $|A||B|$ potential edges, so $K$ has no nontrivial sparsifier under this definition.

It would therefore seem that any attempt at reducing the number of edges in a directed graph while preserving the combinatorial structure is doomed to fail. However, Eulerian graphs present a natural setting that circumvents this because cuts in Eulerian graphs have the same amount of edge weight going in each direction, the bipartite graph counterexamples above are precluded. This balancedness allows one to incorporate sparsification based tools for flows and routings in this setting to solve combinatorial flow and cut problems quickly on Eulerian graphs [16].

Most closely related to our notion of sparsification of directed graphs is the work by Chung on Cheeger's inequality for directed graphs [8]. This result transforms the graph into an Eulerian graph $G$ in a way identical to how we obtain Eulerian graphs [13]: by rescaling each edge weight by the probability of its source vertex in a stationary distribution. It then relates the convergence rate of random walks on $G$ to the eigenvalues of the undirected graph obtained by removing directions on all edges. Specifically if the Eulerian directed Laplacian is $\mathcal{L}$, this symmetrization is $\left(\mathcal{L} + \mathcal{L}^\top\right)/2$.

Since the eigenvalues of the symmetrization of an Eulerian graph give information about random walks on the original graph, it might be tempting to define approximation for Eulerian graphs in terms of whether their symmetrizations approximate each other in the conventional positive semidefinite sense. For our purposes, we require (and obtain) a substantially stronger notion of approximation that preserves much of the directed structure that would be erased by symmetrizing. The reason why we need a stronger notion of approximation is that we want graphs that approximate each other under this notion to be good preconditioners of one another. In contrast, if one defines approximation according to whether the symmetrizations approximate one another, one would have to say that the length $n$ undirected cycle and the length $n$ directed cycle approximate each other, since they are both Eulerian and have the same undirected symmetrization. However, they are not good preconditioners of one another, and using one as a substitute for the other would incur very large losses in our applications. Under the notion of approximation we introduce in this paper, these graphs differ by a factor of $\Omega(n^2)$.

*1.1.3 Laplacian System Solvers.* Our algorithms build heavily on the literature for solving undirected Laplacians systems. Since undirected Laplacians are special cases of directed Laplacians, any directed solver will yield an undirected solver when given a symmetric input. It is thus helpful to consider what undirected solver we would like our method to resemble in this case.

There are now a fairly large number of reasonably distinct algorithms for solving such systems, and we believe that several of them provide a template that could be turned into a working directed solver. Of these, the one that our solver most closely resembles is the parallel solver by Peng and Spielman [32], which we will briefly summarize here.

To simplify the notation and avoid having to keep track of degree normalizations, we only consider regular graphs when giving the intuition behind the algorithm. Suppose that we are given a $d$-regular undirected graph $G$ with Laplacian $\mathcal{L} = d\mathbf{I} - \mathbf{A} = d(I - \mathcal{A})$, where $\mathcal{A} = \mathbf{A}/d$ has $\|\mathcal{A}\| < 1$ on $\ker(\mathcal{L})^\perp$. For simplicity, in the equations that follow, we restrict our attention to the space perpendicular to the kernel of $\mathcal{L}$. With this convention, the algorithm of [32] is then motivated by the series expansion

$$(\mathbf{I} - \mathcal{A})^{-1} = \sum_{i \geq 0} \mathcal{A}^i = \prod_{k \geq 0} \left(\mathbf{I} + \mathcal{A}^{2^k}\right), \tag{1.2}$$

which is a matrix version of the standard scalar identity $1/(1-x) = 1 + x + x^2 + x^3 + \cdots = 1(1+x)(1+x^2)(1+x^4)\cdots$. If $\lambda$ is the smallest nonzero eigenvalue of $\mathbf{I} - \mathcal{A}$, then truncating this product at $k = \Theta(\log 1/\lambda)$ yields a constant relative error, which can be made arbitrarily small by further increasing $k$. Hence if $\lambda > 1/\text{poly}(n)$, we obtain a small error by multiplying the first $O(\log n)$ terms of the product. This seems to suggest a good algorithm for solving a system $\mathcal{L}\vec{x} = \vec{b}$: simply compute $\mathbf{I} + \mathcal{A}^{2^k}$ for $k = 0, \ldots, t = O(\log n)$ and then return $\frac{1}{d}(\mathbf{I} + \mathcal{A}^{2^0}) \cdots (\mathbf{I} + \mathcal{A}^{2^t})\vec{b}$.

Unfortunately, this algorithm (implemented naively) would be too slow. As $k$ grows, $\mathcal{A}^k$ quickly becomes dense, so computing it requires repeatedly squaring dense matrices, which takes time

$O(n^\omega)$. To deal with this, their algorithm instead replaces these matrices with sparse approximations of them. Peng and Spielman showed that given a graph with $n$ vertices and $m$ edges, one can compute a sparse approximation of the requisite squared matrix in nearly-linear-time.

Making this idea work requires care, since in general it is not true that the product of two matrices will be well approximated by the product of their approximations. For positive semidefinite matrices, however, there is a variant of this statement that holds if one takes the products symmetrically: if $\mathbf{A}$ and $\mathbf{B}$ are PSD and $\mathbf{A}$ is a good approximation of $\mathbf{B}$, then for any matrix $\mathbf{V}$, $\mathbf{V}^\top\mathbf{AV}$ is a good approximation of $\mathbf{V}^\top\mathbf{BV}$ . This led the authors of [32] to work with a more stable symmetric version of the series described above, which allowed them to obtain their result.

This turns out to be a reasonably convenient template for our directed solver. In particular, it has fewer moving parts than many of the other methods, and it does not require constructing combinatorial objects, like low-stretch spanning trees. Instead it directly relies on sparsification, which is our main new technical tool for directed graphs.

Unfortunately, we cannot directly apply the methods described above, since the symmetric product constructions that are used to control the error are no longer available for the (asymmetric) Laplacians of directed graphs. Moreover, the strong notions of graph approximation and positive semidefinite inequalities that facilitate the analysis for the undirected solver are unavailable in the directed setting. As such, we end up having to work with weaker error guarantees, and correct the extra error they introduce using a more involved iterative method.

## 1.2 Our Results

In this paper, we show that, in spite of these seemingly fundamental differences between the directed and undirected settings, we can develop directed analogues of several of the core spectral primitives that have been deployed to great effect on undirected graphs, and we use them to obtain the first almost-linear-time algorithms for many of the central problems in the analysis of non-reversible Markov chains. The main new theoretical tools and algorithmic primitives we introduce are:

- **Directed graph approximation:** We develop a well-behaved notion of spectral approximation for directed graphs, despite the fact that the corresponding Laplacians lack the symmetry and positive semidefiniteness properties that the undirected version crucially relies on. Our definition specializes to the standard version based on PSD matrix inequalities when applied to undirected graphs, and it retains many of the useful features of the undirected definition. For example, our notion of graph approximations roughly preserve the behavior of random walks, behave well under composition and change of basis, retain certain key aspects of the combinatorial structure, and provide good preconditioners for iterative methods.
- **Directed sparsification:** We show that, under our notion of approximation, any strongly-connected directed graph can be approximated by a *sparsifier* with only $\widetilde{O}(n/\epsilon^2)$ edges, and we give an algorithm to compute such a sparsifier in almost-linear time. To our knowledge, this is the first time that directed sparsifiers with $o(n^2)$ edges have been proven to exist, even non-algorithmically, for any computationally useful definition that retains the directed structure of a graph.
- **Almost-linear-time solvers for directed Laplacian systems:** Given the Laplacian $\mathcal{L} = \mathbf{D} - \mathbf{A}^\top$ of a directed graph with $n$ vertices and $m$ edges, we provide an algorithm that leverages our sparsifier construction to solve the linear system $\mathcal{L}\vec{x} = \vec{b}$ in time

$$\begin{aligned}\mathcal{T} &= O\left(m + n2^{O(\sqrt{\log n \log\log n})}\right)\log^{O(1)}\left(n\kappa\epsilon^{-1}\right)\\ &= O\left(m + n^{1+o(1)}\right)\log^{O(1)}\left(n\kappa\epsilon^{-1}\right),\end{aligned} \tag{1.3}$$

  where $\kappa = \max(\kappa(\mathcal{L}), \kappa(\mathbf{D}))$ is the maximum of the condition numbers of $\mathcal{L}$ and $\mathbf{D}$, improving on the best previous running time of $O\left(nm^{3/4} + n^{2/3}m\right)\log^{O(1)}\left(n\kappa\epsilon^{-1}\right)$. To do so, we introduce a novel iterative scheme and analysis that allows us to mitigate the accumulation of errors from multiplying sparse approximations without having access to the more stable constructions and bounds available for symmetric matrices.

In [13], we provided a suite of reductions that used a solver for directed Laplacians to solve a variety other problems. Plugging our new solver's running time into these reductions immediately gives the following almost-linear-time algorithms:[3]

- **Computing stationary distributions:** We can compute a vector within $\ell_2$ distance $\epsilon$ of the stationary distribution of a random walk on a strongly connected directed graph in time $\mathcal{T}$.
- **Computing Personalized PageRank vectors:** We can compute a vector within $\ell_2$ distance $\epsilon$ of the Personalized PageRank vector with restart probability $\beta$ for a directed graph in time $\mathcal{T}\log^2(1/\beta)$.
- **Simulating random walks:** We can compute the escape probabilities, commute times, and hitting times for a random walk on a directed graph and estimate the mixing time of a lazy random walk up to a polynomial factor in time $\mathcal{T}$.
- **Estimating all-pairs commute times:** We can build a data structure of size $\widetilde{O}(n\epsilon^{-2}\log n)$ in time $\mathcal{T}/\epsilon^2$ that, when queried with any two vertices $a$ and $b$, outputs a $1 \pm \epsilon$ multiplicative approximation to the expected commute time between $a$ and $b$.
- **Solving row- and column-diagonally dominant linear systems:** We can solve linear systems that are row- or column-diagonally dominant in time $\mathcal{T}\log K$, where $K$ denotes the ratio of the largest and smallest diagonal entries.

This gives the first almost-linear-time algorithm for each of these problems. For all of them, the best previous running time for obtaining high-quality solutions was what is obtained by replacing $\mathcal{T}$ with $O\left(nm^{3/4} + n^{2/3}m\right)\log^{O(1)}\left(n\kappa\epsilon^{-1}\right)$ and was proven in [13].

[3] We use $\mathcal{T}$ to denote anything of the form given in equation 1.3, not the time required for one call to the solver. Some of the reductions call the solver a logarithmic number of times, so precise value of the $\log^{O(1)}(n\kappa\epsilon^{-1})$ term varies among the applications. Also, note that in this paper we give solving running times in terms of the condition number of symmetric Laplacian whereas in [13] they are often given in terms of the condition number of the corresponding diagonal matrix. However it is well-known that these differ only by a $O(\text{poly}(n))$ factor and as they are in the logarithmic terms, this does not affect the running times.

## 2 PRELIMINARIES

First we give notation in Section 2.1 and then we give basic information about directed Laplacians in Section 2.2. Much of this is inherited from [13]. With this notation in place we give an overview of our approach in Section 2.3.

### 2.1 Notation

**Matrices:** We use bold to denote matrices and let $\mathbf{I}, \mathbf{0} \in \mathbb{R}^{n\times n}$ denote the identity matrix and zero matrix respectively. For a matrix $\mathbf{A}$ we use $\text{nnz}(\mathbf{A})$ to denote the number of non-zero entries in $\mathbf{A}$. When $\mathbf{A} \in \mathbb{R}^{n\times n}$ we use $\text{supp}(\mathbf{A})$ to denote the subset of $[n]$ corresponding to the indices for which at least one of the corresponding row or column in $\mathbf{A}$ is non-zero.

**Vectors:** We use the vector notation when we wish to highlight that we are representing a vector. We let $\vec{0}, \vec{1} \in \mathbb{R}^n$ denote the all zeros and ones vectors, respectively. We use $\vec{1}_i \in \mathbb{R}^n$ to denote the $i$-th basis vector, i.e. $(\vec{1}_i)_j = 0$ for $j \neq i$ and $(\vec{1}_i)_i = 1$. Occasionally, when it is obvious from the context, we apply scalar operations to vectors with the interpretation that they be applied coordinate-wise. As with matrices, we use $\text{supp}(\vec{x})$ to denote the indices of $\vec{x}$ with non-zero entries.

**Positive Semidefinite Ordering:** For symmetric matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n\times n}$ we use $\mathbf{A} \preceq \mathbf{B}$ to denote the condition that $x^\top \mathbf{A} x \leq x^\top \mathbf{B} x$, for all $x$. We define $\succeq$, $\prec$, and $\succ$ analogously. We call a symmetric matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ positive semidefinite (PSD) if $\mathbf{A} \succeq \mathbf{0}$. For vectors $x$, we let $\|x\|_{\mathbf{A}} \stackrel{\text{def}}{=} \sqrt{x^\top \mathbf{A} x}$. For asymmetric $\mathbf{A} \in \mathbb{R}^{n\times n}$ we let $\mathbf{U}_{\mathbf{A}} \stackrel{\text{def}}{=} \frac{1}{2}(\mathbf{A} + \mathbf{A}^\top)$ and note that $x^\top \mathbf{A} x = x^\top \mathbf{A}^\top x = x^\top \mathbf{U}_{\mathbf{A}} x$ for all $x \in \mathbb{R}^n$.

**Operator Norms:** For any norm $\|\cdot\|$ defined on vectors in $\mathbb{R}^n$ we define the *seminorm* it induces on $\mathbb{R}^{n\times n}$ by $\|\mathbf{A}\| = \max_{x\neq 0} \frac{\|\mathbf{A}x\|}{\|x\|}$ for all $\mathbf{A} \in \mathbb{R}^{n\times n}$. When we wish to make clear that we are considering such a ratio we use the $\to$ symbol; e.g., $\|\mathbf{A}\|_{\mathbf{H}\to\mathbf{H}} = \max_{x\neq 0} \frac{\|\mathbf{A}x\|_{\mathbf{H}}}{\|x\|_{\mathbf{H}}}$, but we may also simply write $\|\mathbf{A}\|_{\mathbf{H}} \stackrel{\text{def}}{=} \|\mathbf{A}\|_{\mathbf{H}\to\mathbf{H}}$ in this case. For symmetric positive definite $\mathbf{H}$ we have that $\|\mathbf{A}\|_{\mathbf{H}\to\mathbf{H}}$ can be equivalently expressed in terms of $\|\cdot\|_2$ as $\|\mathbf{A}\|_{\mathbf{H}\to\mathbf{H}} = \|\mathbf{H}^{1/2}\mathbf{A}\mathbf{H}^{-1/2}\|_2$. Also note that $\|\mathbf{A}\|_1$ is the is the maximum $\ell_1$ norm of a column of $\mathbf{A}$, and $\|\mathbf{A}\|_\infty$ is the maximum $\ell_1$ norm of a row of $\mathbf{A}$.

**Diagonals** For $x \in \mathbb{R}^n$ we let $\mathbf{diag}(x) \in \mathbb{R}^{n\times n}$ denote the diagonal matrix with $\mathbf{diag}(x)_{ii} = x_i$ and typically use $\mathbf{X} \stackrel{\text{def}}{=} \mathbf{diag}(x)$. For $\mathbf{A} \in \mathbb{R}^{n\times n}$ we let $\text{diag}(\mathbf{A}) \in \mathbb{R}^n$ denote the vector corresponding to the diagonal of $\mathbf{A}$, i.e. $\text{diag}(\mathbf{A})_i = \mathbf{A}_{ii}$ and we let $\mathbf{diag}(\mathbf{A})$ denote the diagonal matrix having the same diagonal as $\mathbf{A}$.

**Linear Algebra** For a matrix $\mathbf{A}$, we let $\mathbf{A}^+$ denote the (Moore-Penrose) pseudoinverse of $\mathbf{A}$. For a symmetric positive semidefinite matrix $\mathbf{B}$, we let $\mathbf{B}^{1/2}$ denote the square root of $\mathbf{B}$, that is the unique symmetric positive semidefinite matrix such that $\mathbf{B}^{1/2}\mathbf{B}^{1/2} = \mathbf{B}$. Furthermore, we let $\mathbf{B}^{+/2}$ denote the pseudoinverse of the square root of $\mathbf{B}$. We use $\ker(\mathbf{A})$ to denote nullspace (kernel) of $\mathbf{A}$. We use $\text{span}(x_1, x_2, ..., x_k)$ to denote the subspace spanned by $x_1, ..., x_k$. For a symmetric PSD matrix $\mathbf{A}$ we let $\lambda_*(\mathbf{A})$ denote the smallest non-zero eigenvalue of $\mathbf{A}$.

**Misc**: We let $[n] \stackrel{\text{def}}{=} \{1, ..., n\}$. For $\mathbf{A} \in \mathbb{R}^{n\times n}$, let $\kappa(\mathbf{A}) \stackrel{\text{def}}{=} \|\mathbf{A}\|_2 \cdot \|\mathbf{A}^+\|_2$ denote the condition number of $\mathbf{A}$. For symmetric PSD matrices $\mathbf{A}$ and $\mathbf{B}$ with the same kernel, let $\kappa(\mathbf{A}, \mathbf{B}) \stackrel{\text{def}}{=} \kappa(\mathbf{A}^{+/2}\mathbf{B}\mathbf{A}^{+/2})$ denote their relative condition number (e.g. if $\alpha\mathbf{B} \preceq \mathbf{A} \preceq \beta\mathbf{B}$ then $\kappa(\mathbf{A}, \mathbf{B}) \leq \beta/\alpha$). Note that our use of pseudoinverse rather than inverse in these definitions is non-standard but convenient.

### 2.2 Directed Laplacians

Here we provide notation regarding directed Laplacians and review basic facts regarding these matrices that were proved in [13]. We begin with some basic definitions and notation regarding Laplacians:

**Directed Laplacian:** A matrix $\mathcal{L} \in \mathbb{R}^{n\times n}$ is called a *directed Laplacian* if (1) its off diagonal entries are non-positive, i.e. $\mathcal{L}_{i,j} \leq 0$ for all $i \neq j$, and (2) it satisfies $\vec{1}^\top \mathcal{L} = \vec{0}$, i.e. $\mathcal{L}_{ii} = -\sum_{j\neq i} \mathcal{L}_{ji}$ for all $i$.

**Associated Graph:** To every directed Laplacian $\mathcal{L} \in \mathbb{R}^{n\times n}$ we associate a graph $G_{\mathcal{L}} = (V, E, w)$ with vertices $V = [n]$, and edges $(i, j)$ of weight $w_{ij} = -\mathcal{L}_{ji}$, for all $i \neq j \in [n]$ with $\mathcal{L}_{ji} \neq 0$. Occasionally we write $\mathcal{L} = \mathbf{D} - \mathbf{A}^\top$ to denote that we decompose $\mathcal{L}$ into the diagonal matrix $\mathbf{D}$ (where $\mathbf{D}_{ii} = \mathcal{L}_{ii}$ is the out degree of vertex $i$ in $G_{\mathcal{L}}$) and non-negative matrix $\mathbf{A}$ (which is weighted adjacency matrix of $G_{\mathcal{L}}$, with $\mathbf{A}_{ij} = w_{ij}$ if $(i, j) \in E$, and $\mathbf{A}_{ij} = 0$ otherwise).

**Eulerian Laplacian:** A matrix $\mathcal{L}$ is called an *Eulerian Laplacian* if it is a directed Laplacian with $\mathcal{L}\vec{1} = \vec{0}$. Note that $\mathcal{L}$ is an *Eulerian Laplacian* if and only if its associated graph is Eulerian.

**(Symmetric) Laplacian**: A matrix $\mathbf{U} \in \mathbb{R}^{n\times n}$ is called a *symmetric* or *undirected Laplacian* or just a *Laplacian* if it is symmetric and a directed Laplacian. Note that the graph associated with an undirected Laplacian is undirected, i.e. for every forward edge there is a backward edge of the same weight.

**Running Times:** Our central object is almost always a directed Laplacian $\mathcal{L} = \mathbf{D} - \mathbf{A} \in \mathbb{R}^{n\times n}$, where $m = \text{nnz}(\mathbf{A})$, $U \stackrel{\text{def}}{=} \max_{i,j} |\mathbf{A}_{ij}| / \min_{i,j:\mathbf{A}_{ij}\neq 0} |\mathbf{A}_{ij}|$. We use $\tilde{O}(\cdot)$ notation to suppress factors polylogarithmic in $n$, $m$, $U$, and $\kappa$, the natural condition number of the particular problem.

### 2.3 Overview of Our Approach

Here we provide an overview of our approach for solving linear systems in directed Laplacians. We split it into three parts. In the first part, Section 2.3.1, we describe how to reduce the problem to the special case of solving Eulerian Laplacians with polynomial condition number. In the second part, Section 2.3.2 we cover the efficient construction of sparsifiers. Finally, in the third part, Section 2.3.3, we discuss how to use the sparsifier construction to build an almost-linear-time solver for polynomially well-conditioned Eulerian Laplacian systems.

*2.3.1 Reductions.* We begin by applying two reductions. The first is a result from [13], which states that one can solve row- and column-diagonally dominant linear systems, which include general directed Laplacian systems, by solving a small number of Laplacian systems in which the graphs are Eulerian. Letting $\mathcal{T}$ be the time required to solve an Eulerian Laplacian system in the sense that finding an approximate solution $x'$ to $\mathcal{L}x = b$ such that $\|x' - \mathcal{L}^+ b\|_{\mathbf{U}} \leq \epsilon \|b\|_{\mathbf{U}^+}$, with high probability, can be done in time $O(\mathcal{T} \log(1/\epsilon))$ time, where $\mathbf{U} = \frac{1}{2}(\mathcal{L} + \mathcal{L}^\top)$, we have the following theorem:

**Theorem 2.1** (Theorem 42 from [13]). *Let $\mathbf{M}$ be an arbitrary $n \times n$ column-diagonally-dominant or row-diagonally-dominant matrix with diagonal $\mathbf{D}$. Let $b \in im(\mathbf{M})$. Then for any $0 < \epsilon \leq 1$, one can compute, with high probability and in time*

$$O\left(\mathcal{T} \log^2\left(\frac{n \cdot \kappa(\mathbf{D}) \cdot \kappa(\mathbf{M})}{\epsilon}\right)\right)$$

*a vector $x'$ satisfying $\|\mathbf{M}x' - b\|_2 \leq \epsilon \|b\|_2$.*

*Furthermore, all the intermediate Eulerian Laplacian solves required to produce the approximate solution involve only matrices $\mathbf{R}$ for which $\kappa(\mathbf{R} + \mathbf{R}^\top)$, $\kappa(\mathbf{diag}(\mathbf{R})) \leq (n\kappa(\mathbf{D})\kappa(\mathbf{M})/\epsilon)^{O(1)}$.*

If we were to combine this directly with our algorithm for solving Eulerian Laplacian systems, it would give a running time of $\widetilde{O}\left(\left(m + n \exp O\left(\sqrt{\log \kappa \cdot \log\log \kappa}\right)\right) \log^{O(1)}(1/\epsilon)\right)$ to solve linear systems in a directed Laplacian $\mathcal{L} = \mathbf{D} - \mathbf{A}^\top$, where $\kappa$ is the condition number of the normalized Laplacian $\mathbf{D}^{-1/2}\mathcal{L}\mathbf{D}^{-1/2}$. While $\kappa$ is typically polynomial in $n$, it is possible for it to be exponential, so we would like our running time to depend on it logarithmically, instead of just sub-polynomially.

We show how to do this in the full version of the paper, where we give an algorithm to solve an arbitrarily ill-conditioned Eulerian Laplacian systems by solving $O(\log(n\kappa))$ Eulerian Laplacians whose condition numbers are polynomial in $n$. This allows us to restrict our attention for the rest of the paper to the case where $\kappa$ is polynomial in $n$ and, when used in conjunction with our linear system solving algorithm, gives the final running time of $\log^{O(1)}(n\kappa\epsilon^{-1}))$.

*2.3.2 Sparsification.* Our primary new graph theoretic tool is a directed notion of spectral sparsifiers, along with efficient techniques for constructing them for an Eulerian graph and its square. As discussed in the introduction, there are seemingly intrinsic problems with many of the notions of directed sparsification that one would propose based on analogies to the undirected case. In particular, both the cut-based and spectral notions have seemingly fatal issues that preclude their use in directed graphs. For the cut-based notion, as shown in Section 1.1.2, good sparsifiers provably don't exist for some graphs.

If one instead seeks to generalize the undirected definition of spectral sparsifiers, which requires a sparsifier $H$ of a graph $G$ to obey $(1-\epsilon)\vec{x}^\top \mathcal{L}_H \vec{x} \leq \vec{x}^\top \mathcal{L}_G \vec{x} \leq (1+\epsilon)\vec{x}^\top \mathcal{L}_H \vec{x}$, the problems are perhaps even more severe. For instance, when $G$ is directed $\mathcal{L}_G$ is no longer symmetric, so it's not clear that it makes sense to use it as a quadratic form $\vec{x}^\top \mathcal{L}_G \vec{x}$, and doing so essentially symmetrizes it and discards the directed structure, since $\vec{x}^\top \mathcal{L}_G \vec{x} = \vec{x}^\top \mathcal{L}_G^\top \vec{x} = \vec{x}^\top \left(\frac{\mathcal{L}_G + \mathcal{L}_G^\top}{2}\right) \vec{x}$. In addition, the resulting quadratic form is not typically PSD, i.e. there often exist $\vec{x}$ for which $\vec{x}^\top \mathcal{L}_G \vec{x} < 0$, in which case $G$ would not approximate itself under the definition given for $\epsilon > 0$.

One also has to deal with the fact that, unlike in the undirected case, the kernels of directed graph Laplacians are rather subtle objects: for a strongly-connected graph $G$, the kernel of $\mathcal{L}_G = \mathbf{D} - \mathbf{A}^\top$ is given by $\mathbf{D}^{-1}\phi$, where $\phi$ is the stationary distribution of the random walk on $G$. This carries various problematic consequences, including the fact that $\mathcal{L}$ and $\mathcal{L}^\top$ typically have different kernels, and even small changes in the graph can change whether $\mathcal{L}\vec{x} = 0$ for a given vector $\vec{x}$.

Our approach to this is based on the fact that many of these problems do not occur for Eulerian graphs. In particular, if $\mathcal{L}$ is the Laplacian of an Eulerian directed graph $G$, $\mathbf{U}_{\mathcal{L}} = (\mathcal{L} + \mathcal{L}^\top)/2$ is the Laplacian of an undirected graph and thus positive semidefinite, and cuts in the corresponding undirected graph are the same as those in $G$. In addition, the kernel of $\mathcal{L}$ is spanned by the all-ones vector and is the same as the kernel of $\mathcal{L}^\top$. In addition, the following was shown in [13], which says that the Laplacian of any strongly connected graph can be turned into an Eulerian Laplacian by applying a diagonal scaling:

**Lemma 2.2** (Lemma 1 from [13], abridged). *Given a directed Laplacian $\mathcal{L} = \mathbf{D} - \mathbf{A}^\top \in \mathbb{R}^{n \times n}$ whose associated graph is strongly connected, there exists a positive vector $\vec{x} \in \mathbb{R}^n_{>0}$ (unique up to scaling) such that $\mathcal{L} \cdot \mathbf{diag}(\vec{x})$ is an Eulerian Laplacian. Furthermore,* $\ker(\mathcal{L}) = \text{span}(\vec{x})$, *and* $\ker(\mathcal{L}^\top) = \text{span}(\vec{1})$.

Moreover, it was shown in [13] that one could find a high-precision approximation to this scaling efficiently given access to an Eulerian solver.

Intuitively, we define our notion of sparsification and approximation for Eulerian graphs, and we show that this notion induces a well-behaved definition for other strongly-connected graphs through the Eulerian scaling. As we do not want to neglect the directed structure, we will think of Laplacians as linear operators, not quadratic forms, and we study their sizes through various operator norms.

For Laplacians of Eulerian graphs, we use the fact that their symmetrizations are PSD, and our definition of approximation will demand that the difference between the two operators be small relative to the corresponding quadratic form. More precisely, we say that an Eulerian Laplacian $\mathcal{L}_H$ $\epsilon$-approximates another Eulerian Laplacian $\mathcal{L}_G$ if $\left\|\mathbf{U}_{\mathcal{L}_G}^{+/2}(\mathcal{L}_H - \mathcal{L}_G)\mathbf{U}_{\mathcal{L}_G}^{+/2}\right\|_2 \leq \epsilon$. We note that this use of $\mathbf{U}_{\mathcal{L}_G}$ is closely related to the $\mathcal{L}_G^\top \mathbf{U}_{\mathcal{L}_G}^+ \mathcal{L}_G$ matrix that appeared in [13]. The difference, however, is that we are not trying to directly use this matrix as a symmetric stand-in for our Laplacian; we are working directly with the original (asymmetric) Laplacians and are just using it to help define a matrix norm.

To construct sparsifiers of Eulerian graphs with respect to this notion, we follow a similar approach to the one originally used by Spielman and Teng for spectral sparsification, but carefully tailored to the directed setting. The idea is to first partition our graph into well-connected components. Because the cuts in an Eulerian graph match those in its symmetrization, it makes sense to do this partitioning by simply partitioning the corresponding undirected

graph into clusters with good expansion. We use existing decomposition techniques to argue that one can find such a partition with a significant fraction of the edges contained in the clusters. We then show a concentration result for asymmetric matrices that says that appropriately sub-sampling within these clusters preserves the relevant structure reasonably well while only keeping a small number of edges relative to the cluster size.

In the undirected case, one would just repeat this procedure until the graph is sparse. Where our procedure differs, however, is that we keep track of the directed structure along the way, and "patch" the subsampled object to keep it from diverging from what it should be. In particular, the sampling procedure, when applied to an Eulerian graph will produce a non-Eulerian graph. However, we add additional edges to fix this after every sampling step and use our concentration bounds to show that the patches we add are sufficiently small to not decrease the quality of our approximation.

Carefully, analyzing this procedure allows us to produce a sparsifier in nearly linear time. However, in order to use our sparsification routine to produce a solver, we also need to sparsify the Laplacian of the square of a graph. To do this, we could just explicitly form the square and then sparsify it. However, we would like to perform this procedure in time that is nearly-linear in the number of edges of the original graph, whereas explicitly forming the square would cause the running time to grow with the number of edges of the square, which could be substantially larger. To prevent this, we instead show how to work with an implicit representation of the square that we can manipulate more efficiently, similar to [32].

*2.3.3 Linear System Solving.* We describe our algorithm for solving Eulerian Laplacian systems of equations. It begins with a similar template to the Peng-Spielman solver [32] described in Section 1.1.3, but with modifications to accommodate our non-symmetric setting. Given a linear system in an Eulerian Laplacian $\mathcal{L} = \mathbf{D} - \mathbf{A}^\top$, we write $\mathcal{L} = \mathbf{D}^{1/2}(\mathbf{I} - \mathcal{A})\mathbf{D}^{1/2}$, where $\mathcal{A} = \mathbf{D}^{-1/2}\mathbf{A}^\top\mathbf{D}^{-1/2}$. This reduces the problem to solving linear systems in $\mathbf{L} = \mathbf{I} - \mathcal{A}$ where we can show that $\|\mathcal{A}\|_2 = 1$. We then apply the expansion in Equation (1.2), but with some slight modifications:

- We find it convenient to build up the product expansion in Equation (1.2) recursively. We do so using the identity
$$(\mathbf{I} - \mathcal{A})^+ = (\mathbf{I} - \mathcal{A}^2)^+(\mathbf{I} + \mathcal{A}), \tag{2.1}$$
which can be thought of as a matrix analogue of the rational function identity
$$\frac{1}{1-z} = \frac{1+z}{1-z^2}.$$
Applying this identity repeatedly gives
$$\begin{aligned}(\mathbf{I} - \mathcal{A})^+ &= (\mathbf{I} - \mathcal{A}^2)^+(\mathbf{I} + \mathcal{A}) = (\mathbf{I} - \mathcal{A}^4)^+(\mathbf{I} + \mathcal{A}^2)(\mathbf{I} + \mathcal{A})\\ &= (\mathbf{I} - \mathcal{A}^8)^+(\mathbf{I} + \mathcal{A}^4)(\mathbf{I} + \mathcal{A}^2)(\mathbf{I} + \mathcal{A}) = \ldots.\end{aligned}$$
After $k$ applications of the identity, this yields the first $k$ terms of the product expansion in (1.2) times $(\mathbf{I}-\mathcal{A}^{2^k})^+$, which converges to the identity as $k$ gets large if $\|\mathcal{A}\|_2 < 1$. Some advantages of this compared to the infinite product expansion are that it gives an exact expression rather than an asymptotic result, which will be more convenient to work with when analyzing the growth of errors, and that the pseudoinverses in the expression gives a correct answer when $\|\mathcal{A}\| = 1$, which decreases the extent to which we need to explicitly handle the kernel of $\mathcal{L}$ as a special case.
- If $z \neq 1$ is a complex number with $|z| = 1$, $1/(1-z)$ exists but the series $1/(1-z) = 1 + z + z^2 + \ldots$ does not converge, and our matrix expansion will exhibit similar behavior. Graph theoretically, this case corresponds periodic behavior in the random walk, and we deal with it, as usual, by adding self-loops and working with a lazy random walk. Algebraically, we work with a convex combination with the identity,
$$\mathcal{A}^{(\alpha)} = \alpha\mathbf{I} + (1-\alpha)\mathcal{A},$$
and we note that $\mathbf{I} - \mathcal{A}^{(\alpha)} = (1-\alpha)(\mathbf{I} - \mathcal{A})$. We then replace the identity in Equation (2.1) with the modified identity
$$\begin{aligned}(\mathbf{I} - \mathcal{A})^+ &= (1-\alpha)\left(\mathbf{I} - \mathcal{A}^{(\alpha)}\right)^+\\ &= (1-\alpha)\left(\mathbf{I} - \mathcal{A}^{(\alpha)2}\right)^+\left(\mathbf{I} + \mathcal{A}^{(\alpha)}\right),\end{aligned} \tag{2.2}$$
which leads to better convergence behavior. This step insures that each application of the identity causes a change that is more gradual than squaring. Moreover, our analysis takes advantage of the fact that taking a linear combination with the identity makes it easier to relate $\mathbf{I} - \mathcal{A}^{(\alpha)}_{j+1}$ to $\mathbf{I} - \mathcal{A}^{(\alpha)}_j$. While it may not be necessary to do at every step, it is used to simplify the current analysis. Note, that this algebraic simplification through 'lazy' random walks is also present in other works involving squaring [7, 18].

Similarly to the approach in [32], our strategy is to repeatedly apply (2.2), but to replace $(\mathcal{A}^{(\alpha)})^2$ with a sparsifier in each step to allow us to decrease the computational costs. More precisely, we show how to efficiently construct a sequence of matrices $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_d$ and associated matrices $\mathcal{A}^{(\alpha)}_i = \alpha\mathbf{I} + (1-\alpha)\mathcal{A}_i$ such that each matrix in the sequence has $\widetilde{O}(n/\epsilon^2)$ nonzero entries, $\mathbf{I} - \mathcal{A}_0$ is an $\epsilon$-approximation of $\mathbf{I} - \mathcal{A}$, and $\mathbf{I} - \mathcal{A}_i$ is an $\epsilon$-approximation of $\mathbf{I} - (\mathcal{A}^{(\alpha)}_{i-1})^2$ for each $i \geq 1$ (note that we set $\mathcal{A}_0$ by sparsifying the original Laplacian). We call this a *square-sparsification chain*. We show how to compute all of the matrices in such a chain in time $\widetilde{O}(\mathrm{nnz}(\mathcal{L}) + n\epsilon^{-2}d)$, which we note is within logarithmic factors of the total number of nonzero entries.

The length of the chain is then dictated by the condition number $\kappa = \kappa(\mathbf{U}_{\mathbf{I}-\mathcal{A}})$, the condition number of the symmetric Laplacian associated with the input Eulerian Laplacian. Note that $\kappa = O(\mathrm{poly}(nU))$ where $U \stackrel{\text{def}}{=} \max_{i,j}|\mathbf{A}_{ij}|/\min_{i,j:\mathbf{A}_{ij}\neq 0}|\mathbf{A}_{ij}|$ and may be smaller. If we set $d = \Omega(\log\kappa)$, we show that $\mathbf{I} - \mathcal{A}^{(\alpha)}_d$ well-conditioned. We can thus stop our recursion at this point and (approximately) apply $(\mathbf{I} - \mathcal{A}^{(\alpha)}_d)^+$ using a small number of iterations of a standard iterative method, Richardson iteration.

Expanding the recurrence in (2.2) gives
$$\begin{aligned}(\mathbf{I} - \mathcal{A}_i)^+ \quad &\approx (1-\alpha)^{j-i}\left(\mathbf{I} - \mathcal{A}^{(\alpha)}_j\right)^+\left(\mathbf{I} + \mathcal{A}^{(\alpha)}_{j-1}\right)\\ &\quad\cdot\left(\mathbf{I} + \mathcal{A}^{(\alpha)}_{j-2}\right)\cdots\left(\mathbf{I} + \mathcal{A}^{(\alpha)}_i\right).\end{aligned} \tag{2.3}$$
If we have already computed the matrices in the chain, we can apply the right-hand side to a vector $\vec{b}$ by performing $(j-i)$ matrix-vector multiplications and solving a linear system in $\mathbf{I} - \mathcal{A}^{(\alpha)}_j$. It is useful to think of this as an approximate reduction from applying

$(\mathbf{I} - \mathcal{A}_i)^+$ to applying $(\mathbf{I} - \mathcal{A}_j^{(\alpha)})^+$. The matrices in (2.3) have at most $\widetilde{O}(n/\epsilon^2)$ nonzero entries, so the total time for the matrix vector multiplications is then at most $\widetilde{O}\left((j-i)n\epsilon^{-2}\right)$.

Because of the errors introduced by the sparsification steps, the right-hand side of (2.3) is only an approximation of $(\mathbf{I} - \mathcal{A}_i)^+$, so applying it directly to $\vec{b}$ only yields a (typically somewhat crude) approximation to solution to $(\mathbf{I} - \mathcal{A}_i)\vec{x} = \vec{b}$. To obtain a better solution, we instead use it as a preconditioner inside an iterative method for the linear system. This allows us to obtain an arbitrarily good solution to the system, and the quality of the approximation in (2.3) then determines the number of iterations required.

This suggests that we quantify the error in our approximations using a notion that directly bounds the convergence rate of such a preconditioned iterative method. We do so with the notion of an $\epsilon$-approximate pseudoinverse (defined with respect to some PSD matrix $\mathbf{U}$). Roughly speaking, solving a linear system with an appropriate iterative method using such a matrix as a preconditioner will guarantee the $\mathbf{U}$-norm of the error to decrease by a factor of $\epsilon$ in each iteration. We note that this is only useful for $\epsilon < 1$. For technical reasons, we measure the quality of approximate pseudoinverses with respect to different $\mathbf{U}$ matrices at different stages of the algorithm and translate between them. For simplicity, we just refer to an "$\epsilon$-approximate pseudoinverse" in this overview, but in our algorithm we set the value of $\epsilon$ in our sparsification routines and apply iterative methods, again Richardson iterations, to appropriately pay for the costs of translating between norms.

To analyze the errors introduced by sparsification, we therefore need to:

(1) Relate our notion of graph approximation to approximate pseudoinverses, and
(2) Bound the rate at which the quality of the approximate pseudoinverse we produce decreases as we increase the number of terms in (2.3). We use (2.3) recursively, so it is also useful to bound how this is affected if we use an approximate pseudoinverse instead of the exact operator $\left(\mathbf{I} - \mathcal{A}_j^{(\alpha)}\right)^+$.

For the former, we show that our notion of an $\epsilon$-sparsifier leads to an $O(\epsilon)$-approximate pseudoinverse. For the latter, we show that using a square-sparsifier chain of length $d$ with some given $\epsilon$, and using an $\epsilon'$-approximate pseudoinverse of $\left(\mathbf{I} - \mathcal{A}_j^{(\alpha)}\right)$ in place of $\left(\mathbf{I} - \mathcal{A}_j^{(\alpha)}\right)^+$, produces an $(\epsilon+\epsilon')\cdot 2^{O(d)}$-approximate pseudoinverse for $\mathbf{I} - \mathcal{A}_j^{(\alpha)}$.

The exponential dependence of the error on length of the chain is a key difference between our analysis and the undirected case, and it is what prevents us from having a simpler and more efficient algorithm. If the dependence on the chain length were polynomial, applying (2.3) with $i = 0$ and $j = d$ would provide an $\epsilon \cdot \text{polylog}(n)$-approximate pseudoinverse. We could thus set $\epsilon = 1/\text{polylog}(\kappa)$ in our sparsifier chain and get an $O(1)$-approximate pseudoinverse in $\widetilde{O}(n)$ time. An iterative method could then call this $\log(1/\delta)$ times to obtain a solution with error $\delta$. However, because of the exponential dependence on the chain length, we would only get an $\epsilon \cdot \text{poly}(\kappa)$-approximate pseudoinverse. We would thus need to set $\epsilon = 1/\text{poly}(\kappa)$ to get a value less than 1, which would lead to "sparsifiers" with $\widetilde{\Omega}(n \cdot \text{poly}(\kappa))$ edges. In the typical case where $\kappa = \text{poly}(n)$, simply writing these down would exceed the desired almost-linear time bound.

To prevent this, we do not wait until the end to apply an iterative method to reduce the error. Instead, we break our sparsification and squaring steps into $\lceil d/\Delta \rceil$ blocks of size $\Delta \ll d$, each of which we will wrap in several steps of Richardson iteration, in order to keep the error under control.

Our algorithm first computes (once, not recursively) a square-sparsifier chain of length $d = O(\log \kappa)$ in which the sparsifiers are $\epsilon_{\text{spar}}$-approximations. It then recursively combines two types of steps that are suggested by the discussion above:

- **High error $\left(\mathbf{I} - \mathcal{A}_i^{(\alpha)}\right)^+$ from low error $\left(\mathbf{I} - \mathcal{A}_{i+\Delta}^{(\alpha)}\right)^+$**: Given a routine to apply an $\epsilon_{\text{lo}}$-approximate pseudoinverse of $\mathbf{I} - \mathcal{A}_{i+\Delta}^{(\alpha)}$ in time $\mathcal{T}_{i+\Delta,\epsilon_{\text{lo}}}$, we can use the expansion in (2.3) to apply an $\epsilon_{\text{hi}}$-approximate pseudoinverse of $\mathbf{I} - \mathcal{A}_i^{(\alpha)}$ in time $\mathcal{T}_{i,\epsilon_{\text{hi}}} = \mathcal{T}_{i+\Delta,\epsilon_{\text{lo}}} + \widetilde{O}(\Delta n \epsilon_{\text{spar}}^{-2})$, where $\epsilon_{\text{hi}} = (\epsilon_{\text{spar}} + \epsilon_{\text{lo}})2^{O(\Delta)}$.
- **Low error $\left(\mathbf{I} - \mathcal{A}_i^{(\alpha)}\right)^+$ from high error $\left(\mathbf{I} - \mathcal{A}_i^{(\alpha)}\right)^+$**: By running Richardson iteration for $O(\log \epsilon_{\text{lo}} / \log \epsilon_{\text{hi}})$ steps, we can turn an $\epsilon_{\text{hi}}$-approximate pseudoinverse of $\mathbf{I} - \mathcal{A}_i^{(\alpha)}$ into a $\epsilon_{\text{lo}}$-approximate pseudoinverse. This applies the former once in each iteration, so it takes time

$$\begin{aligned}\mathcal{T}_{i,\epsilon_{\text{lo}}} &= O\left(\frac{\log \epsilon_{\text{lo}}}{\log \epsilon_{\text{hi}}}\right)\mathcal{T}_{i,\epsilon_{\text{hi}}} \\ &= O\left(\frac{\log \epsilon_{\text{lo}}}{\log \epsilon_{\text{hi}}}\right)\left(\mathcal{T}_{i+\Delta,\epsilon_{\text{lo}}} + \widetilde{O}\left(\Delta n \epsilon_{\text{spar}}^{-2}\right)\right). \end{aligned} \tag{2.4}$$

If we set $\epsilon_{\text{hi}}$ to be a constant (say, 1/10), we get $\epsilon_{\text{spar}} + \epsilon_{\text{lo}} = 2^{-\Omega(\Delta)}$, so we set $\epsilon_{\text{spar}} = \epsilon_{\text{lo}} = 2^{-\Theta(\Delta)}$, and (2.4) simplifies to

$$\begin{aligned}\mathcal{T}_{i,\epsilon_{\text{lo}}} &= O(\Delta)\left(\mathcal{T}_{i+\Delta,\epsilon_{\text{lo}}} + \widetilde{O}\left(\Delta n 2^{\Theta(\Delta)}\right)\right) \\ &= O(\Delta)\,\mathcal{T}_{i+\Delta,\epsilon_{\text{lo}}} + \widetilde{O}\left(n 2^{\Theta(\Delta)}\right).\end{aligned}$$

For the base case of our recurrence, $\mathbf{I} - \mathcal{A}_d^{(\alpha)}$ is well-conditioned, so we can approximately apply its pseudoinverse using a standard iterative method in time $\mathcal{T}_{d,\epsilon_{\text{lo}}} = \widetilde{O}(\text{nnz}(\mathcal{A}_d^{(\alpha)})\log \epsilon_{\text{lo}}^{-1}) = \widetilde{O}(n\epsilon_{\text{spar}}^{-2}\log \epsilon_{\text{lo}}^{-1}) = \widetilde{O}\left(n 2^{\Theta(\Delta)}\right)$. This can be folded into the additive $\widetilde{O}\left(n 2^{\Theta(\Delta)}\right)$ term in the recurrence, so it does not significantly affect the time bound.

To estimate the solution to the recurrence, we note that depth of the recursion is $\lceil d/\Delta \rceil$, and at each stage we multiply by $O(\Delta)$. We can think of this as producing a recursion tree with $O(\Delta)^{\lceil d/\Delta \rceil}$ nodes, and we add $\widetilde{O}\left(n 2^{\Theta(\Delta)}\right)$ at each, so we get that

$$\begin{aligned}\mathcal{T}_{0,\epsilon_{\text{lo}}} &= O(\Delta)^{\lceil d/\Delta \rceil}\widetilde{O}\left(n 2^{O(\Delta)}\right) \\ &= nO(\Delta)^{O(d/\Delta)}2^{O(\Delta)} = n 2^{O\left(\Delta + \frac{d\log\Delta}{\Delta}\right)}.\end{aligned}$$

Setting $\Delta = \sqrt{d \log d} = \sqrt{\log \kappa \log\log \kappa}$ approximately balances the two terms in the exponent. Plugging this in and adding the $\widetilde{O}(m)$ for the overhead from the non-recursive parts of the algorithm gives our running time bound of $\widetilde{O}(m) + n 2^{O(\sqrt{\log \kappa \log\log \kappa})}$.

## ACKNOWLEDGMENTS

MC was supported by the National Science Foundation under Grant No. 1111109 and Grant No. 1553428, and by the National Defense Science and Engineering Graduate Fellowship. JK was supported by the National Science Foundation under Grant No. 1111109. JP was supported by the National Science Foundation Graduate Research Fellowship under Grant No. 1122374 and by the National Science Foundation under Grant No. 1065125. RP was supported by the National Science Foundation under Grant No. 1637566. AV was supported by the National Science Foundation under Grant No. 1111109 and Grant No. 1553428.

## REFERENCES

[1] Ittai Abraham, David Durfee, Ioannis Koutis, Sebastian Krinninger, and Richard Peng. 2016. On Fully Dynamic Graph Sparsifiers. *CoRR* abs/1604.02094 (2016). Available at: http://arxiv.org/abs/1604.02094.

[2] Reid Andersen, Fan Chung, and Kevin Lang. 2007. Local partitioning for directed graphs using PageRank. In *International Workshop on Algorithms and Models for the Web-Graph*. Springer, 166–178.

[3] Gely P Basharin, Amy N Langville, and Valeriy A Naumov. 2004. The life and work of AA Markov. *Linear Algebra Appl.* 386 (2004), 3–26.

[4] Joshua Batson, Daniel A. Spielman, and Nikhil Srivastava. 2012. Twice-Ramanujan sparsifiers. *SIAM J. Comput.* 41, 6 (2012), 1704–1721.

[5] Joshua Batson, Daniel A. Spielman, Nikhil Srivastava, and Shang-Hua Teng. 2013. Spectral sparsification of graphs: theory and algorithms. *Commun. ACM* 56, 8 (Aug. 2013), 87–94. DOI: https://doi.org/10.1145/2492007.2492029

[6] András A. Benczúr and David R. Karger. 1996. Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of computing (STOC '96)*. ACM, New York, NY, USA, 47–55. DOI: https://doi.org/10.1145/237814.237827

[7] Dehua Cheng, Yu Cheng, Yan Liu, Richard Peng, and Shang-Hua Teng. 2015. Efficient Sampling for Gaussian Graphical Models via Spectral Sparsification. *Proceedings of The 28th Conference on Learning Theory* (2015), 364–390. Available at http://jmlr.org/proceedings/papers/v40/Cheng15.pdf.

[8] Fan Chung. 2005. Laplacians and the Cheeger inequality for directed graphs. *Annals of Combinatorics* 9, 1 (2005), 1–19.

[9] Fan Chung and Olivia Simpson. 2013. Solving Linear Systems with Boundary Conditions Using Heat Kernel Pagerank. In *Algorithms and Models for the Web Graph - 10th International Workshop, WAW 2013, Cambridge, MA, USA, December 14-15, 2013, Proceedings*. 203–219.

[10] Fan Chung and Olivia Simpson. 2014. Computing Heat Kernel Pagerank and a Local Clustering Algorithm. In *Combinatorial Algorithms - 25th International Workshop, IWOCA 2014, Duluth, MN, USA, October 15-17, 2014, Revised Selected Papers*. 110–121.

[11] Fan Chung and Wenbo Zhao. 2010. A sharp PageRank algorithm with applications to edge ranking and graph sparsification. In *International Workshop on Algorithms and Models for the Web-Graph*. Springer, 2–14.

[12] Michael B. Cohen, Jonathan A. Kelner, John Peebles, Richard Peng, Anup Rao, Aaron Sidford, and Adrian Vladu. 2016. Almost-Linear-Time Algorithms for Markov Chains and New Spectral Primitives for Directed Graphs. *CoRR* abs/1611.00755 (2016). http://arxiv.org/abs/1611.00755

[13] Michael B. Cohen, Jonathan A. Kelner, John Peebles, Richard Peng, Aaron Sidford, and Adrian Vladu. 2016. Faster Algorithms for Computing the Stationary Distribution, Simulating Random Walks, and More. *arXiv preprint arXiv:1608.03270* (2016).

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

[15] Michael B. Cohen, Aleksander Madry, Piotr Sankowski, and Adrian Vladu. 2016. Negative-Weight Shortest Paths and Unit Capacity Minimum Cost Flow in $\tilde{O}(m^{10/7} \log W)$ Time. *CoRR* abs/1605.01717 (2016). http://arxiv.org/abs/1605.01717

[16] Alina Ene, Gary L. Miller, Jakub Pachocki, and Aaron Sidford. 2016. Routing under balance. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*. 598–611. Available at: https://arxiv.org/abs/1603.09009.

[17] Wai Shing Fung, Ramesh Hariharan, Nicholas J.A. Harvey, and Debmalya Panigrahi. 2011. A General Framework for Graph Sparsification. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing (STOC '11)*. ACM, New York, NY, USA, 71–80. DOI: https://doi.org/10.1145/1993636.1993647 Available at http://arxiv.org/abs/1004.4080.

[18] Gorav Jindal and Pavel Kolev. 2015. An Efficient Parallel Algorithm for Spectral Sparsification of Laplacian and SDDM Matrix Polynomials. *arXiv preprint arXiv:1507.07497* (2015).

[19] David R Karger. 1994. Random sampling in cut, flow, and network design problems. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*. ACM, 648–657.

[20] David R Karger. 2000. Minimum cuts in near-linear time. *Journal of the ACM (JACM)* 47, 1 (2000), 46–76.

[21] David R Karger and Matthew S Levine. 2002. Random sampling in residual graphs. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. ACM, 63–66.

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

[23] Ioannis Koutis, Gary L. Miller, and Richard Peng. 2010. Approaching Optimality for Solving SDD Linear Systems. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science (FOCS '10)*. IEEE Computer Society, Washington, DC, USA, 235–244. DOI: https://doi.org/10.1109/FOCS.2010.29 Available at http://arxiv.org/abs/1003.2958.

[24] Ioannis Koutis, Gary L. Miller, and Richard Peng. 2011. 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)*. IEEE Computer Society, Washington, DC, USA, 590–598. DOI: https://doi.org/10.1109/FOCS.2011.85 Available at http://arxiv.org/abs/1102.4842.

[25] Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A Spielman. 2016. Sparsified Cholesky and multigrid solvers for connection laplacians. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*. ACM, 842–850. Available at http://arxiv.org/abs/1512.01892.

[26] Rasmus Kyng and Sushant Sachdeva. 2016. Approximate Gaussian Elimination for Laplacians: Fast, Sparse, and Simple. *CoRR* abs/1605.02353 (2016). http://arxiv.org/abs/1605.02353

[27] Yin Tat Lee and Aaron Sidford. 2013. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*. IEEE, 147–156.

[28] Yin Tat Lee and Aaron Sidford. 2014. Path Finding Methods for Linear Programming: Solving Linear Programs in $\tilde{O}\sqrt{rank}$ Iterations and Faster Algorithms for Maximum Flow. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*. IEEE, 424–433. Available at http://arxiv.org/abs/1312.6677 and http://arxiv.org/abs/1312.6713.

[29] Yin Tat Lee and He Sun. 2015. Constructing Linear-Sized Spectral Sparsification in Almost-Linear Time. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*. 250–269. DOI: https://doi.org/10.1109/FOCS.2015.24

[30] Aleksander Madry. 2013. Navigating Central Path with Electrical Flows: From Flows to Matchings, and Back. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*. 253–262. DOI: https://doi.org/10.1109/FOCS.2013.35

[31] Aleksander Madry. 2016. Computing Maximum Flow with Augmenting Electrical Flows. *CoRR* abs/1608.06016 (2016). http://arxiv.org/abs/1608.06016

[32] Richard Peng and Daniel A. Spielman. 2014. An Efficient Parallel Solver for SDD Linear Systems. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC '14)*. ACM, New York, NY, USA, 333–342. DOI: https://doi.org/10.1145/2591796.2591832 Available at http://arxiv.org/abs/1311.3286.

[33] Yousef Saad. 2003. *Iterative Methods for Sparse Linear Systems* (2nd ed.). Society for Industrial and Applied Mathematics, Philadelphia, PA, USA. Available at: http://www-users.cs.umn.edu/~saad/toc.pdf.

[34] Daniel A. Spielman and Nikhil Srivastava. 2011. Graph Sparsification by Effective Resistances. *SIAM J. Comput.* 40, 6 (2011), 1913–1926. DOI: https://doi.org/10.1137/080734029 Available at http://arxiv.org/abs/0803.0929.

[35] Daniel A. Spielman and Shang-Hua Teng. 2011. Spectral Sparsification of Graphs. *SIAM J. Comput.* 40, 4 (2011), 981–1025. DOI: https://doi.org/10.1137/08074489X Available at http://arxiv.org/abs/0808.4134.

[36] Daniel A. Spielman and Shang-Hua Teng. 2014. Nearly Linear Time Algorithms for Preconditioning and Solving Symmetric, Diagonally Dominant Linear Systems. *SIAM J. Matrix Anal. Appl.* 35, 3 (2014), 835–885. DOI: https://doi.org/10.1137/090771430 Available at http://arxiv.org/abs/cs/0607105.

[37] Williams J Stewart. 1994. *Introduction to the numerical solutions of Markov chains*. Princeton Univ. Press.

[38] Shang-Hua Teng. 2010. The Laplacian Paradigm: Emerging Algorithms for Massive Graphs. In *Proceedings of the 7th Annual Conference on Theory and Applications of Models of Computation (TAMC'10)*. Springer-Verlag, Berlin, Heidelberg, 2–14. DOI: https://doi.org/10.1007/978-3-642-13562-0_2

[39] Zeyuan Allen Zhu, Zhenyu Liao, and Lorenzo Orecchia. 2015. Spectral Sparsification and Regret Minimization Beyond Matrix Multiplicative Updates. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*. 237–245.