58th Annual IEEE Symposium on Foundations of Computer Science

Matrix Scaling and Balancing via Box Constrained Newton's Method and Interior Point Methods

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

Aleksander Mądry
MIT
Cambridge, USA
madry@mit.edu

Dimitris Tsipras
MIT
Cambridge, USA
tsipras@mit.edu

Adrian Vladu
MIT
Cambridge, USA
avladu@mit.edu

***Abstract*—In this paper[1], we study matrix scaling and balancing, which are fundamental problems in scientific computing, with a long line of work on them that dates back to the 1960s. We provide algorithms for both these problems that, ignoring logarithmic factors involving the dimension of the input matrix and the size of its entries, both run in time $\widetilde{O}\left(m \log \kappa \log^2(1/\varepsilon)\right)$ where ε is the amount of error we are willing to tolerate. Here, κ represents the ratio between the largest and the smallest entries of the optimal scalings. This implies that our algorithms run in nearly-linear time whenever κ is quasi-polynomial, which includes, in particular, the case of strictly positive matrices. We complement our results by providing a separate algorithm that uses an interior-point method and runs in time $\widetilde{O}(m^{3/2} \log(1/\varepsilon))$.**

In order to establish these results, we develop a new second-order optimization framework that enables us to treat both problems in a unified and principled manner. This framework identifies a certain generalization of linear system solving that we can use to efficiently minimize a broad class of functions, which we call *second-order robust*. We then show that in the context of the specific functions capturing matrix scaling and balancing, we can leverage and generalize the work on Laplacian system solving to make the algorithms obtained via this framework very efficient.

***Keywords*-matrix scaling; matrix balancing; Newton's method; interior point methods; SDD solver**

I. Introduction

Matrix balancing and scaling are problems of fundamental importance in scientific computing, as well as in statistics, operations research, image reconstruction, and engineering. The literature on these problems [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15] is truly extensive and dates back to 1960s. They both are key primitives in most mainstream numerical software packages (MATLAB, R, LAPACK, EISPACK) [16], [17], [18], [19], [20]. Also, both these problems can be seen as task in which we are aiming to find diagonal scalings of a given matrix so that the rescaled matrix gains some favorable structure.

More specifically, in the matrix scaling problem, we are given a nonnegative matrix $\mathbf{A}$, and our goal is to find diagonal matrices $\mathbf{X}, \mathbf{Y}$ such that the matrix $\mathbf{XAY}$ has prescribed row and column sums. The most common instance of this problem is the one where we want to scale the matrix so to make it doubly stochastic – in other words, we want to make all row and column sums be equal to one. This procedure has been repeatedly used since as early as 1937 in a number of diverse areas, such as telecommunication [21], engineering [13], statistics [22], [10], machine learning [23], and even computational complexity [24], [25]. A standard application for scaling is preconditioning linear system solving. Given a linear system $\mathbf{A}x = b$, one can produce a solution by computing $\mathbf{Y}(\mathbf{XAY})^{-1}\mathbf{X}b$, since applying the inverse of $\mathbf{XAY}$ is more numerically stable procedure than directly applying the inverse of $\mathbf{A}$ [11]. Another example application, which commonly occurs in statistics, is iterative proportional fitting. This primitive is often used for standardizing cross-tabulations and has been studied since 1912 [26]. Even more interestingly, matrix scaling turned out to have surprising connections to fundamental problems in the theory of computation. Notably, in [24], it is observed that scaling can be used to approximate the permanent of any nonnegative matrix within a multiplicative factor of e^n. Furthermore, deciding whether the permanent of a bipartite graph's adjacency matrix is 0 or at least 1 is equivalent to deciding whether that graph contains a perfect matching. Such scaling–based method can, as a matter of fact, be used to compute maximum matchings in bipartite graphs, which is a classic and intensely studied problem in graph algorithms [27], [28], [29]. For more history and information on this problem, we refer the reader to Idel's extensive survey [15], or [30] for a list of applications.

Now, in the matrix balancing problem, we are, again, given a nonnegative matrix $\mathbf{A}$, and our goal here is to find a diagonal matrix $\mathbf{D}$ such that the matrix $\mathbf{DAD}^{-1}$ is *balanced*, that is the sum of each row is equal to the sum of the corresponding column. This procedure has been introduced first by Osborne [2], who was using it to precondition matrices in order to increase the accuracy of the eigenvalue computation. (Note that the balancing operation does not change the eigenvalues of the matrix.) The initially proposed algorithm for it was based on a simple iterative approach, and was then followed by a long series of improvements and extensions. The initial work on this problem focused on a variant in which one aims to balance ℓ_2-norms of rows and columns. It turns out, however, that the ℓ_1-norm–

[1]The full version of this paper is available as [1].

0272-5428/17 $31.00 © 2017 IEEE
DOI 10.1109/FOCS.2017.88

based version we study here is equivalent. In fact, balancing problems with respect to ℓ_p norms, with constant $p \geq 1$, are all reducible to each other.

A. Previous Work

The early methods used for solving these problems – Osborne's iteration for balancing, and the RAS method for scaling – are simple iterative algorithms. However, merely the task of analyzing their convergence turned out to be a major challenge. Significant effort has gone into understanding their convergence [31], [32], [33], [24], and providing better analyses or better iterative methods resulted in a long line of work in this context.

The major shortcoming of the methods obtained so far for exactly solving the problem (depending only logarithmically on $1/\varepsilon$) is their very large running time. In the following discussion we ommit runtime factors that depend (logarithmically) on the size of the input entries. For matrix scaling, Kalantari and Kachiyan [34] obtained an algorithm that finds an ε-approximate solution and runs in time $\widetilde{O}(n^4 \log(1/\varepsilon))$, where n denotes the dimension of the matrix (we can assume the matrix is square w.l.o.g.) and ε is the desired accuracy parameter. This algorithm was based on the ellipsoid method. These authors also proposed – but not formally analyzed – an algorithm based on interior point method, which they expected to run in time $\widetilde{O}(m^{3.5} \log(1/\varepsilon))$, where m denotes the number of non-zero entries of the input matrix. Then, Nemirovsky and Rothblum [35] analyzed an interior point method–based algorithm which run in time $\widetilde{O}(m^4 \log(1/\varepsilon))$. Finally, Linial, Samorodnitsky, and Wigderson [24] gave an $\widetilde{O}(n^7 \log(1/\varepsilon))$ time algorithm that is also strongly polynomial, in the sense that it does not depend at all on the size of input entries.

For the case of matrix balancing, Parlett and Reinsch [3] provided an iterative method based on Osborne's iteration, without proving convergence. Then, Grad [4] proved that Osborne's iteration converges in the limit. The first polynomial time bound was obtained by Kalantari, Khachiyan, and Shokoufandeh [7], who gave an algorithm with running time $\widetilde{O}(n^4 \log(1/\varepsilon))$.

Alternatively, if one is interested in the regime where the running time is allowed to depend polynomially – instead of logarithmically – on the (inverse of the) desired accuracy of the solution, there are algorithms that have an even better dependence on the other parameters. Specifically, the current state-of-the-art is given by Linial, Samorodnitsky, and Wigderson [24], who obtain $O(n^3 \varepsilon^{-2})$ running time for the scaling problem. In the case of the balancing problem, recently, Ostrovsky, Rabani, and Yousefi [33] made a significant progress by obtaining running times of $\widetilde{O}(m + n\varepsilon^{-2})$ and $\widetilde{O}(n^{3.5}\varepsilon^{-1})$.

Finally, another important line of work in this domain was focused on the related ℓ_∞ variant of the balancing problem, where the maximum entry of each row is required to be equal to the maximum entry of the corresponding column. Schneider and Schneider [8] gave a non-iterative algorithm running in time $O(n^4)$, improved to $\widetilde{O}(mn+n^2)$ by Young, Tarjan, and Orlin [36]. More recently, Schulman and Sinclair [32] provided an analysis of the classical Osborne-Parlett-Reinsch obtaining a running time of $\widetilde{O}(n^2 m)$, and gave a version of it with running time $\widetilde{O}(n^3 \log(1/\varepsilon))$.

B. Our Contributions

We provide algorithms for both matrix scaling and balancing problems.

For the matrix scaling problem, we establish an algorithm that runs in time

$$\widetilde{O}\left(m \log(\kappa(\mathbf{U}^*) + \kappa(\mathbf{V}^*)) \log^2 \frac{s_{\mathbf{A}}}{\varepsilon}\right),$$

where $\mathbf{U}^*$ and $\mathbf{V}^*$ are the optimal scaling matrices, $\kappa(\cdot)$ is the maximum ratio between the diagonal entries of its argument, $s_{\mathbf{A}}$ is the sum of the entries in the input matrix, and ε is the measure of the target error of the scaling, formally defined in Definition 5.

For the matrix balancing problem, we establish a running time of

$$\widetilde{O}\left(m \log \kappa(\mathbf{D}^*) \log^2 \frac{w_{\mathbf{A}}}{\varepsilon}\right),$$

where $w_{\mathbf{A}}$ is the ratio of the sum of the entries to the minimum nonzero entry, $\mathbf{D}^*$ is the optimal balancing matrix, $\kappa(\cdot)$ has the same meaning as above, and ε is the measure of the balancing error, as formally defined in Definition 19.

Notably, our running times depend logarithmically on both the target accuracy and the magnitude of the entries in the optimal balancing or scaling. This implies that if the optimal solution has quasi-polynomially bounded entries, our algorithms run in nearly linear time $\widetilde{O}(m \log(1/\varepsilon))$ (ignoring logarithmic factors involving the entries of the input matrix). This includes, for instance, the case when input matrix has all its entries positive or, in case of matrix balancing, if there just exists a single row/column pair with all positive entries.

However, there are matrices for which κ can be exponentially large (in n). For the case of such matrices we develop algorithms with negligible dependence on κ. These algorithms are based on interior point methods, with appropriately chosen barriers, commonly used in exponential programming [37]. We show that the linear system solves required by the interior point method every iteration can be reduced via Schur complementing to approximately solving a Laplacian system, which can be done in nearly linear time using any standard Laplacian solver [38], [39], [40], [41], [42], [43], [44]. This yields a running time of

$$\widetilde{O}\left(m^{3/2} \log \frac{w_{\mathbf{A}}}{\varepsilon}\right),$$

where $w_{\mathbf{A}}$ is the ratio between the largest and smallest nonzero entry of $\mathbf{A}$.

C. Our Approach

We approach the scaling and balancing problems by developing a continuous optimization based perspective on them. More precisely, we solve both matrix scaling and balancing problems by casting them as tasks of minimizing certain corresponding convex functions. In fact, in the case of the balancing problem, that function is directly inspired by the one used in [7]; for the scaling problem, it is function derived from the one used in [34].

Since our goal is to obtain logarithmic – instead of polynomial – dependence on the (inverse of the) desired accuracy ε, it would be tempting to use well-known tools for convex programing, such as ellipsoid method or interior point method. However, these methods are, a priori, computationally expensive. This motivates us to look for different, more direct approaches.

To this end, we develop a technique for minimizing a broader class of functions that we call *second-order robust (with respect to ℓ_∞)*. Intuitively, this class corresponds to functions whose Hessians do not change too much within any unit ℓ_∞-ball. And the consequence of that property that will be crucial for us is that local quadratic approximation of such functions at any given point is relatively accurate within the unit ℓ_∞ neighborhood of that point. As a result, iteratively optimizing the local approximation around the current point, while staying within that ℓ_∞ neighborhood, will be guaranteed to make progress towards minimizing the function. This iterative procedure can be viewed as a "box-constrained" variant of the Newton's method.

A priori, performing a single step of such a box-constrained Newton's method, i.e., minimizing a quadratic function subject to box constraints might be a computationally costly task. We show, however, that it suffices to implement a weaker primitive, which we call a *k-oracle*. That primitive corresponds to (approximately) minimizing a quadratic function within a region that is within a factor of k larger than the target ℓ_∞-ball. Once such a k-oracle is implemented efficiently, we can compute the global optimum of our second-order robust function using a small number of calls to it. More precisely, we show that one can minimize a convex function f that is second-order robust with respect to ℓ_∞ to within ε additive error from optimum in

$$O\left((kR_\infty + 1)\log\left(\frac{f(x_0) - f(x^*)}{\varepsilon}\right)\right) \qquad (1)$$

iterations, where each iteration consists of one call to the k-oracle, x_0 is the starting point, x^* is the minimizer of f, and R_∞ is the ℓ_∞ radius of the level set of x_0.

In the light of the above, the main technical difficulty remaining is obtaining an efficient implementation of a k-oracle. We show that for functions whose Hessian is symmetrically diagonally dominant, with nonzero off-diagonal entries, or SDD for short[2], we can implement a k-oracle, with $k = O(\log n)$, in time that is nearly linear in the sparsity of the Hessian. We build here on the strategy underlying the Laplacian solver of Lee, Peng and Spielman [45]. Specifically, we carefully lift the solutions corresponding to coarser (and smaller) approximations of the underlying matrix to the desired solutions corresponding to the initial matrix in a way that does not allow these lifted solutions to exceed the boundaries of a $O(\log n)$-radius ℓ_∞-ball.

Once the above optimization framework is developed, applying it to the scaling and balancing problems is fairly straightforward. It boils down to verifying that the functions that capture the respective problems are indeed second-order robust and have an SDD Hessian, and then bounding all the relevant quantities that (1) involves.

Independent Work: Finally, we note that Allen-Zhu, Li, Oliveira, and Wigderson [46] obtained independently very similar results for the exact version of the problem. The running time of the algorithms they develop have a bit worse dependence on m, but they were able to establish better absolute bounds on κ (in terms of the problem parameters and the magnitude of the input entries) for the general, non-doubly stochastic variant of the matrix scaling problem.

D. Roadmap

The rest of the paper is organized as follows. First, we introduce relevant notation and concepts in Section II. Then, in Section III we formally introduce the class of convex functions we call *second-order robust with respect to ℓ_∞*. For these, we develop a specific optimization primitive called *box-constrained Newton method*.

We describe how we can apply the primitive from Section III to matrix balancing and scaling in Section IV, by reducing these problem to a convex function minimization with favorable structure. In order to complete our algorithm, in Section V, we show how to efficiently implement an iteration of the box-constrained Newton in the special case where the Hessian of the function is SDD. In Section VI we provide a different approach for balancing and scaling based on interior point methods. Complete proofs and technical details are presented in the full version of the paper.

II. Preliminaries

A. Notations

Vectors: We let $\vec{0}, \vec{1} \in \mathbb{R}^n$ denote the all zeros and all ones vectors, respectively. When it is clear from the context, we apply scalar operations to vectors with the interpretation that they are applied coordinate-wise.

[2]Such matrices can essentially be viewed as a Laplacian matrix plus a nonnegative diagonal.

Matrices: We write matrices in bold. We use $\mathbf{I}$ to denote the identity matrix, and $\mathbf{0}$ to denote the zero matrix. Given a matrix $\mathbf{A}$, we denote its number of nonzero entries by $\text{nnz}(\mathbf{A})$. When it is clear from the context, we use m to denote the the number of nonzeros; similarly, we use n to denote the dimension of the ambient space.

We denote by $s_{\mathbf{A}}$ the sum of entries of $\mathbf{A}$, by $\ell_{\mathbf{A}}$ the minimum nonzero entry of $\mathbf{A}$, and by $w_{\mathbf{A}}$ the ratio between these quantities. We use $\text{supp}(\mathbf{A})$ to denote the set of pairs of indices (i, j) corresponding to the nonzero entries of $\mathbf{A}$. Given a matrix $\mathbf{A}$, we define $r_{\mathbf{A}} = \mathbf{A}\vec{1}$ to be the vector consisting of row sums, and $c_{\mathbf{A}} = \mathbf{A}^\top\vec{1}$ to be the vector consisting of column sums. For a positive diagonal matrix $\mathbf{A}$ we denote the maximum ratio between its diagonal elements by $\kappa(\mathbf{A})$.

Positive Semidefinite Ordering and Approximation: For symmetric matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n\times n}$ we use $\mathbf{A} \preceq \mathbf{B}$ to represent the fact that that $x^\top\mathbf{A}x \leq x^\top\mathbf{B}x$, for all x. A symmetric matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ is positive semidefinite (PSD) if $\mathbf{A} \succeq 0$. We use $\preceq, \succ, \prec$ in a similar fashion. For vectors x, we define the norm $\|x\|_{\mathbf{A}} = \sqrt{x^\top\mathbf{A}x}$. Given two PSD matrices $\mathbf{A}$ and $\mathbf{B}$, and a parameter $\alpha > 0$, we use $\mathbf{A} \approx_\alpha \mathbf{B}$ to denote the fact that $e^{-\alpha} \cdot \mathbf{B} \preceq \mathbf{A} \preceq e^{\alpha} \cdot \mathbf{B}$.

Laplacian and SDD matrices: A family of matrices that will play an important role in this paper are symmetric diagonally dominant (SDD) matrices. These are matrices $\mathbf{A}$, that symmetric and, moreover, have each diagonal entry be larger than the sum of absolute values of the corresponding row entries. That is, for every i

$$A_{ii} \geq \sum_{j\neq i} |A_{ij}|.$$

A special case of SDD matrices are Laplacian matrices, which have negative off-diagonal entries and the sum of each row is required to be zero. The crucial fact about these matrices is that one can exploit their structure to solve linear systems in them in time that is only nearly linear [38], [39], [40], [41], [42], [43], [44].

Diagonal Matrices: For $x \in \mathbb{R}^n$ we denote by $\mathbb{D}(x) \in \mathbb{R}^{n\times n}$ the diagonal matrix where $\mathbb{D}(x)_{ii} = x_i$. Given a nonnegative diagonal matrix $\mathbf{D}$, we use $\kappa(\mathbf{D})$ to denote the ratio between its largest and smallest entry. We will overload notation and, for any matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, use $\mathbb{D}(\mathbf{A})$ to denote the main diagonal of $\mathbf{A}$, that is $(\mathbb{D}(\mathbf{A}))_{ii} = \mathbf{A}_{ii}$ and $(\mathbb{D}(\mathbf{A}))_{ij} = 0$ for $i \neq j$.

Gradients and Hessians: Given a function f we denote by $\nabla f(x)$ its gradient at x, and by $\nabla^2 f(x)$ its Hessian at x. When the function is clear from the context, we also use $\mathbf{H}_x$ to denote its Hessian at x.

Block Matrices: As part of our algorithms, we will consider partitioning the coordinates of vectors into sets of indices F and C. When we compute the quadratic form of a matrix with these vectors, we need to be able to reason about how values in each component interact with the rest of the vector. For that reason it is convenient to denote the block form notation for a matrix $\mathbf{A}$ as:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{[F,F]} & \mathbf{A}_{[F,C]} \\ \mathbf{A}_{[C,F]} & \mathbf{A}_{[C,C]} \end{bmatrix}.$$

Schur Complements: For a matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ and a partition of its indices (F, C), the Schur complement of F in $\mathbf{A}$ is defined as

$$\text{Sc}(\mathbf{A}, F) \stackrel{\text{def}}{=} \mathbf{A}_{[C,C]} - \mathbf{A}_{[C,F]}\mathbf{A}^{-1}_{[F,F]}\mathbf{A}_{[F,C]}\,.$$

The exact use of Schur complements will become clear in Sections V,VI. These are objects that naturally arise during Gaussian elimination for the solution of linear systems. By pivoting out variables F the remaining system to solve for variables of C is exactly the Schur complement of F in $\mathbf{A}$.

III. Box-Constrained Newton Method for Second-Order Robust Functions

The central element of our approach is developing an efficient second-order method based minimization framework for a broad class of functions that we will call second-order robust with respect to ℓ_∞. To motivate the choice of this class, recall that second-order methods for function minimization are iterative in nature, and they boil down to repeated minimizing the local quadratic approximation of the function around the current point. Consequently, in order to obtain meaningful guarantees about the progress made by such methods, one needs to ensure that this local quadratic approximation constitutes a good approximation of the function not only at the current point but also in a reasonably large neighborhood of that point. The most natural way to obtain such a guarantee is to ensure that the Hessian of the function (which is the basis of our local quadratic approximations) does not change by more than a constant factor in that neighborhood. As a result, the functions we are interested in optimizing in this paper are the ones that satisfy that property in an ℓ_∞-ball around the current point. This is formalized in the following definition.

Definition 1 (Second-Order Robust w.r.t. ℓ_∞)**.** We say that a convex function $f : \mathbb{R}^n \to \mathbb{R}$ is *second-order robust (SOR) with respect to* ℓ_∞ if, for any $x, y \in \mathbb{R}^n$ such that $\|x - y\|_\infty \leq 1$,

$$\nabla^2 f(x) \approx_2 \nabla^2 f(y), \text{ that is,}$$

$$\frac{1}{e^2}\nabla^2 f(x) \preccurlyeq \nabla^2 f(y) \preccurlyeq e^2\nabla^2 f(x)\,.$$

Note that the size of the ℓ_∞-ball, as well as the exact factor by which the Hessian is allowed to change, are chosen somewhat arbitrarily – all choices of the constants can be made equivalent via an appropriate rescaling. Moreover, even if these quantities are not constant, they would only appear in the running time as a small polynomial factor.

Now, the above definition suggests a natural framework for optimizing such functions. Namely, in every iteration,

we optimize a local quadratic approximation of the function within a unit ℓ_∞-ball around the current point. As we will see shortly, this approach can be rigorously analyzed. In particular, our key technical result is that if we apply this approach to an SOR function whose Hessians has additionally a special structure, i.e., those for which the Hessian is, essentially, a symmetric diagonally dominant (SDD) matrix, we can implement every iteration in time nearly linear in the number of nonzero entries of the Hessian. This leads to running time bounds captured by the following theorem.

Theorem 2 (Minimizing Second-Order Robust Functions w.r.t ℓ_∞)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a second-order robust (SOR) function with respect to ℓ_∞, such that its Hessian is symmetric diagonally dominant (SDD) with nonpositive off-diagonals, and has m nonzero entries. Given a starting point $x_0 \in \mathbb{R}^n$ we can compute a point x, such that $f(x) - f(x^*) \leq \varepsilon$, in time*

$$\widetilde{O}\left((m+T)R_\infty \log\left(\frac{f(x_0)-f(x^*)}{\varepsilon}\right)\right),$$

where x^* is a minimizer of f, $R_\infty = \sup_{x:f(x)\leq f(x_0)} \|x - x^*\|_\infty$ is the ℓ_∞ diameter of the corresponding level set of f, and T is the time required to compute the Hessian.

Note that the bounds provided by the above theorem are, in a sense, the best possible for any kind of approach that relies on repeated minimization of a local approximation of a function in an ℓ_∞-ball neighborhood. In particular, as each step can make a progress of at most 1 in ℓ_∞-norm towards the optimal solution, one would expect the total number of steps to be $\Omega(R_\infty)$.

It turns out that the above theorem is all we need to establish our results for scaling and balancing problems (except the ones relying on the interior point method). That is, these results can be obtained by direct application of the above theorem to an appropriate SOR function. We provide the details in Section IV.

Now, the first step to proving the above Theorem 2 is to view each iteration of our iterative minimization procedure as a call to a certain oracle problem.

Definition 3. We say that a procedure $\mathcal{O}$ is a k-oracle for a class of matrices $\mathcal{M}$, if on input $(\mathbf{A}, b)$, where $\mathbf{A} \in \mathcal{M} \subseteq \mathbb{R}^{n\times n}$, and $b \in \mathbb{R}^n$, returns a vector $\tilde{z}$ satisfying

1) $\|\tilde{z}\|_\infty \leq k$, and
2) $\frac{1}{2}\tilde{z}^\top \mathbf{A}\tilde{z} + b^\top \tilde{z} \leq \frac{1}{2} \cdot \min_{\|z\|_\infty \leq 1} \left(\frac{1}{2} z^\top \mathbf{A} z + b^\top z\right)$.

Note that the minimum of the left-hand side of Condition (2) above is always non-positive. This is desired, since this expression is supposed to measure our function minimization progress.

Observe that minimizing the function $\frac{1}{2}z^\top \mathbf{A} z + b^\top z$ without any constraints on z corresponds to solving a linear system $\mathbf{A}z = -b$. So, one can view the k-oracle problem as a certain generalization of linear system solving. Specifically, it is a task in which we aim to find a point in the ℓ_∞-ball of diameter k around the origin that is closest (in a certain sense) to the solution to that linear system.

One can view the parameter k as the measure of the "quality" of our k-oracle. The smaller it is, the faster convergence the overall procedure will have. Importantly, however, the value of k impacts only the convergence and not the quality of the final solution. The following theorem makes this relationship precise.

Theorem 4. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function that is second-order robust with respect to ℓ_∞. Let $\mathcal{O}$ be a k-oracle for $\{\nabla^2 f(x) : x \in \mathbb{R}^n\}$, along with an initial point $x_0 \in \mathbb{R}^n$ and an accuracy parameter ε. Let $R_\infty = \sup_{x:f(x)\leq f(x_0)} \|x - x^*\|_\infty$, where x^* is a minimizer of f. Then one can produce a solution x_T satisfying $f(x_T) - f(x^*) \leq \varepsilon$ using*

$$O\left((kR_\infty + 1)\log\left(\frac{f(x_0)-f(x^*)}{\varepsilon}\right)\right)$$

calls to $\mathcal{O}$.

In Section V we design an efficient k-oracle, with $k = O(\log n)$, for the family of SDD matrices. Combining Theorem 38 with Theorem 4 immediately gives the proof of Theorem 2. We remark that while Theorems 2, 4 are stated and proved for functions defined over $\mathbb{R}^n$, they can be extended in a straightforward way to hold when f is defined over an arbitrary closed, convex set.

IV. Matrix Scaling and Balancing

Having developed our main optimization primitives, we can develop efficient algorithms for matrix scaling and matrix balancing. Our approach is essentially the same for both problems, and differs only in technical details.

At the high level, we will construct convex functions with optima corresponding to exact scaling/balancing of the matrix. Moreover, the gradient of these functions at a specific scaling/balancing of the matrix will be directly related to the quality of this particular scaling/balancing. This will allow us to prove that approximately optimal points correspond to ε-balancing/ε-scaling. The fact that that these functions are second-order robust with respect to ℓ_∞ makes it sufficient to apply the optimization method from Section III. To complete the algorithm and its running time analysis, we need then to address two issues.

Firstly, proving running time bounds for this method requires an upper bound on the ℓ_∞ radius of the level set of the initial point, i.e. the R_∞ parameter defined in Theorem 4. Depending on the structure of the matrix, there are several different bounds that one can prove, depending only on parameters of the original problem. However, the most interesting case is when we are promised that the exact scaling/balancing of the matrix is "small" (in the sense that

the *ratio* between factors is, say, polynomial). In that case, we can regularize the function to turn this promise into a guarantee for the size of the level set without sacrificing too much accuracy. Moreover, by using a simple doubling approach, we can make the algorithm not require explicit knowledge of such a parameter, and it will only appear as a factor in the final runnning time of the algorithm.

Secondly we need to ensure that we can efficiently implement k-oracles for the Hessians of these functions. In our case, this boils down to proving that the Hessians are SDD matrices with sparsity equal to that of the input matrix, and then build on the existing Laplacian solving work. For the remainder of this section, we define the convex functions that we need optimize, show how to regularize them, and prove bounds on the corresponding R_∞ parameters. We describe and sketch the analysis for the implementation of a $O(\log n)$-oracle in Section V.

A. Matrix Scaling

We now formally define the scaling problem, along with the notion of ε-scaling.

Definition 5 (Matrix Scaling)**.** Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be a nonnegative matrix and $r, c \in \mathbb{R}^n$ be vectors such that $\sum_{i=1}^n r_i = \sum_{j=1}^n c_i$, and $\|r\|_\infty, \|c\|_\infty \leq 1$[3]. We say that two nonnegative diagonal matrices $\mathbf{X}$ and $\mathbf{Y}$ (r,c)-scale $\mathbf{A}$ if the matrix $\mathbf{M} = \mathbf{XAY}$ satisfies $\mathbf{M}\vec{1} = r$ and $\mathbf{M}^\top\vec{1} = c$, i.e. row i sums to r_i and column j sums to c_j for every i, j.

Definition 6 (ε-(r,c) scaling)**.** Given nonnegative $\mathbf{A}$ and positive diagonal matrices $\mathbf{X}, \mathbf{Y}$, we say that $(\mathbf{X}, \mathbf{Y})$ is an ε-(r,c) scaling (or ε-scaling, when r and c are clear from the context) for matrix $\mathbf{A}$ if the matrix $\mathbf{M} = \mathbf{XAY}$ satisfies

$$\|r_\mathbf{M} - r\|_2^2 + \|c_\mathbf{M} - c\|_2^2 \leq \varepsilon\,.$$

Definition 7 (Scalable and Almost-Scalable Matrices)**.** A nonnegative matrix $\mathbf{A}$, is called (r,c)-scalable, if there exist $\mathbf{X}$ and $\mathbf{Y}$ that (r,c)-scale $\mathbf{A}$. It is called almost (r,c)-scalable if for every $\varepsilon > 0$, there exist $\mathbf{X}_\varepsilon$ and $\mathbf{Y}_\varepsilon$ that ε-(r,c) scale $\mathbf{A}$.

We will cast matrix scaling as a convex optimization problem and show that applying the method from section III yields a good approximate scaling.

Theorem 8. *Let $\mathbf{A}$ be a matrix, that has an (r,c) scaling $(\mathbf{U}^*, \mathbf{V}^*)$. Then, we can compute an ε-(r,c) scaling of $\mathbf{A}$ in time*

$$\widetilde{O}\left(m \log(\kappa(\mathbf{U}^*) + \kappa(\mathbf{V}^*)) \log^2(s_\mathbf{A}/\varepsilon)\right)\,.$$

[3]In literature we also encounter this problem for non-square matrices; however solving squares is sufficient, since given $\mathbf{A} \in \mathbb{R}^{n\times c}$, we can reduce to this instance by scaling the square matrix $\begin{bmatrix}\mathbf{0}_{c,c} & \mathbf{A}^\top \\ \mathbf{A} & \mathbf{0}_{r,r}\end{bmatrix}$. The upper bound on r and c is harmless, since for larger values we can always shrink all of $\mathbf{A}$, r, c and ε by the same factor in order to enforce this constraint.

This implies that if $\mathbf{U}^*$ and $\mathbf{V}^*$ are, say, quasi-polynomially bounded, we can find an approximate scaling in nearly linear time. If fact, we can generalize this statement to obtain a similar result for the case of approximate scalings. This is made precise in Theorem 9.

1) Matrix Scaling via Convex Optimization: Recall that we want to encode the matrix scaling problem as a an instance of minimizing of a certain convex function. Given the input matrix $\mathbf{A}$, the function we want to consider is:

$$f(x,y) = \sum_{1\leq i,j\leq n} \mathbf{A}_{ij} e^{x_i - y_j} - \left(\sum_{1\leq i\leq n} r_i x_i - \sum_{1\leq j\leq n} c_j y_j\right)\,. \tag{2}$$

We want to argue now that computing an (approximate) scaling of the matrix $\mathbf{A}$ can indeed be recovered from an (approximate) minimum of the above function. Specifically, we want to establish the following theorem.

Theorem 9. *Suppose that there exist a point $z_\varepsilon^* = (x_\varepsilon^*, y_\varepsilon^*)$ for which $f(z_\varepsilon^*) - f^* \leq \varepsilon^2/(3n)$ and $\|z_\varepsilon^*\|_\infty \leq B$. Then we can compute an ε-(r,c) scaling of $\mathbf{A}$ in time*

$$\widetilde{O}\left(mB \log^2(s_\mathbf{A}/\varepsilon)\right)\,.$$

The proof is straightforward given the lemmas below and is presented in the full version of the paper. First, we will prove that approximate optimality of f implies an approximate scaling of the matrix.

Lemma 10. *Let $\mathbf{A}$ be an ε-scalable matrix. Let $f^* = \inf_{(x,y)} f(x,y)$. Then, a pair of vectors (x,y) satisfying $f(x,y) - f^* \leq \varepsilon^2/3n$, for $0 < \varepsilon \leq 1$, yields an ε-(r,c) scaling of $\mathbf{A}$:*

$$\mathbf{M} = \mathbb{D}(\exp(x)) \cdot \mathbf{A} \cdot \mathbb{D}(\exp(y))\,.$$

Note that we compare the value of $f(x,y)$ to its infimum, as for the case of almost scalable matrices it is possible that this value is attained only to the limit.

To prove this lemma, we first look at the first and second order derivatives of f.

Lemma 11. *Let $\mathbf{M}$ be the matrix obtained by scaling $\mathbf{A}$ with vectors (x,y), i.e. $\mathbf{M} = \mathbb{D}(\exp(x)) \cdot \mathbf{A} \cdot \mathbb{D}(\exp(y))$. The gradient and Hessian of f satisfy the identities:*

$$\nabla f(x,y) = \begin{bmatrix} r_\mathbf{M} \\ -c_\mathbf{M}\end{bmatrix} - \begin{bmatrix} r \\ -c\end{bmatrix}\,,$$
$$\nabla^2 f(x,y) = \begin{bmatrix} \mathbb{D}(r_\mathbf{M}) & -\mathbf{M} \\ -\mathbf{M}^\top & \mathbb{D}(c_\mathbf{M})\end{bmatrix}\,.$$

We can observe that any (x,y) for which $\nabla f(x,y)$ is equal to 0 yields diagonal matrices that exactly scale $\mathbf{A}$. Moreover, this statement also holds in an approximate sense. One can prove that a large gradient in ℓ_2 norm implies that the current point is far from optimal in function value. Making this statement precise, allows us to prove Lemma 10. The technical details can be found in the full version of the paper.

2) Regularization for Solving via Box-Constrained Newton Method: It is straightforwards to verify that the function we are minimizing (defined in Equation 2), satisfies the requirements necessary for us to be able to apply the tools from Section III.

Lemma 12. *The function f defined in* (2) *is convex, second-order robust with respect to ℓ_∞, and its Hessian is SDD.*

One should observe, however, that Theorem 4 requires bounding the radius of the entire level set containing our initial point and not merely the distance to some (approximate) minimizer of our function f. This means that the existence of an (approximate) minimizer that is close to our initial point is not sufficient to apply Theorem 4. To circumvent that problem, we regularize the function f by adding to it a term that, on one hand, has a relatively small impact on the additive error we can achieve, but, on the other hand, ensures that the entire relevant level set is contained in some sufficiently small ℓ_∞-ball around our initial point. The following lemma makes these statements precise.

Lemma 13. *Let $z^*_\varepsilon = (x^*_\varepsilon, y^*_\varepsilon)$ be a point for which $f(z^*_\varepsilon) - f^* \leq \varepsilon^2/(3n)$ and $\|z^*_\varepsilon\|_\infty \leq B$. Then, the regularization of f defined as*

$$\widetilde{f}(x,y) = f(x,y) + \frac{\varepsilon^2}{36n^2e^B}\left(\sum_i (e^{x_i} + e^{-x_i}) + \sum_j (e^{y_j} - e^{-y_j})\right) \quad (3)$$

satisfies the following properties

1) *$\widetilde{f}$ is second-order robust with respect to ℓ_∞ and its Hessian is SDD,*
2) *$f(z) \leq \widetilde{f}(z)$, and there is a point $\widetilde{z}^*$ such that $\widetilde{f}(\widetilde{z}^*) \leq f^* + \frac{\varepsilon^2}{9n}$,*
3) *for all z' such that $\widetilde{f}(z') \leq \widetilde{f}(0)$, $\|z'\|_\infty = O(B\log(ns_{\mathbf{A}}/\varepsilon))$.*

Theorems 8 and 9 follow from applying Theorem 2 to the regularized function defined in Equation 3, and then combining it with the guarantees of Lemmas 10 and 13. We note that we don't need an explicit knowledge of an a priori bound on B. We can simply run our algorithm repeatedly, doubling our guess at the value of B each time. This will not increasing the overall running time by more than a factor of two.

3) Bounding the Magnitude of the Optimal and Approximately Optimal Scalings for Doubly Stochastic Scaling: In order to provide bounds for the magnitude of the scaling factors that only depend on the parameters of the initial problem, we refer to the following lemmas from [34] for the case of double stochastic (i.e. (1,1)) scaling.

Lemma 14 (Lemma 1 of [34])**.** *If $\mathbf{A}$ is strictly positive, then it can be scaled to doubly stochastic by diagonal matrices $\mathbf{U}$, $\mathbf{V}$ with $\log(\kappa(\mathbf{U}) + \kappa(\mathbf{V})) \leq O(\log(w_{\mathbf{A}}))$.*

Lemma 15 (Corollary 1 of [34])**.** *If $\mathbf{A}$ is scalable, then it can be scaled to doubly stochastic by diagonal matrices $\mathbf{U}$, $\mathbf{V}$ with $\log(\kappa(\mathbf{U}) + \kappa(\mathbf{V})) \leq O(n\log(w_{\mathbf{A}}))$.*

For almost scalable matrices, there can be arbitrarily good solutions, using arbitrarily large scaling factors. To prove bounds on the runtime of finding an approximate doubly-stochastic matrix, we will have to explicitly demonstrate an vector that approximately minimizes function f while having small ℓ_∞ norm.

Lemma 16. *If $\mathbf{A}$ is almost-doubly-stochastic scalable, then there exist points (x, y) such that $f(x, y) - f^* \leq \varepsilon^2/3n$, such that $\|(x, y)\|_\infty \leq O(n\log(nw_{\mathbf{A}}/\varepsilon))$.*

For the general case of (r, c)-scaling we refer to the recent lemmas from the parallel work of [46]. The assumption that the scaling targets are integral is mild, since one can approximate real numbers by rational ones which can then by scaled to be integral (the dependence on this scaling is logarithmic).

Lemma 17 (Lemma 3.3 of [46])**.** *If $\mathbf{A}$ is almost (r, c)-scalable with r, c being integral, then it can be ε-scaled by diagonal matrices $\mathbf{U}$, $\mathbf{V}$ with $\log(\kappa(\mathbf{U}) + \kappa(\mathbf{V})) \leq O(n\log(nw_{\mathbf{A}}\|r\|_1/\varepsilon))$.*

B. Matrix Balancing

Our approach for the balancing problem is completely analogous to the one we used for the scaling problem. There are only minor technical differences. To state them, we first formally define the problem and the notion of approximation we are considering for it.

Definition 18 (Matrix Balancing)**.** Let $\mathbf{A}$ be a square nonnegative matrix. We say that $\mathbf{A}$ is balanced if the sum of each row is equal to the sum of the corresponding column, i.e. $r_{\mathbf{A}} = c_{\mathbf{A}}$. We say that a nonnegative diagonal matrix $\mathbf{D}$ balances $\mathbf{A}$ if the matrix $\mathbf{M} = \mathbf{DAD}^{-1}$ is balanced.

Definition 19 (ε-Balanced Matrix [7])**.** We say that a nonnegative matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$ is ε-balanced if

$$\frac{\|r_{\mathbf{M}} - c_{\mathbf{M}}\|_2}{\sum_{1\leq i,j\leq n}\mathbf{M}_{ij}} = \frac{\sqrt{\sum_{i=1}^n((r_{\mathbf{M}})_i - (c_{\mathbf{M}})_i)^2}}{\sum_{1\leq i,j\leq n}\mathbf{M}_{ij}} \leq \varepsilon.$$

Observe that this definition is invariant to a global scaling of all the entries of the matrix by some factor. There is a very simple condition that characterizes the set of matrices that can be balanced

Lemma 20 ([7])**.** *A nonnegative matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ can be balanced if and only if the graph with adjacency matrix $\mathbf{A}$ is strongly connected.*

In the case when the graph is not strongly connected, the matrix can have its rows and columns rearranged so as to be written as a lower triangular block matrix with strongly connected diagonal blocks. The reason no exact

balancing exists is that off diagonal block elements will always create imbalances. This, however, is not an obstacle for approximately balancing the matrix. Once we balance the diagonal blocks, we can set all of the off-diagonal block entries to a very small value, say ε/n, so that they don't cause significant imbalances. This corresponds to implicitly scaling the block rows and collumns by a very large amount, making the off-diagonal entries arbitrarily close to zero. Therefore, since the case of matrices that cannot be exactly balanced is easy to detect, and can be easily reduced to the exactly balanceable case, from now on we consider only matrices that can be balanced, and therefore represent strongly connected graphs.

We can now state our main theorem for this section, which follows our initial discussion.

Theorem 21. *Let $\mathbf{A}$ be a matrix that can be balanced by the diagonal matrix $\mathbf{D}^*$. Then, we can compute an ε-approximate balancing of $\mathbf{A}$ in time*

$$\widetilde{O}(m \log \kappa(\mathbf{D}^*) \log^2(w_{\mathbf{A}}/\varepsilon)) .$$

This immediately implies that if $\mathbf{D}^*$ is, say, quasi-polynomially conditioned, we can find an approximate balancing in nearly linear time.

Again, we can generalize this result to hold for approximate balancings. We make this statement precise in Theorem 22.

1) Reducing Matrix Balancing to Convex Optimization: Similarly to the case of the scaling problem, we encode this problem as a minimization of an appropriately constructed convex function. The function we consider here is

$$f(x) = \sum_{1 \le i,j \le n} \mathbf{A}_{ij} e^{x_i - x_j}, \tag{4}$$

and this function was already defined in [7]. Similarly to the case of matrix scaling, we will show that (approximately) minimizing this function corresponds to (approximately) balancing the matrix $\mathbf{A}$. For the rest of this section, we will define f_* to be the infimum value of f in its domain, that is $f_* = \inf_x f(x)$. The main theorem of this section is the following.

Theorem 22. *Suppose that there exists a point x such that $f(x) \le f_* + \varepsilon^2 \ell_{\mathbf{A}}/24$, and $\|x\|_\infty \le B$. Then, we can compute an ε-approximate balancing of $\mathbf{A}$ in time*

$$\widetilde{O}(mB \log^2(w_{\mathbf{A}}/\varepsilon)) .$$

Similarly to the matrix scaling case, the proof of this theorem follows directly from the key lemmas presented below. First, we prove that small additive error in function optimization implies an approximate balancing for $\mathbf{A}$.

Lemma 23. *Consider a matrix $\mathbf{A}$ and the corresponding function f. Any vector x satisfying $f(x) - f_* \le \varepsilon^2 \ell_{\mathbf{A}}/8$ yields an ε-approximate balancing of $\mathbf{A}$:*

$$\mathbf{M} = \mathbb{D}(\exp(x)) \cdot \mathbf{A} \cdot \mathbb{D}(\exp(-x)) .$$

Proving the lemma requires computing the first and second order derivatives of f.

Lemma 24. *Let $\mathbf{M}$ be the matrix obtained by balancing $\mathbf{A}$ with the vector x, which corresponds to $\mathbf{M} = \mathbb{D}(\exp(x)) \cdot \mathbf{A} \cdot \mathbb{D}(\exp(-x))$. The gradient and Hessian of f satisfy the identities:*

$$\nabla f(x) = r_{\mathbf{M}} - c_{\mathbf{M}} ,$$
$$\nabla^2 f(x) = \mathbb{D}(r_{\mathbf{M}} + c_{\mathbf{M}}) - (\mathbf{M} + \mathbf{M}^\top) .$$

Intuitively, since the gradient is 0 precisely when the corresponding point produces an exact balancing, a small gradient should imply a good approximate balancing. This guides the proof of Lemma 23. We will prove that a large gradient corresponds to being able to significantly decrease the function value, thus contradicting the approximate optimality of the point.

2) Regularization for Solving via Box-Constrained Newton Method: We observe that the function f defined in (4) satisfies all the conditions required to efficiently minimize it using the method we described in Section III.

Lemma 25. *The function f is convex, second-order robust with respect to ℓ_∞, and its Hessian is SDD.*

The method we described in Section III depends on a promise concerning the point we initialize it with. Recall that in order to apply Theorem 2 we require an upper bound on the size of the ℓ_∞-ball containing the level set of the initial point. In order to provide good bounds, we regularize f. The description and effect of this regularization in captured in the following lemma.

Lemma 26. *Suppose that there exists a point x such that $f(x) \le f_* + \varepsilon^2 \ell_{\mathbf{A}}/24$, and $\|x\|_\infty \le B$. Then, the regularization of f is defined as*

$$\widetilde{f}(x) = f(x) + \frac{\varepsilon^2 \ell_{\mathbf{A}}}{48 n e^B} \sum_{i=1}^{n} (e^{x_i} + e^{-x_i}) \tag{5}$$

and satisfies the following properties:

1) *$\widetilde{f}$ is second-order robust with respect to ℓ_∞ and has a SDD Hessian,*
2) *$f(x) \le \widetilde{f}(x)$, and if $\widetilde{x}^*$ is the minimizer of $\widetilde{f}$, then $\widetilde{f}(\widetilde{x}^*) \le f(x^*) + \varepsilon^2 \ell_{\mathbf{A}}/24$,*
3) *for all y such that $\widetilde{f}(y) \le \widetilde{f}(0)$, $\|y - x^*\|_\infty = O(B \log(n w_{\mathbf{A}}/\varepsilon))$.*

In particular, this lemma implies that approximately optimizing the regularized function will still produce an approximately balanced matrix.

Theorem 21 then follows by applying Theorem 2 to the regularized function defined in Lemma 26, and combining it with the error guarantee of Lemma 23. Similarly to the case of the scaling problem, we don't need to know any a priori bound on B. Just trying increasingly larger value of B (i.e., doubling our guess at each iteration) is sufficient.

3) Bounding the Condition Number of the Optimal Balancing: As we saw above, the running time given by Theorem 21 depends logarithmically on $\kappa(\mathbf{D}^*)$, where $\mathbf{D}^*$ is the matrix that achieves the optimal balancing. This parameter can be upper bound using certain input-dependent quantities:

Lemma 27. *Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be a nonnegative matrix. Suppose that the graph with adjacency matrix $\mathbf{A}$ is strongly connected, and every vertex can reach every other vertex within at most k hops. Then the matrix $\mathbf{D}^*$ that perfectly balances $\mathbf{A}$ has $\log\kappa(\mathbf{D}^*) = O(k \log w_{\mathbf{A}})$.*

The lemma yields the following upper bound on the value of $\kappa(\mathbf{D}^*)$.

Corollary 28. *If $\mathbf{A}$ is a balanceable matrix, and $\mathbf{D}^*$ perfectly balances it, then $\log\kappa(\mathbf{D}^*) = O(n \log w_{\mathbf{A}})$. If $\mathbf{A}$ is strictly positive, then $\log\kappa(\mathbf{D}^*) = O(\log w_{\mathbf{A}})$.*

V. Implementing an $O(\log n)$-Oracle in Nearly Linear Time

In Section IV we reduced the balancing and scaling problems to the approximate minimization of second-order robust functions with respect to the ℓ_∞ norm. All that is left to have a complete algorithm, we need a fast procedure to implement a k-oracle as in Definition 3. Namely, show how to construct an $O(\log n)$-oracle for the problem,

$$\min_{\|x\|_\infty \le 1} x^\top \mathbf{M} x + \langle b, x\rangle\,, \tag{6}$$

where $\mathbf{M}$ is an SDD matrix. For this section, whenever we say that a matrix is SDD we will also imply that the off-diagonal entries are nonpositive.

One possible approach, is to use standard convex optimization reductions to turn this problem into the minimization of the maximum of an ℓ_∞ norm and an ℓ_2 norm subject to linear constraints. This problem can be solved in time $\widetilde{O}(mn^{1/3})$ using the multiplicative weights framework as applied in [47], [48]. The resulting algorithm for implementing the k-oracle would take time $\widetilde{O}(m+n^{4/3})$, by taking advantage of spectral sparsification algorithms [49], [50], [51]. Instead, we will come up with a faster algorithm.

Our approach, based on the Lee-Peng-Spielman solver [45], is to identify large sets of vertices where the problem is "easy" to solve and then deal with the rest of the graph (reduced in size) recursively. The particular notion of "easy" we are going to use, is that of strong diagonal dominance.

Definition 29. A matrix $\mathbf{M}$ is *α-strongly diagonally dominant (α-SDD)*, if for all i

$$\mathbf{M}_{ii} \ge (1+\alpha)\sum_{j\neq i} |\mathbf{M}_{ij}|.$$

The reason that such matrices enable us to solve the corresponding problems fast is that they can be well-approximated by a diagonal matrix.

Lemma 30. *Every α-SDD matrix $\mathbf{M}$, with diagonal $\mathbb{D}(\mathbf{M})$, satisfies*

$$\left(1-\frac{1}{1+\alpha}\right)\mathbb{D}(\mathbf{M}) \preccurlyeq \mathbf{M} \preccurlyeq \left(1+\frac{1}{1+\alpha}\right)\mathbb{D}(\mathbf{M}).$$

In our context, problems in the form of Equation 6, where $\mathbf{M}$ is an α-SDD matrix for some $\alpha \ge \Omega(1)$, can be turned into well conditioned quadratic minimization problems for which we can apply standard linearly convergent algorithms. For a more detailed description and analysis of such algorithms can be found in [52].

Lemma 31. *There is an algorithm* FastSolve*, that given an $\Omega(1)$-SDD matrix $\mathbf{M}$, and $\varepsilon > 0$, returns a point $\widetilde{x}$, such that $\|\widetilde{x}\|_\infty \le 2$, and*

$$\widetilde{x}^\top \mathbf{M}\widetilde{x} + \langle \widetilde{x}, b\rangle \le (1-\varepsilon)\min_{\|x\|_\infty\le 2} x^\top \mathbf{M} x + \langle x, b\rangle$$

in time $O(m\log(1/\varepsilon))$, where m is the number of nonzero entries of $\mathbf{M}$.

An even simpler case is when the matrix is of size 1, in which case the problem can be *exactly* solved in constant time:

Lemma 32. *There is an algorithm* TrivialSolve*, that given a 1 by 1 matrix $\mathbf{M}$ returns an x optimizing $x^\top \mathbf{M} x + \langle b, x\rangle$ over the interval $[-1, 1]$.*

A key insight of [45] is that one can find $\Omega(1)$-SDD submatrices of $\mathbf{M}$ of size $\Omega(n)$. We denote such a subset by F and $V \setminus F$ by C. To ensure that solving the problem for x_F will not interfere with our solution x_C we map a solution $\hat{x}_C$ supported only on coordinates of C to a solution x_C through a linear mapping $\mathbf{P}$. If $\mathbf{P}$ were the energy minimizing extension of voltages on C to voltages on V,

$$(\mathbf{P}\hat{x}_C)_F = \mathbf{M}^{-1}_{[F,F]}\mathbf{M}_{[F,C]}\hat{x}_C,$$

we would have that x_F and x_C are $\mathbf{M}$-orthogonal, since $x_F^\top \mathbf{M}\mathbf{P}\hat{x}_C = 0$. Then, optimizing over $\hat{x}_C$ involves the quadratic $\mathbf{P}^\top\mathbf{M}\mathbf{P}$ which is exactly equal to $\mathbf{M}_{[C,C]} - \mathbf{M}_{[C,F]}\mathbf{M}^{-1}_{[F,F]}\mathbf{M}_{[F,C]} = \mathrm{Sc}(\mathbf{M}, F)$. Applying this proccess recursively leads to the notion of vertex sparsifier chains that we will heavily rely on.

Definition 33 (Definition 5.7 of [45])**.** For any SDD matrix $\mathbf{M}^{(0)}$, a vertex sparsifier chain of $\mathbf{M}^{(0)}$ with parameters $\alpha_i \ge 4$ and $1/2 \ge \varepsilon_i > 0$, is a sequence of matrices and subsets $(\mathbf{M}^{(1)}, \ldots, \mathbf{M}^{(d)}; F_1, \ldots, F_{d-1})$ such that

1) $\mathbf{M}^{(1)} \approx_{\varepsilon_0} \mathbf{M}^{(0)}$,
2) $\mathbf{M}^{(i+1)} \approx_{\varepsilon_i} \mathrm{Sc}(\mathbf{M}^{(i)}, F_i)$,
3) $\mathbf{M}^{(i)}_{[F_i,F_i]}$ is α_i-strongly diagonally dominant, and

4) $\mathbf{M}^{(d)}$ has size 1.

To be able to reason about the approximation guarantees of the chain as a whole we will use an error-quantifying definition.

Definition 34 (Definition 5.9 of [45])**.** An ε-vertex sparsifier chain of an SDD matrix $\mathbf{M}^{(0)}$ of work W, is a vertex sparsifier chain of $\mathbf{M}^{(0)}$ with parameters $\alpha_i \geq 4$ and $1/2 \geq \varepsilon_i > 0$ that satisfies

1) $2\sum_{i=0}^{d-1} \varepsilon_i \leq \varepsilon$,
2) $\sum_{i=0}^{d-1} m_i \log_{\alpha_i} \varepsilon_i^{-1} \leq W$, where m_i is the number of nonzeros in $L^{(i)}$.

Finally, the construction of such chains, as well as their error guarantees have been already analyzed in [45] and can be used in a black-box manner.

Theorem 35 (Theorem 5.10 of [45])**.** *Every SDD matrix $\mathbf{M}$ of dimension n has a δ-vertex sparsifier chain of work $O(n)$ and $d \leq O(\log n)$, for any constant $0 < \delta \leq 1$. Such a chain can be constructed in time, $\widetilde{O}(m)$.*

Since we cannot exactly compute the energy minizing mapping $\mathbf{P}$, we will define an approximate mapping that suffices for our purposes.

Definition 36. A linear mapping $\widetilde{\mathbf{P}}$ is an *ε-approximate voltage extension from C to V according to L* if for any $\hat{x}_C \in \mathbb{R}^{|C|}$,

1) $\|(\widetilde{\mathbf{P}} - \mathbf{P})\hat{x}_C\|_{\mathbf{M}} \leq \varepsilon \|\mathbf{P}\hat{x}_C\|_{\mathbf{M}}$,
2) $\widetilde{\mathbf{P}}$ is the identity on coordinates in C
3) the coordinates of $\widetilde{\mathbf{P}}\hat{x}_C$ are convex combination of the coordinates of $\hat{x}_C$ and 0.

where $\mathbf{P}$ is the energy-minimizing extension.

We will construct such a mapping through a simple averaging scheme. First we set the voltage of every vertex in F to be the weighted average of its neighbors in C. Then at every step we replace its voltage by the weighted average of *all* its neighbors. (Here, excess diagonal is treated as an edge to a vertex with voltage 0.) We do so for $O(\log(1/\varepsilon))$ iterations. We formally state the procedure and prove its correctness in the full version of the paper.

It is easy to see that all steps of the algorithm are linear maps, and we can therefore also implement its transpose.

Lemma 37. *For any SDD matrix $\mathbf{M}$, given an $\Omega(1)$-SDD subset F and some $\varepsilon > 0$ one can apply an ε-approximate voltage extension mapping in time $O(m \log(1/\varepsilon))$.*

Having expressed all of the components of our approach, stating the algorithm is simple. Given the decomposition of the problem the vertex sparsifier chain provides, we will solve the smallest problem and then iteratively combine it with the solution of the submatrices along the chain. The algorithm is formally described in Figure 1, and the main claim in Theorem 38.

OPTIMIZECHAIN$((\mathbf{M}^{(1)}, \ldots, \mathbf{M}^{(d)}; F_1, \ldots, F_{d-1}; \varepsilon_0, \ldots, \varepsilon_{d-1}), b)$

1) $b^{(1)} \leftarrow b/e^{\varepsilon_0}$
2) For $i \leftarrow 1, \ldots, d-1$
   a) $\widetilde{\mathbf{P}}^{(i)} \leftarrow$ APPROXMAPPING$(\mathbf{M}^{(i)}, F_i, \varepsilon_i)$
   b) $b^{(i+1)} \leftarrow (\widetilde{\mathbf{P}}^{(i)})^\top b^{(i)}/(e^{\varepsilon_i}(1 + \varepsilon_i + \varepsilon_i^2))$
3) $x^{(d)} \leftarrow$ TRIVIALSOLVE$(\mathbf{M}^{(d)}, b^{(d)})$
4) For $i \leftarrow d-1, \ldots, 1$
   a) $x_C^{(i)} \leftarrow \widetilde{\mathbf{P}}^{(i)} x^{(i+1)}$
   b) $x_F^{(i)} \leftarrow$ FASTSOLVE$(\mathbf{M}^{(i)}_{[F_i,F_i]}, b^{(i)}_{F_i}, \varepsilon_i)$
   c) $x^{(i)} \leftarrow x_C^{(i)} + x_F^{(i)}$
5) return $x^{(1)}$

Figure 1. Optimizing a vertex sparsifier chain

Theorem 38. *Algorithm* OPTIMIZECHAIN *implements a $O(\log n)$-oracle, and runs in time $\widetilde{O}(m)$.*

VI. Matrix Scaling and Balancing with Exponential Cone Programming

The algorithm developed in the previous sections is essentially optimal in the regime where the ratio between the scaling factors is relatively small (say polynomial in n). Since there are matrices for which this ratio is exponential, we develop a complementary algorithm with negligible runtime dependence on this ratio, at the cost of a mild increase in the dependence on m. The algorithm is based on interior point methods.

Although interior point methods would seem like a natural option for the problems of matrix scaling and balancing, standard formulations require solving linear systems involving various rescalings of the input matrix. A priori, it is not clear whether these can be solved faster than matrix multiplication time. However, it turns out that a somewhat nonstandard formulation requires solving linear systems for more structured matrices. Particularly, we see that these matrices admit a decomposition involving only matrices that are easy to invert (triangular matrices, solvable by back substitution, and SDD matrices which can be tackled via a standard Laplacian solver). Notably, a similar observation was made by Daitch and Spielman [53], in the case of interior point methods applied to flow problems on graphs. [34] also consider a formulation similar to ours for the matrix scaling problem, however they do not prove exact convergence bounds or state the algorithm rigorously. Moreover, since nearly-linear time SDD solvers where not known at the time, their algorithm provided no benefit compared to other approaches.

The main result of this section is the following.

Theorem 39. *Given a nonnegative matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, one can:*

1) *compute an ε-balancing in time $\widetilde{O}(m^{3/2}\log(w_{\mathbf{A}}\varepsilon^{-1}))$,*
2) *if the matrix is almost (r,c)-scalable, compute a ε-(r,c)-scaling in time $\widetilde{O}(m^{3/2}\log(s_{\mathbf{A}}\varepsilon^{-1}))$.*

This is as a matter of fact a consequence of the fact that a specific class of functions, which capture both balancing and scaling, can be minimized efficiently. We capture this result in the following Theorem.

Theorem 40. *Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be a nonnegative matrix with m nonzero entries, let f be the function*

$$f(x) = \sum_{(i,j)\in \mathrm{supp}(\mathbf{A})} \mathbf{A}_{ij} e^{x_i - x_j} - \langle d, x\rangle, \tag{7}$$

and let B_x be a positive real number. There exists an algorithm which, for any $\varepsilon > 0$, finds a vector x such that $f(x) - f(x^{*(B_x)}) \leq \varepsilon$ (where $x^{*(B_x)}$ is the optimum of f over the region $\|x\|_\infty \leq B_x$) in time

$$\widetilde{O}\left(m^{3/2}\log\left(2 + B_x + \frac{s_{\mathbf{A}}}{\varepsilon} + \frac{\|d\|_1}{\varepsilon}\right)\right).$$

Using this result, one can then conclude the proof of Theorem 39.

Acknowledgements

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. AM and DT were supported by the National Science Foundation under Grant No. 1553428. AV was supported by the National Science Foundation under Grant No. 1111109 and Grant No. 1553428.

References

[1] M. B. Cohen, A. Madry, D. Tsipras, and A. Vladu, "Matrix scaling and balancing via box constrained newton's method and interior point methods," *arXiv preprint arXiv:1704.02310*, 2017.

[2] E. E. Osborne, "On pre-conditioning of matrices," *Journal of the ACM*, vol. 7, no. 4, pp. 338–345, 1960.

[3] B. N. Parlett and C. Reinsch, "Balancing a matrix for calculation of eigenvalues and eigenvectors," *Numer. Math.*, vol. 13, no. 4, pp. 293–304, 1969.

[4] J. Grad, "Matrix balancing," *The Computer Journal*, vol. 14, no. 3, pp. 280–284, 1971.

[5] D. J. Hartfiel, "Concerning diagonal similarity of irreducible matrices," *Proceedings of the American Mathematical Society*, vol. 30, no. 3, pp. 419–425, 1971.

[6] B. C. Eaves, A. J. Hoffman, U. G. Rothblum, and H. Schneider, "Line-sum-symmetric scalings of square nonnegative matrices," in *Mathematical Programming Essays in Honor of George B. Dantzig Part II*. Springer, 1985, pp. 124–141.

[7] B. Kalantari, L. Khachiyan, and A. Shokoufandeh, "On the complexity of matrix balancing," *SIAM Journal on Matrix Analysis and Applications*, vol. 18, no. 2, pp. 450–463, 1997.

[8] H. Schneider and M. H. Schneider, "Max-balancing weighted directed graphs and matrix scaling," *Mathematics of Operations Research*, vol. 16, no. 1, pp. 208–222, 1991.

[9] R. Sinkhorn and P. Knopp, "Concerning nonnegative matrices and doubly stochastic matrices," *Pacific Journal of Mathematics*, vol. 21, no. 2, pp. 343–348, 1967.

[10] R. Sinkhorn, "A relationship between arbitrary positive matrices and doubly stochastic matrices," *The annals of mathematical statistics*, vol. 35, no. 2, pp. 876–879, 1964.

[11] J. H. Wilkinson, *Rounding errors in algebraic processes*. Courier Corporation, 1994.

[12] T. Raghavan, "On pairs of multidimensional matrices," *Linear Algebra and its Applications*, vol. 62, pp. 263–268, 1984.

[13] D. T. Brown, "A note on approximations to discrete probability distributions," *Information and Control*, vol. 2, no. 4, pp. 386–392, 1959.

[14] S. Friedland, C.-K. Li, and H. Schneider, "Additive decomposition of nonnegative matrices with applications to permanents and scalingt," *Linear and Multilinear Algebra*, vol. 23, no. 1, pp. 63–78, 1988.

[15] M. Idel, "A review of matrix scaling and sinkhorn's normal form for matrices and positive maps," *arXiv preprint arXiv:1609.06349*, 2016.

[16] MathWorks, "eig – eigenvalues and eigenvectors," https://www.mathworks.com/help/matlab/ref/eig.html.

[17] ——, "balance – diagonal scaling to improve eigenvalue accuracy," https://www.mathworks.com/help/matlab/ref/balance.html.

[18] RDocumentation, "Balance a square matrix via lapack's dgebal," https://www.rdocumentation.org/packages/expm/versions/0.99-1.1/topics/balance.

[19] V. Goulet, C. Dutang, M. Maechler, D. Firth, M. Shapira, and M. Stadelmann, "Package 'expm'," https://cran.r-project.org/web/packages/expm/expm.pdf.

[20] S. Blackford, "Balancing and conditioning," http://www.netlib.org/lapack/lug/node94.html.

[21] R. T. Kruithof, "De ingenieur 52," *E15-E25*, 1937.

[22] S. E. Fienberg, *The Analysis of Cross-Classified Categorical Data*. Springer, 1976.

[23] M. Cuturi, "Sinkhorn distances: Lightspeed computation of optimal transport," in *Advances in Neural Information Processing Systems*, 2013, pp. 2292–2300.

[24] N. Linial, A. Samorodnitsky, and A. Wigderson, "A deterministic strongly polynomial algorithm for matrix scaling and approximate permanents," in *STOC'98: Proceedings of the 30th Annual ACM Symposium on the Theory of Computing*, 1998, pp. 644–652.

[25] L. Gurvits, "Classical deterministic complexity of edmonds' problem and quantum entanglement," in *STOC'03: Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, 2003, pp. 10–19.

[26] G. U. Yule, "On the methods of measuring association between two attributes," *Journal of the Royal Statistical Society*, vol. 75, no. 6, pp. 579–652, 1912.

[27] J. Edmonds, "Maximum matching and a polyhedron with 0, 1-vertices," *Journal of Research of the National Bureau of Standards B*, vol. 69, no. 125-130, pp. 55–56, 1965.

[28] H. N. Gabow and R. E. Tarjan, "Faster scaling algorithms for network problems," *SIAM Journal on Computing*, vol. 18, no. 5, pp. 1013–1036, 1989.

[29] A. 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*. IEEE, 2013, pp. 253–262.

[30] M. H. Schneider and S. A. Zenios, "A comparative study of algorithms for matrix balancing," *Operations research*, vol. 38, no. 3, pp. 439–455, 1990.

[31] T.-Y. Chen and J. W. Demmel, "Balancing sparse matrices for computing eigenvalues," *Linear algebra and its applications*, vol. 309, no. 1-3, pp. 261–287, 2000.

[32] L. J. Schulman and A. Sinclair, "Analysis of a classical matrix preconditioning algorithm," in *STOC'15: Proceedings of the 47th Annual ACM on Symposium on Theory of Computing*, 2015, pp. 831–840.

[33] R. Ostrovsky, Y. Rabani, and A. Yousefi, "Matrix balancing in L_{p} norms: Bounding the convergence rate of osborne's iteration," in *SODA'17: Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2017, pp. 154–169.

[34] B. Kalantari and L. Khachiyan, "On the complexity of nonnegative-matrix scaling," *Linear Algebra and its applications*, vol. 240, pp. 87–103, 1996.

[35] A. Nemirovski and U. Rothblum, "On complexity of matrix scaling," *Linear Algebra and its Applications*, vol. 302, pp. 435–460, 1999.

[36] N. E. Young, R. E. Tarjan, and J. B. Orlin, "Faster parametric shortest path and minimum-balance algorithms," *Networks*, vol. 21, no. 2, pp. 205–221, 1991.

[37] A. Ben-Tal and A. Nemirovski, *Lectures on modern convex optimization: analysis, algorithms, and engineering applications*. SIAM, 2001.

[38] D. A. Spielman and S.-H. Teng, "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems," in *STOC'04: Proceedings of the 36th Annual ACM Symposium on the Theory of Computing*, 2004, pp. 81–90.

[39] I. Koutis, G. L. Miller, and R. Peng, "Approaching optimality for solving SDD systems," in *FOCS'10: Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science*, 2010, pp. 235–244.

[40] ——, "A nearly $m \log n$-time solver for SDD linear systems," in *FOCS'11: Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science*, 2011, pp. 590–598.

[41] J. A. Kelner, L. Orecchia, A. Sidford, and Z. A. Zhu, "A simple, combinatorial algorithm for solving SDD systems in nearly-linear time," in *STOC'13: Proceedings of the 45th Annual ACM Symposium on the Theory of Computing*, 2013, pp. 911–920.

[42] M. B. Cohen, R. Kyng, G. L. Miller, J. W. Pachocki, R. Peng, A. B. Rao, and S. C. Xu, "Solving sdd linear systems in nearly m $\log^{1/2} n$ time," in *STOC'14: Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, 2014, pp. 343–352.

[43] R. Kyng, Y. T. Lee, R. Peng, S. Sachdeva, and D. A. Spielman, "Sparsified cholesky and multigrid solvers for connection laplacians," in *STOC'16: Proceedings of the 48th Annual ACM Symposium on Theory of Computing*, 2016.

[44] R. Kyng and S. Sachdeva, "Approximate gaussian elimination for laplacians: Fast, sparse, and simple," in *FOCS'16: Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science*, 2016.

[45] Y. T. Lee, R. Peng, and D. A. Spielman, "Sparsified cholesky solvers for sdd linear systems," *arXiv preprint arXiv:1506.08204*, 2015.

[46] Z. Allen-Zhu, Y. Li, R. Oliveira, and A. Wigderson, "Much faster algorithms for matrix scaling," *arXiv preprint arXiv:1704.02315*, 2017.

[47] P. Christiano, J. A. Kelner, A. Madry, D. A. Spielman, and S. Teng, "Electrical flows, Laplacian systems, and faster approximation of maximum flow in undirected graphs," in *STOC'11: Proceedings of the 43rd ACM Symposium on Theory of Computing*, 2011, pp. 273–282.

[48] H. H. Chin, A. Madry, G. L. Miller, and R. Peng, "Runtime guarantees for regression problems," in *Innovations in Theoretical Computer Science, ITCS '13, Berkeley, CA, USA, January 9-12, 2013*, 2013, pp. 269–282.

[49] D. A. Spielman and N. Srivastava, "Graph sparsification by effective resistances," *SIAM Journal on Computing*, vol. 40, no. 6, pp. 1913–1926, 2011.

[50] Y. T. Lee and H. Sun, "Constructing linear-sized spectral sparsification in almost-linear time," in *STOC'17: Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science*, 2017.

[51] ——, "An SDP-based algorithm for linear-sized spectral sparsification," in *STOC'17: Proceedings of the 49th Annual ACM Symposium on the Theory of Computing*, 2015, pp. 250–269.

[52] Y. Nesterov, "Introductory lectures on convex programming volume i: Basic course," 1998.

[53] S. I. Daitch and D. A. Spielman, "Faster approximate lossy generalized flow via interior point algorithms," in *STOC '08: Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, 2008, pp. 451–460.