FULL LENGTH PAPER

Near-linear convergence of the Random Osborne algorithm for Matrix Balancing

Jason M. Altschuler[1] 🆔 · Pablo A. Parrilo[1]

© The Author(s) 2022

Abstract

We revisit Matrix Balancing, a pre-conditioning task used ubiquitously for computing eigenvalues and matrix exponentials. Since 1960, Osborne's algorithm has been the practitioners' algorithm of choice, and is now implemented in most numerical software packages. However, the theoretical properties of Osborne's algorithm are not well understood. Here, we show that a simple random variant of Osborne's algorithm converges in near-linear time in the input sparsity. Specifically, it balances $K \in \mathbb{R}_{\geq 0}^{n \times n}$ after $O(m\varepsilon^{-2} \log \kappa)$ arithmetic operations in expectation and with high probability, where m is the number of nonzeros in K, ε is the ℓ_1 accuracy, and $\kappa = \sum_{ij} K_{ij}/(\min_{ij:K_{ij} \neq 0} K_{ij})$ measures the conditioning of K. Previous work had established near-linear runtimes either only for ℓ_2 accuracy (a weaker criterion which is less relevant for applications), or through an entirely different algorithm based on (currently) impractical Laplacian solvers. We further show that if the graph with adjacency matrix K is moderately connected—e.g., if K has at least one positive row/column pair—then Osborne's algorithm initially converges exponentially fast, yielding an improved runtime $O(m\varepsilon^{-1} \log \kappa)$. We also address numerical precision issues by showing that these runtime bounds still hold when using $O(\log(n\kappa/\varepsilon))$-bit numbers. Our results are established through an intuitive potential argument that leverages a convex optimization perspective of Osborne's algorithm, and relates the per-iteration progress to the current imbalance as measured in Hellinger distance. Unlike previous analyses, we critically exploit log-convexity of the potential. Notably, our analysis extends to other variants of Osborne's algorithm: along the way, we also establish

✉ Jason M. Altschuler
   jasonalt@mit.edu

   Pablo A. Parrilo
   parrilo@mit.edu

[1] Laboratory for Information and Decision Systems, MIT, Cambridge, MA 02139, USA

significantly improved runtime bounds for cyclic, greedy, and parallelized variants of Osborne's algorithm.

1 Introduction

Let $\mathbf{1}$ denote the all-ones vector in $\mathbb{R}^n$. A nonnegative square matrix $A \in \mathbb{R}_{\geq 0}^{n \times n}$ is said to be *balanced* if its row sums $r(A) := A\mathbf{1}$ equal its column sums $c(A) := A^T\mathbf{1}$, i.e.

$$r(A) = c(A). \tag{1}$$

This paper revisits the classical problem of *Matrix Balancing*—sometimes also called *diagonal similarity scaling* or *line-sum-symmetric scaling*—which asks: given a nonnegative matrix $K \in \mathbb{R}_{\geq 0}^{n \times n}$, find a positive diagonal matrix D (if one exists[1]) such that $A := DKD^{-1}$ is balanced.

Matrix Balancing is a fundamental problem in numerical linear algebra, scientific computing, and theoretical computer science with many applications and an extensive literature dating back to 1960. The original papers [31, 34] considered the setup of balancing a matrix so that for every i, its i-th row and column have the same ℓ_p norm (rather than sum). Despite this problem's rich history, for nearly 60 years polynomial runtimes were unknown for Osborne's algorithm, the standard algorithm used in practice, until the breakthrough papers [40] for $p = \infty$ and then [32] for p finite. See Remark 3 for an expanded discussion of this history, the relations between these Matrix Balancing variants, and a straightforward reduction which extends all near-linear runtime results established in this paper to ℓ_p Matrix Balancing for finite p.

A particularly celebrated application of Matrix Balancing is pre-conditioning matrices before linear algebraic computations such as eigenvalue decomposition [31, 34] and matrix exponentiation [20, 47]. The point is that performing these linear algebra tasks on a balanced matrix can drastically improve numerical stability and readily recovers the desired answer on the original matrix [31]. Moreover, in practice, the runtime of (approximate) Matrix Balancing is essentially negligible compared to the runtime of these downstream tasks [35, Sect. 11.6.1]. The ubiquity of these applications has led to the implementation of Matrix Balancing in most linear algebra software packages, including EISPACK [42], LAPACK [5], R [36], and MATLAB [26]. In fact, Matrix Balancing is performed by default in the command for eigenvalue decomposition in MATLAB [27] and in the command for matrix exponentation for R [18]. Matrix Balancing also has other diverse applications in economics [39], information retrieval [46], and combinatorial optimization [4].

---

[1] K can be balanced if and only if K is irreducible [16]. This can be efficiently checked in linear time [45].

In practice, Matrix Balancing is performed approximately rather than exactly, since this can be done efficiently and typically suffices for applications. Specifically, in the *approximate Matrix Balancing problem*, the goal is to compute a scaling $A := DKD^{-1}$ that is ε-*balanced* in the ℓ_1 sense, i.e.,

$$\frac{\|r(A) - c(A)\|_1}{\sum_{ij} A_{ij}} \leq \varepsilon. \tag{2}$$

Remark 1 (ℓ_1 versus ℓ_2 error criterion) Several papers [22, 32] study approximate Matrix Balancing with ℓ_2 error criterion—rather than ℓ_1 as done here in (2) and in e.g., [29]—for what appears to be essentially historical reasons. Here, we focus solely on the ℓ_1 error criterion as it appears to be more useful for applications—e.g., it is critical for near-linear time approximation of the Min-Mean-Cycle problem [4]— in large part due to its natural interpretations in both probabilistic problems (as total variation imbalance) and graph theoretic problems (as netflow imbalance) [4, Remarks 2.1 and 5.8].[2] Note also that the approximate balancing criterion (2) is significantly easier to achieve[3] for ℓ_2 than ℓ_1: in fact, any matrix can be balanced to constant ℓ_2 error by only rescaling a vanishing $1/n$ fraction of the entries [32], whereas this is impossible for the ℓ_1 norm. (Note that this issue of which norm to measure error should *not* be confused with the ℓ_p Matrix Balancing problem, see Remark 3.)

1.1 Previous algorithms

The many applications of Matrix Balancing have motivated an extensive literature focused on solving it efficiently. However, there is still a large gap between theory and practice, and several key issues remain. We overview the relevant previous results below.

1.1.1 Practical state-of-the-art

Ever since its invention in 1960, *Osborne's algorithm* has been the algorithm of choice for practitioners [31, 34]. Osborne's algorithm is a simple iterative algorithm which initializes D to the identity (i.e., no balancing), and then in each iteration performs an *Osborne update* on some update coordinate $k \in [n]$, in which D_{kk} is updated to $\sqrt{c_k(A)/r_k(A)} D_{kk}$ so that the k-th row sum $r_k(A)$ and k-th column sum $c_k(A)$ of the

---

[2] The analogous observation has also been made for the intimately related problem of Matrix Scaling. For example, the ℓ_1 norm is pivotal there for applications including Optimal Transport [3] and Bipartite Perfect Matching [9].

[3] As a simple concrete example, let n be even and consider the $n \times n$ matrix A which is 0 everywhere except is the identity on the top right $n/2 \times n/2$ block. Note that $r(A)/\sum_{ij} A_{ij} = [\frac{2}{n}\mathbf{1}_{n/2}, \mathbf{0}_{n/2}]^T$ and $c(A)/\sum_{ij} A_{ij} = [\mathbf{0}_{n/2}, \frac{2}{n}\mathbf{1}_{n/2}]^T$. Thus A is *as unbalanced as possible* in ℓ_1 norm since $\|r(A) - c(A)\|_1/\sum_{ij} A_{ij} = 2$; however, A is very well balanced in ℓ_2 norm since $\|r(A) - c(A)\|_2/\sum_{ij} A_{ij} = 2/\sqrt{n}$ is vanishingly small.

current balancing $A = DKD^{-1}$ agree.[4] A more precise statement is in Algorithm 1 later.

The classical version of Osborne's algorithm, henceforth called *Round-Robin Cyclic Osborne*, chooses the update coordinates by repeatedly cycling through $\{1, \ldots, n\}$. This algorithm[5] performs remarkably well in practice and is the implementation of choice in most linear algebra software packages.

Despite this widespread adoption of Osborne's algorithm, a theoretical understanding of its convergence has proven to be quite challenging: indeed, non-asymptotic convergence bounds (i.e., runtime bounds) were not known for nearly 60 years until the breakthrough 2017 paper [32]. The paper [32] shows[6] that Round-Robin Cyclic Osborne computes an ε-balancing after $O(mn^2\varepsilon^{-2}\log\kappa)$ arithmetic operations, where m is the number of nonzeros in K, and $\kappa := (\sum_{ij} K_{ij})/(\min_{ij:K_{ij}\neq 0} K_{ij})$. They also show faster $\tilde{O}(n^2\varepsilon^{-2}\log\kappa)$ runtimes for two variants of Osborne's algorithm which choose update coordinates in different orders than cyclically. Here and henceforth, the $\tilde{O}$ notation suppresses polylogarithmic factors in n and ε^{-1}. The first variant, which we call *Greedy Osborne*, chooses the coordinate with maximal imbalance as measured by $\operatorname{argmax}_k(\sqrt{r_k(A)} - \sqrt{c_k(A)})^2$. They show that Greedy Osborne's runtime dependence on ε can be improved from ε^{-2} to ε^{-1}; however, this comes at the high cost of an extra factor of n. A disadvantage of Greedy Osborne is that it has numerical precision issues and requires operating on $O(n\log\kappa)$-bit numbers. The second variant, which we call *Weighted Random Osborne*, chooses coordinate k with probability proportional to $r_k(A) + c_k(A)$, and can be implemented using $O(\log(n\kappa/\varepsilon))$-bit numbers.

Collectively, these runtime bounds are fundamental results since they establish that Osborne's algorithm has polynomial runtime in n and ε^{-1}, and moreover that variants of it converge in roughly $\tilde{O}(n^2\varepsilon^{-2})$ time for matrices satisfying $\log\kappa = \tilde{O}(1)$—henceforth called *well-conditioned matrices*. However, these theoretical runtime bounds are still much slower than both Osborne's rapid empirical convergence and the state-of-the-art theoretical algorithms described below.

Two remaining open questions that this paper seeks to address are:

1. *Near-linear runtime.*[7] Does (any variant of) Osborne's algorithm have near-linear runtime in the input sparsity m? The fastest known runtimes scale as n^2, which is significantly slower for sparse problems.
2. *Scalability in accuracy.* The fastest runtimes for (any variant of) Osborne's algorithm scale poorly in the accuracy as ε^{-2}. (Except Greedy Osborne, for which it

---

[4] We assume throughout that the diagonal of K is zero. This ensures that the Osborne update makes the row and column sums agree. This assumption is without loss of generality because if D ε-balances K with zeroed-out diagonal, then it also ε-balances K.

[5] To be precise, following [34], some implementations have two minor modifications: a pre-processing step where K is permuted to a triangular block matrix with irreducible diagonal blocks; and a restriction of the entries of D to exact powers of the radix base. We presently ignore these minor modifications since the former is easily performed in linear-time [45], and the latter is solely to safeguard against numerical precision issues in practice.

[6] Note that in [32], bounds are written for the ℓ_2 error criterion; see Remark 1.

[7] Throughout, we say a runtime is near-linear if it is $O(m)$, up to polylogarithmic factors in n and polynomial factors in the inverse accuracy ε^{-1}.

is only known that ε^{-2} can be replaced by ε^{-1} at the high cost of an extra factor of n.) Can this be improved?

1.1.2 Theoretical state-of-the-art

A separate line of work leverages sophisticated optimization techniques to solve a convex optimization problem equivalent to Matrix Balancing. These algorithms have $\log \varepsilon^{-1}$ dependence on the accuracy, but are not practical (at least currently) due to costly overheads required by their significantly more complicated iterations. This direction originated in [22], which showed that the Ellipsoid algorithm produces an approximate balancing in $\tilde{O}(n^4 \log((\log \kappa)/\varepsilon))$ arithmetic operations on $O(\log(n\kappa/\varepsilon))$-bit numbers. Recently, [12][8] gave an Interior Point algorithm with runtime $\tilde{O}(m^{3/2} \log(\kappa/\varepsilon))$ and a Newton-type algorithm with runtime $\tilde{O}(md \log^2(\kappa/\varepsilon) \log \kappa)$, where d denotes the diameter of the directed graph G_K with vertices $[n]$ and edges $\{(i, j) : K_{ij} > 0\}$ [12, Theorem 4.18, Theorem 6.1, and Lemma 4.24]. Note that under the condition that K is a *well-connected matrix*—by which we mean that G_K has polylogarithmic diameter $d = \tilde{O}(1)$—then this latter algorithm has near-linear runtime in the input sparsity m. However, these algorithms heavily rely upon near-linear time Laplacian solvers, for which practical implementations are not known.

1.2 Contributions

Random Osborne converges in near-linear time Our main result (Theorem 9) addresses the two open questions above by showing that a simple random variant of the ubiquitously used Osborne's algorithm has runtime that is (i) near-linear in the input sparsity m, and also (ii) linear in the inverse accuracy ε^{-1} for well-connected inputs. Property (i) amends the aforementioned gap between theory and practice that the fastest known runtime of Osborne's algorithm scales as n^2 [32], while a different, impractical algorithm has theoretical runtime which is (conditionally) near-linear in m [12]. Property (ii) shows that improving the runtime dependence in ε from ε^{-2} to ε^{-1} does not require paying a costly factor of n (c.f., [32]).

Specifically, we propose a variant of Osborne's algorithm—henceforth called *Random Osborne*[9]— which chooses update coordinates uniformly at random, and show the following.

Theorem 1 (Informal version of Theorem 9) *Random Osborne solves the approximate Matrix Balancing problem on input $K \in \mathbb{R}_{\geq 0}^{n \times n}$ to accuracy $\varepsilon > 0$ after*

$$O\left(\frac{m}{\varepsilon} \left(\frac{1}{\varepsilon} \wedge d \right) \log \kappa \right), \tag{3}$$

---

[8] Similar runtimes were also developed by [1].

[9] Not to be confused with the different randomized variant of Osborne's algorithm in [32, Sect. 5], which draws coordinates with non-uniform probabilities. We call that algorithm Weighted Random Osborne to avoid confusion.

Table 1 Variants of Osborne's algorithm for balancing a matrix $K \in \mathbb{R}_{\geq 0}^{n \times n}$ with m nonzeros to $\varepsilon \ell_1$ accuracy. For simplicity, here K is assumed well-conditioned (i.e., $\log \kappa = \tilde{O}(1)$) and well-connected (i.e., $d = \tilde{O}(1)$); see the main text for detailed dependence on $\log \kappa$ and d. Note that in [32], bounds are written for the ℓ_2 error criterion; see Remark 1. See the main text for descriptions of each variant, and also Sect. 2.4 for more details on Random-Reshuffle Cyclic, Greedy, and Random Osborne. Our new bounds are in bold. Theorems 8 and 10 provide runtimes which, while not-linear, improve upon previous complexity bounds for greedy and cyclic variants of the Osborne algorithm, respectively. Our main result, Theorem 9, provides the first near-linear runtime for any variant of Osborne's algorithm

Variant	Best runtime bound (arithmetic operations)	Polylog bits
Cyclic (Round-Robin)	$\tilde{O}(mn^2/\varepsilon^2)$ [32]	No
Cyclic (Random-Reshuffle)	$\boldsymbol{\tilde{O}(mn/\varepsilon)}$ **(Theorem** 10**)**	**Yes (Theorem** 13**)**
Weighted Random	$\tilde{O}(n^2/\varepsilon^2)$ [32]	Yes [32]
Greedy	$\tilde{O}((n^2/\varepsilon^2) \wedge (n^3/\varepsilon))$ [32] $\longrightarrow \boldsymbol{\tilde{O}(n^2/\varepsilon)}$ **(Theorem** 8**)**	No
Random	$\boldsymbol{\tilde{O}(m/\varepsilon)}$ **(Theorem** 9**)**	**Yes (Theorem** 13**)**

arithmetic operations, both in expectation and with high probability.

We make several remarks about Theorem 1. First, we interpret the runtime (3). This is the minimum of $O(m\varepsilon^{-2} \log \kappa)$ and $O(md\varepsilon^{-1} \log \kappa)$. The former is near-linear in m. The latter is too if G_K has polylogarithmic diameter $d = \tilde{O}(1)$—important special cases include matrices K containing at least one strictly positive row/column pair (there, $d = 1$), and matrices with random sparsity patterns (there, $d = \tilde{O}(1)$ with high probability, see, e.g., [8, Theorem 10.10]). Note that the complexity of Matrix Balancing is intimately related to the connectivity of G_K: indeed, K can be balanced if and only if G_K is strongly connected (i.e., if and only if d is finite) [31]. Intuitively, the runtime dependence on d is a quantitative measure of "how balanceable" the input K is.

We note that the high probability bound in Theorem 1 has tails that decay exponentially fast. This is optimal with our analysis, see Remark 8.

Next, we comment on the $\log \kappa$ term in the runtime. This term appears in all other state-of-the-art runtimes [12, 32] and is mild: indeed, $\log \kappa \leq \log m + \log(\max_{ij} K_{ij} / \min_{ij:K_{ij}>0} K_{ij})$, where the former summand is $\tilde{O}(1)$—hence why the runtime is *near*-linear—and the latter is the input size for the entries of K. In particular, if K has quasi-polynomially bounded entries, then $\log \kappa = \tilde{O}(1)$.

Next, we compare to existing runtimes. Theorem 9 (a.k.a., the formal version of Theorem 1) gives a faster runtime than any existing practical algorithm, see Table 1. If comparing to the (impractical) algorithm of [12] on a purely theoretical plane, neither runtime dominates the other, and which is faster depends on the precise parameter regime: [12] is better for high accuracy solutions,[10] while Random Osborne has better dependence on the conditioning κ of K and the connectivity d of G_K.

---

[10] We remark that in practical applications of Matrix Balancing such as pre-conditioning, low accuracy solutions typically suffice. Indeed, this is a motivation of the commonly used variant of Osborne's algorithm which restricts entries of the scaling D to exact powers of the radix base [34].

Table 2 Parallelized variants of Osborne's algorithm for balancing a matrix $K \in \mathbb{R}_{\geq 0}^{n \times n}$ with m nonzeros to ε ℓ_1 accuracy, given a partitioning of the dataset into p blocks (see Sect. 2.5 for details). For simplicity, here K is assumed well-conditioned (i.e., $\log \kappa = \tilde{O}(1)$) and well-connected (i.e., $d = \tilde{O}(1)$); see the main text for detailed dependence on $\log \kappa$ and d. All results are ours. The runtime and work bounds are in Theorem 11, and the bit-complexity bounds are in Theorem 13

Variant	Best runtime bound (rounds)	Total work	Polylog bits
Cyclic Block (Random-Reshuffle)	$\tilde{O}(p^2/\varepsilon)$	$\tilde{O}(mp/\varepsilon)$	Yes
Greedy Block	$\tilde{O}(p/\varepsilon)$	$\tilde{O}(mp/\varepsilon)$	No
Random Block	$\tilde{O}(p/\varepsilon)$	$\tilde{O}(m/\varepsilon)$	Yes

Finally, we remark about bit-complexity. In Sect. 8, we show that with only minor modification, Random Osborne is implementable using numbers with only logarithmically few $O(\log(n\kappa/\varepsilon))$ bits; see Theorem 13 for formal statement.

Simple, streamlined analysis for different Osborne variants. We prove Theorem 1 using an intuitive potential argument (overviewed in Sect. 1.3 below). An attractive feature of this argument is that with only minor modification, it adapts to other Osborne variants. We elaborate below; see also Tables 1 and 2 for summaries of our improved rates.

Greedy Osborne. We show an improved runtime for Greedy Osborne where the ε^{-2} dependence is improved to ε^{-1} at the cost of d (rather than a full factor of n as in [32]). Specifically, in Theorem 8, we show convergence after $O(n^2\varepsilon^{-1}(\varepsilon^{-1} \wedge d) \log \kappa)$ arithmetic operations, which improves upon the previous best $O(n^2\varepsilon^{-1} \log n \cdot (\varepsilon^{-1} \log \kappa \wedge n \log(\kappa/\varepsilon)))$ from [32]. (The other improved $\log n$ factor comes from simplifying the data structure used for efficient greedy updates, see Remark 6.)

Random-Reshuffle Cyclic Osborne. We analyze Random-Reshuffle Cyclic Osborne, which is the variant of Osborne's algorithm that cycles through all n indices using a fresh random permutation in each cycle. We show that this algorithm converges after $O(mn\varepsilon^{-1}(\varepsilon^{-1} \wedge d) \log \kappa)$ arithmetic operations (Theorem 10). Previously, the only known runtime bound for any variant of Osborne with "cyclic" updates in the sense that each index is updated exactly once per epoch, was the $O(mn^2\varepsilon^{-2} \log \kappa)$ runtime bound for Round-Robin Cyclic Osborne [32]. Although the version of Cyclic Osborne we study is different than the one studied in [32], we note that our runtime bound is a factor of n faster, and additionally a factor of $1/\varepsilon$ faster if the matrix is well-connected. Moreover, we show that Random-Reshuffle Cyclic Osborne can be implemented on numbers with $O(\log(n\kappa/\varepsilon))$-bit numbers (Theorem 13), whereas the analysis of Round-Robin Cyclic Osborne in [32] requires $O(n \log \kappa)$-bit numbers.

Parallelized Osborne. We also show fast convergence for the analogous greedy, cyclic, and random variants of a parallelized version of Osborne's algorithm that is recalled in Sect. 2.5. These runtimes bounds are summarized in Table 2. Our main result here is that—modulo at most a single $\log n$ factor arising from the conditioning $\log \kappa$ of the input—Random Block Osborne converges after (i) only a linear number $O(\frac{p}{\varepsilon}(\frac{1}{\varepsilon} \wedge d) \log \kappa)$ of synchronization rounds in the size p of the dataset partition; and (ii) the same amount of total work as its non-parallelized counterpart Random Osborne, which

is in particular near-linear in m (see Theorem 1 and the ensuing discussion). Property (i) shows that, when giving an optimal coloring of G_K, Random Osborne converges in linear time in the chromatic number $\chi(G_K)$ of G_K (see Sect. 2.5 for further details). Property (ii) shows that the speedup of parallelization comes at no cost in the total work.

1.3 Overview of approach

We establish all of our runtime bounds with essentially the same potential argument. Below, we first sketch this argument for Greedy Osborne, since it is the simplest. Next, we describe the modifications for Random Osborne—the argument is identical modulo probabilistic tools which, albeit necessary for a rigorous analysis, are not the heart of the argument. We then outline the analysis for Random-Reshuffle Cyclic Osborne, which follows as a straightforward corollary. We then briefly remark upon the very minor modifications required for the parallelized Osborne variants.

For all variants, the potential we use is $D \mapsto \Phi(D) - \inf_{D^*} \Phi(D^*)$, where for a positive diagonal matrix D, we write $\Phi(D) = \log \sum_{ij} A_{ij}$ to denote the logarithm of the sum of the entries of the current balancing $A = DKD^{-1}$. Minimizing this potential function is well-known to be equivalent to Matrix Balancing; details in the Preliminaries section Sect. 2.3. Note also that Osborne's algorithm is equivalent to Exact Coordinate Descent on this function—which, importantly, is convex after a re-parameterization; see Sect. 2.4. In the interest of accessibility, the below overview describes our approach at an informal level that does not require further background. Later, Sect. 2 provides these preliminaries, and Sect. 3 gives the technical details of the potential argument.

1.3.1 Argument for Greedy Osborne

Here we sketch the $O(n^2 \varepsilon^{-1}(\varepsilon^{-1} \wedge d) \log \kappa)$ runtime we establish for Greedy Osborne in Sect. 4. Since each Greedy Osborne iteration takes $O(n)$ arithmetic operations (see Sect. 2.4), it suffices to bound the number of iterations by $O(n\varepsilon^{-1}(\varepsilon^{-1} \wedge d) \log \kappa)$.

The first step is relating the per-iteration progress of Osborne's algorithm to the imbalance of the current balancing—as measured in *Hellinger distance* $\mathsf{H}(\cdot, \cdot)$. Specifically, we show that an Osborne update decreases the potential function by at least

$$\text{(per-iteration decrease in potential)} \gtrsim \frac{\mathsf{H}^2(r(P), c(P))}{n}, \tag{4}$$

where $P = A/\sum_{ij} A_{ij}$ is the normalization of the current scaling $A = DKD^{-1}$. Note that since P is normalized, its marginals $r(P)$ and $c(P)$ are both probability distributions.

The second step is lower bounding this Hellinger imbalance $\mathsf{H}^2(r(P), c(P))$ by something large, so that we can argue that each iteration makes significant progress. Following is a simple such lower bound that yields an $O(n^2 \varepsilon^{-2} \log \kappa)$ runtime bound. Modulo small constant factors: a standard inequality in statistics lower bounds

Hellinger distance by ℓ_1 distance (a.k.a. total variation distance), and the ℓ_1 distance is by definition at least ε if the current iterate is not ε-balanced (see (2)). Therefore

$$(\text{per-iteration decrease in potential}) \gtrsim \frac{\varepsilon^2}{n} \tag{5}$$

for each iteration before convergence. Since the potential is initially not very large (at most $\log \kappa$, see Lemma 3) and by construction always nonnegative, the total number of iterations before convergence is therefore at most $n\varepsilon^{-2} \log \kappa$.

The key to the improved bound is an extra inequality that shows that *the per-iteration decrease is very large when the potential is large*. Specifically, this inequality—which has a simple proof using convexity of the potential—implies the following improvement of (5)

$$(\text{per-iteration decrease in potential}) \gtrsim \frac{1}{n} \left[\frac{(\text{current potential})}{R} \vee \varepsilon \right]^2 \tag{6}$$

where $R = d \log \kappa$. The per-iteration decrease is thus governed by the maximum of these two quantities. In words, the former ensures a *relative improvement* in the potential, and the latter ensures an *additive improvement*. Which is bigger depends on the current potential: the former dominates when the potential is $\Omega(\varepsilon R)$, and the latter for $O(\varepsilon R)$. It can be shown that both "phases" require $O(n\varepsilon^{-1}d \log \kappa)$ iterations, yielding the desired improved rate (details in Sect. 4).

1.3.2 Argument for Random Osborne

The argument for Random Osborne is nearly identical, except for two minor changes. The first change is the per-iteration potential decrease. All the same bounds hold (i.e., (4), (5), and (6)), except that they are now *in expectation* rather than deterministic. Nevertheless, this large expected progress is sufficient to obtain the same iteration-complexity bound. Specifically, an expected bound on the number of iterations is proved using Doob's Optional Stopping Theorem, and a h.p. bound using a martingale Chernoff bound (details in Sect. 5.2).

The second change is the per-iteration runtime: it is faster in expectation.

Observation 2 (Per-iteration runtime of Random Osborne) *An iteration of Random Osborne requires $O(m/n)$ arithmetic operations in expectation.*

Proof The number of arithmetic operations required by an Osborne update on coordinate k is proportional to the number of nonzero entries on the k-th row and column of K. Since Random Osborne draws k uniformly from $[n]$, this number of nonzeros is $2m/n$ in expectation. □

Note that this per-iteration runtime is n^2/m times faster than Greedy Osborne's. This is why our bound on the total runtime of Random Osborne is roughly $O(m)$, whereas for Greedy Osborne it is $O(n^2)$.

A technical nuance is that arguing a final runtime bound from a per-iteration runtime and an iteration-complexity bound is a bit more involved for Random Osborne. This is essentially because the number of iterations is not statistically independent from the per-iteration runtimes. For Greedy Osborne, the final runtime is bounded simply by the product of the per-iteration runtime and the number of iterations. We show a similar bound for Random Osborne in expectation via a slight variant of Wald's inequality, and w.h.p. via a Chernoff bound; details in Sect. 5.1.

1.3.3 Argument for Random-Reshuffle Cyclic Osborne

Analyzing Cyclic Osborne (either Round-Robin or Random-Reshuffle) is difficult because the improvement of an Osborne update is significantly affected by the previous Osborne updates in the cycle—and this effect is difficult to track. We observe that our improved analysis for Random Osborne implies, as a straightforward corollary, a fast runtime for Random-Reshuffle Cyclic Osborne. Specifically, since Osborne updates monotonically improve the potential, the per-cycle improvement of Random-Reshuffle Cyclic Osborne is at least the improvement of the first iteration of the cycle, which equals the improvement of a single iteration of Random Osborne. This implies that Random-Reshuffle Cyclic Osborne requires at most n times more iterations than Random Osborne. Details in Sect. 6. We remark that while arguing about a cycle only through its first iteration is clearly quite pessimistic, improvements seem difficult. A similar difficulty occurs for the analysis of Cyclic Coordinate Descent in more general convex optimization setups; see, e.g., [44, 48].

1.3.4 Argument for parallelized Osborne

The argument for the parallelized variants of Osborne are nearly identical to the arguments for their non-parallelized counterparts, described above. Specifically, the main difference for the random and greedy variants is just that in the bounds (4), (5), and (6), the $1/n$ factor is improved to 1 over the partitioning size p. The same argument then results in a final runtime that is sped up by this factor of n/p. The only difference for analyzing the Random-Reshuffle Cyclic variant is that here, the analogous coupling argument only gives a slowdown of p rather than n. Details in Sect. 7.

1.3.5 Key differences from previous approaches

The only other polynomial-time analysis of Osborne's algorithm also uses a potential argument [32]. However, our argument differs in several key ways—which enables much tighter bounds as well as a simpler argument that extends to many variants of Osborne's algorithm. Notably, their proof of Lemma 3.1 (which is where they show that each iteration of Greedy Osborne makes progress; c.f. our Lemma 7) is specifically tailored to Greedy Osborne[11] and seems unextendable to other variants

---

[11] Specifically, to prove their Lemma 3.1, [32] uses in (3.6) the inequality $\max_{i \in [n]} a_i/b_i \geq (\frac{1}{n} \sum_{j=1}^{n} a_j)/(\frac{1}{n} \sum_{j=1}^{n} b_j)$ for positive $a_1, \ldots, a_n, b_1, \ldots, b_n$. Extending their analysis of Greedy

such as Random Osborne. In particular, this precludes obtaining the near-linear runtime shown in this paper. Another key difference is that they do *not* use convexity of their potential (explicitly written on [32, page 157]), whereas we exploit not only convexity but also *log-convexity* (note our potential is the logarithm of theirs). Specifically, they use [32, Lemma 2.2] to improve ε^{-2} to ε^{-1} dependence at the cost of an extra factor of n, whereas here we show a significantly tighter bound (see the proof of Proposition 6) that saves this factor of n for well-connected graphs by exploiting log-convexity of their potential.

1.4 Other related work

We briefly remark about several related lines of work. Reference [11] gives heuristics for speeding up Osborne's algorithm on sparse matrices in practice, but does not provide runtime bounds. Reference [33] gives a more complicated version of Osborne's algorithm that obtains a stricter approximate balancing in a polynomial (albeit less practical) runtime of roughly $\tilde{O}(n^{19}\varepsilon^{-4}\log^4\kappa)$. Reference [25] gives an asynchronous distributed version of Osborne's algorithm with applications to epidemic suppression.

Remark 2 *(Fast Coordinate Descent)* Since Osborne's algorithm is Exact Coordinate Descent on a certain associated convex optimization problem (details in Sect. 2.4), it is natural to ask what runtimes the extensive literature on Coordinate Descent implies for Matrix Balancing. However, applying general-purpose bounds on Coordinate Descent out-of-the-box gives quite pessimistic runtime bounds for Matrix Balancing[12], essentially because they only rely on coordinate-smoothness of the function. In order to achieve the near-linear time bounds in this paper, we heavily exploit the further global structure of the specific convex optimization problem at hand.

Remark 3 *(ℓ_p Matrix Balancing and Max Balancing*)] Historically, Matrix Balancing was first studied in the setting of: given input $K \in \mathbb{C}^{n \times n}$ and $p \in [1, \infty]$, compute $A = DKD^{-1}$ such that for each $i \in [n]$, the i-th row and column of A have (approximately) equal ℓ_p norm. (Note that this choice of ℓ_p norm for balancing should not be confused with the error criterion discussion in Remark 1.) The Matrix Balancing problem studied in this paper is a special case of this: it is ℓ_1 balancing a nonnegative matrix. However,

---

Footnote 11 continued
Osborne to Random Osborne would require replacing $\max_{i \in [n]} a_i/b_i$ by $\frac{1}{n}\sum_{i=1}^n a_i/b_i$ in that inequality; however, this inequality is false because an average of ratios is in general incomparable to the ratio of averages. We bypass this obstacle by arguing in such a way that the quantity we need to bound is not a fraction, since such an analysis readily extends to Random Osborne by linearity of expectation (see (21) and Lemma 8).

[12] E.g., consider applying the state-of-the-art guarantees of [2, 30] for accelerated Coordinate Descent algorithms (which, note also, do *not* correspond exactly to Osborne's algorithm since they do not perform exact coordinate minimization). These bounds apply to Random Coordinate Descent with judiciously chosen non-uniform sampling probabilities, and yield an iteration bound of $(\sum_{i=i}^n \sqrt{L_i})\delta^{-1/2}\|x^*\|_2$ for minimizing Φ (defined in Sect. 2.3) to δ additive accuracy, where L_i is the smoothness of Φ on coordinate i. By [22, Corollary 2] and Cauchy-Schwarz, $\delta = O(\varepsilon^2/n)$ ensures that such a δ-approximate minimizer of Φ corresponds to an ε-approximate balancing. Bounding $L_i \leq 1$ and $\|x^*\|_2 \leq \sqrt{nd}\log\kappa$ by Corollary 7 therefore yields a bound of $O(n^2\varepsilon^{-1}d\log\kappa)$ iterations. Since iterations takes $O(m/n)$ time on average, this yields a final runtime bound of $O(mn\varepsilon^{-1}d\log\kappa)$, which is not near-linear.

it is actually no less general, in the sense that for any finite p, ℓ_p balancing $K \in \mathbb{C}^{n \times n}$ is trivially reducible to ℓ_1 balancing the nonnegative matrix with entries $|K_{ij}|^p$, see, e.g., [37]. Thus, following the literature, we focus only on the version of Matrix Balancing described above.

A particularly interesting limiting case of ℓ_p Matrix Balancing is the case of $p = \infty$, a.k.a. Max-Balancing. In this case, the aforementioned reduction from p finite to $p = 1$ no longer applies. There is an extensive literature on this problem dating back to 1960, including polynomial-time combinatorial algorithms [38, 49] as well as a natural analog of Osborne's algorithm [31]. Just like the case of finite p, for ℓ_∞ Matrix Balancing Osborne's algorithm has long been the choice in practice, yet its analysis has proven difficult. Indeed, breakthroughs took roughly half a century: asymptotic convergence was not even known until 1998 [10], and the first runtime bound was shown only a few years ago [40]. However, despite the syntactic similarity of ℓ_p Matrix Balancing for p finite and p infinite, the two problems are fundamentally very different: not only are the balancing goals different (which begets remarkably different properties, e.g., the ℓ_∞ Matrix Balancing solution is not unique [10]), but also the algorithms are quite different (even the analogous versions of Osborne's algorithm) and their analyses do not appear to carry over [32].

Remark 4 *(Matrix Scaling and Sinkhorn's algorithm)* The Matrix Scaling problem is: given $K \in \mathbb{R}_{\geq 0}^{n \times n}$ and vectors $\mu, \nu \in \mathbb{R}_{\geq 0}^n$ satisfying $\sum_i \mu_i = \sum_i \nu_i$, find positive diagonal matrices D_1, D_2 such that $A := D_1 K D_2$ satisfies $r(A) = \mu$ and $c(A) = \nu$. The many applications of Matrix Scaling have motivated an extensive literature on it; see, e.g., the survey [21]. In analog to Osborne's algorithm for Matrix Balancing, there is a simple iterative procedure (Sinkhorn's algorithm) for Matrix Scaling [41]. Sinkhorn's algorithm was recently shown to converge in near-linear time [3] (see also [9, 15, 19]). The analysis there also uses a potential argument. Interestingly, the per-iteration potential improvement for Matrix Scaling is the *Kullback-Leibler divergence* of the current imbalance, whereas for Matrix Balancing it is the *Hellinger divergence*. Further connections related to algorithmic techniques in this paper are deferred to Appendix B.

1.5 Roadmap

Section 2 recalls preliminary background. Sect. 3 establishes the key lemmas in the potential argument. Sections 4, 5, 6, and 7 use these tools to prove fast convergence for Greedy, Random, Random-Reshuffle Cyclic, and parallelized Osborne variants, respectively. For simplicity of exposition, these sections assume exact arithmetic; bit-complexity issues are addressed in Sect. 8. Section 9 concludes with several open questions.

2 Preliminaries

2.1 Notation

For the convenience of the reader, we collect here the notation used commonly throughout the paper. We reserve $K \in \mathbb{R}_{\geq 0}^{n \times n}$ for the matrix we seek to balance, $\varepsilon > 0$ for the

balancing accuracy, m for the number of nonzero entries in K, G_K for the graph associated to K, and d for the diameter of G_K. We assume throughout that the diagonal of K is zero; this is without loss of generality because if D solves the ε-balancing problem for the matrix K with zeroed-out diagonal, then D solves the ε-balancing problem for K. The support, maximum entry, minimum nonzero entry, and condition number of K are respectively denoted by $\mathrm{supp}(K) = \{(i, j) : K_{ij} > 0\}$, $K_{\max} = \max_{ij} K_{ij}$, $K_{\min} = \min_{(i,j)\in\mathrm{supp}(K)} K_{ij}$, and $\kappa = (\sum_{ij} K_{ij})/K_{\min}$. The $\tilde{O}$ notation suppresses polylogarithmic factors in n and ε. The all-ones and all-zeros vectors in $\mathbb{R}^n$ are respectively denoted by $\mathbf{1}$ and $\mathbf{0}$. Let $v \in \mathbb{R}^n$. The ℓ_1 norm, ℓ_∞ norm, and variation semi-norm of v are respectively $\|v\|_1 = \sum_{i=1}^n |v_i|$, $\|v\|_\infty = \max_{i\in[n]} |v_i|$, and $\|v\|_{\mathrm{var}} = \max_i v_i - \min_j v_j$. We denote the entrywise exponentiation of v by $e^v \in \mathbb{R}^n$, and the diagonalization of v by $\mathbb{D}(v) \in \mathbb{R}^{n\times n}$. The set of discrete probability distributions on n atoms is identified with the simplex $\Delta_n = \{p \in \mathbb{R}_{\geq 0}^n : \sum_{i=1}^n p_i = 1\}$. Let $\mu, v \in \Delta_n$. Their Hellinger distance is $\mathsf{H}(\mu, v) = \sqrt{\frac{1}{2}\sum_{\ell=1}^n (\sqrt{\mu_\ell} - \sqrt{v_\ell})^2}$, and their total variation distance is $\mathsf{TV}(\mu, v) = \|\mu - v\|_1/2$. We abbreviate "with high probability" by w.h.p., "high probability" by h.p., and "almost surely" by a.s. We denote the minimum of $a, b \in \mathbb{R}$ by $a \wedge b$, and the maximum by $a \vee b$. Logarithms take base e unless otherwise specified. All other specific notation is introduced in the main text.

2.2 Matrix Balancing

The formal definition of the (approximate) Matrix Balancing problem is in the "log domain" (i.e., output $x \in \mathbb{R}^n$ rather than $\mathbb{D}(e^x)$). This is in part to avoid bit-complexity issues (see Sect. 8).

Definition 3 (Matrix Balancing)] The *Matrix Balancing problem* $\mathrm{BAL}(K)$ for input $K \in \mathbb{R}_{\geq 0}^{n\times n}$ is to compute a vector $x \in \mathbb{R}^n$ such that $\mathbb{D}(e^x)K\mathbb{D}(e^{-x})$ is balanced.

Definition 4 (Approximate Matrix Balancing) The *approximate Matrix Balancing problem* $\mathrm{ABAL}(K, \varepsilon)$ for inputs $K \in \mathbb{R}_{\geq 0}^{n\times n}$ and $\varepsilon > 0$ is to compute a vector $x \in \mathbb{R}^n$ such that $\mathbb{D}(e^x)K\mathbb{D}(e^{-x})$ is ε-balanced (see (1)).

$K \in \mathbb{R}_{\geq 0}^{n\times n}$ is said to be *balanceable* if $\mathrm{BAL}(K)$ has a solution. It is known that non-balanceable matrices can be approximately balanced to arbitrary precision (i.e., ABAL has a solution for every $K \in \mathbb{R}_{\geq 0}^{n\times n}$ and $\varepsilon > 0$), and moreover that this is efficiently reducible to approximately balancing balanceable matrices, see, e.g., [11, 12]. Thus, following the literature, we assume throughout that K is balanceable. In the sequel, we make use of the following classical characterization of balanceable matrices in terms of their sparsity patterns.

Lemma 1 (Characterization of balanceability) *$K \in \mathbb{R}_{\geq 0}^{n\times n}$ is balanceable if and only if it is irreducible—i.e., if and only if G_K is strongly connected [31].*

2.3 Matrix Balancing as convex optimization

Key to to our analysis—as well as much of the other Matrix Balancing literature (e.g., [12, 22, 29, 32])—is the classical connection between (approximately) balancing a

matrix $K \in \mathbb{R}_{\geq 0}^{n \times n}$ and (approximately) solving the convex optimization problem

$$\min_{x \in \mathbb{R}^n} \Phi(x) := \log \sum_{ij} e^{x_i - x_j} K_{ij}. \tag{7}$$

In words, balancing K is equivalent to scaling $D K D^{-1}$ so that the sum of its entries is minimized. This equivalence follows from KKT conditions and convexity of $\Phi(x)$, which ensures that local optimality implies global optimality. Intuition comes from computing the gradient:

$$\nabla \Phi(x) = \frac{A\mathbf{1} - A^T \mathbf{1}}{\sum_{ij} A_{ij}}, \quad \text{where } A := \mathbb{D}(e^x) K \mathbb{D}(e^{-x}). \tag{8}$$

Indeed, solutions of BAL(K) are points where this gradient vanishes, and thus are in correspondence with minimizers of Φ. This also holds approximately: solutions of ABAL(K, ε) are in correspondence with ε-stationary points for Φ w.r.t. the ℓ_1 norm, i.e., $x \in \mathbb{R}^n$ for which $\|\nabla \Phi(x)\|_1 \leq \varepsilon$. The following lemma summarizes these classical connections; for a proof see, e.g., [22].

Lemma 2 (Matrix Balancing as convex optimization) *Let* $K \in \mathbb{R}_{\geq 0}^{n \times n}$ *and* $\varepsilon > 0$. *Then:*

1. Φ *is convex over* $\mathbb{R}^n$.
2. $x \in \mathbb{R}^n$ *is a solution to* BAL(K) *if and only if x minimizes* Φ.
3. $x \in \mathbb{R}^n$ *is a solution to* ABAL(K, ε) *if and only if* $\|\nabla \Phi(x)\|_1 \leq \varepsilon$.
4. *If K is balanceable, then* Φ *has a unique minimizer modulo translations of* $\mathbf{1}$.

2.4 Osborne's algorithm as coordinate descent

Lemma 2 equates the *problems* of (approximate) Matrix Balancing and (approximate) optimization of (7). This correspondence extends to *algorithms*. In particular, in the sequel, we repeatedly leverage the following known connection, which appears in, e.g., [32].

Observation 5 (Osborne's algorithm as Cordinate Descent) *Osborne's algorithm for Matrix Balancing is equivalent to Exact Coordinate Descent for optimizing (7).*

To explain this connection, let us recall the basics of both algorithms. Exact Coordinate Descent is an iterative algorithm for minimizing a function Φ that maintains an iterate $x \in \mathbb{R}^n$, and in each iteration updates x along a coordinate $k \in [n]$ by

$$x \leftarrow \underset{z \in \{x + \alpha e_k : \alpha \in \mathbb{R}\}}{\operatorname{argmin}} \Phi(z), \tag{9}$$

where e_k denotes the k-th standard basis vector in $\mathbb{R}^n$. In words, this update (9) improves the objective $\Phi(x)$ as much as possible by varying only the k-th coordinate of x.

Osborne's algorithm, as introduced briefly in Sect. 1, is an iterative algorithm for Matrix Balancing that repeatedly balances row/column pairs. Algorithm 1 provides pseudocode for an implementation on the "log domain" that maintains the logarithms $x \in \mathbb{R}^n$ of the scalings rather than the scalings $\mathbb{D}(e^x)$ themselves. The connection in Observation 5 is thus, stated more precisely, that *Osborne's algorithm is a specification of the Exact Coordinate Descent algorithm to minimizing the function Φ in* (7) *with initialization of* **0**. This is because the Exact Coordinate Descent update to Φ on coordinate $k \in [n]$ updates x_k so that $\frac{\partial \Phi}{\partial x_k}(x) = 0$, which by the derivative computation in (8) amounts to updating x_k so that the k-th row and column sums of the current balancing are equal—which is precisely the update rule for Osborne's algorithm on coordinate k.

Algorithm 1 Osborne's algorithm for Matrix Balancing. The variant (e.g., Greedy, Random, or Random-Reshuffle Cyclic) depends on how the update coordinate is chosen in Line 3.

    Input: Matrix $K \in \mathbb{R}_{\geq 0}^{n \times n}$ and accuracy $\varepsilon > 0$
    Output: Vector $x \in \mathbb{R}^n$ that solves ABAL(K, ε)
1: $x \leftarrow \mathbf{0}$ ▷ Initialization
2: **while** $\mathbb{D}(e^x)K\mathbb{D}(e^{-x})$ is not ε-balanced **do**
3:     Choose update coordinate $k \in [n]$
4:     $x_k \leftarrow x_k + \frac{\log(c_k(\mathbb{D}(e^x)K\mathbb{D}(e^{-x}))) - \log(r_k(\mathbb{D}(e^x)K\mathbb{D}(e^{-x})))}{2}$ ▷ Osborne update on coordinate k
5: **return** x

We note that besides elucidating Observation 5, the log-domain implementation of Osborne's Algorithm in Algorithm 1 is also critical for numerical precision, both in theory and practice.

Remark 5 *(Log-domain implementation)* In practice, Osborne's algorithm should be implemented in the "logarithmic domain", i.e., store the iterates x rather than the scalings $\mathbb{D}(e^x)$, operate on K through $\log K_{ij}$ (see Remark 9), and compute Osborne updates using the following standard trick for numerically computing log-sum-exp: $\log(\sum_{i=1}^n e^{z_i}) = \max_j z_j + \log(\sum_{i=1}^n e^{z_i - \max_j z_j})$. In Sect. 8, we show that essentially just these modifications enable a provably logarithmic bit-complexity for several variants of Osborne's algorithm (Theorem 13).

It remains to discuss the choice of update coordinate in Osborne's algorithm (Line 3 of Algorithm 1), or equivalently, in Coordinate Descent. We focus on the following natural options:

– *Random-Reshuffle Cyclic Osborne* Cycle through the coordinates, using an independent random permutation for the order each cycle.
– *Greedy Osborne* Choose the coordinate k for which the k-th row and column sums of the current scaling $A := \mathbb{D}(e^x)K\mathbb{D}(e^{-x})$ disagree most, as measured by

$$\underset{k \in [n]}{\operatorname{argmax}} \left| \sqrt{r_k(A)} - \sqrt{c_k(A)} \right|. \tag{10}$$

(Ties are broken arbitrarily, e.g., lowest number.)

– *Random Osborne* Sample k uniformly from $[n]$, independently between iterations.

Remark 6 *(Efficient implementation of Greedy)* In order to efficiently compute (10), Greedy Osborne maintains an auxiliary data structure: the row and column sums of the current balancing. This requires only $O(n)$ additional space, $O(m)$ additional computation in a pre-processing step, and $O(n)$ additional per-iteration computation for maintenance (increasing the per-iteration runtime by a small constant factor).

2.5 Parallelizing Osborne's algorithm via graph coloring

For scalability, parallelization of Osborne's algorithm can be critical. It is well-known (see, e.g., [7]) that Osborne's algorithm can be parallelized when one can compute a (small) coloring of G_K, i.e., a partitioning $S_1, \ldots, S_p$ of the vertices $[n]$ such that any two vertices in the same partitioning are non-adjacent. This idea stems from the observation that simultaneous Osborne updates do not interfere with each other when performed on coordinates corresponding to *non-adjacent* vertices in G_K. Indeed, this suggests a simple, natural parallelization of Osborne's algorithm given a coloring: update in parallel all coordinates of the same color. We call this algorithm *Block Osborne* due to the following connection to Exact Block Coordinate Descent, i.e., the variant of Exact Coordinate Descent where an iteration exactly minimizes over a subset (a.k.a., block) of the variables.

Remark 7 *(Block Osborne as Block Coordinate Descent)* Extending Observation 5, *Block* Osborne is equivalent to Exact *Block* Coordinate Descent for minimizing Φ. The connection to coloring is equivalently explained through this convex optimization lens: for each S_ℓ, the (exponential[13] of) Φ is separable in the variables in S_ℓ. This is why their updates are independent.

Just like the standard (non-parallelized) Osborne algorithm, the Block Osborne algorithm has several natural options for the choice of update block:

– *Random-Reshuffle Cyclic Block Osborne* Cycle through the blocks, using an independent random permutation for the order each cycle.
– *Greedy Block Osborne* Choose the block ℓ maximizing

$$\frac{1}{|S_\ell|} \sum_{k \in S_\ell} \left(\sqrt{r_k(A)} - \sqrt{c_k(A)} \right)^2 \tag{11}$$

where A denotes the current balancing. (Ties are broken arbitrarily, e.g., lowest number.)
– *Random Block Osborne* Sample ℓ uniformly from $[p]$, independently between iterations.

Note that if $S_1, \ldots, S_p$ are singletons—e.g., when $K \in \mathbb{R}_{>0}^{n \times n}$ is strictly positive—then these variants of Block Osborne degenerate into the corresponding variants of the standard Osborne algorithm.

---

[13] Note that by monotonocity of $\exp(\cdot)$, minimizing $\exp(\Phi(\cdot))$ is equivalent to minimizing $\Phi(\cdot)$.

Of course, Block Osborne first requires a coloring of G_K. A smaller coloring yields better parallelization (indeed we establish a linear runtime in the number of colors, see Sect. 7). However, finding the (approximately) smallest coloring is NP-hard [17, 23, 50]. Nevertheless, in certain cases a relatively good coloring may be obvious or easily computable. For instance, in certain applications the sparsity pattern of K could be structured, known a priori, and thus leveraged. An easily computable setting is matrices with uniformly sparse rows and columns, i.e., matrices whose corresponding graph G_K has bounded max-degree; see Corollary 12.

3 Potential argument

Here we develop the ingredients for our potential-based analysis of Osborne's algorithm. They are purposely stated *independently of the Osborne variant*, i.e., how the Osborne algorithm chooses update coordinates. This enables the argument to be applied directly to different variants in the sequel. We point the reader to Sect. 1.3 for a high-level overview of the argument.

First, we recall the following standard bound on the initial potential. This appears in, e.g., [12, 32]. For completeness, we briefly recall the simple proof. Below, we denote the optimal value of the convex optimization problem (7) by $\Phi^* := \min_{x \in \mathbb{R}^n} \Phi(x)$.

Lemma 3 (Bound on initial potential) $\Phi(\mathbf{0}) - \Phi^* \leq \log \kappa$.

Proof It suffices to show $\Phi^* \geq \log K_{\min}$. Since K is balanceable, G_K is strongly connected (Lemma 1), thus G_K contains a cycle. By an averaging argument, this cycle contains an edge (i, j) such that $x_i^* - x_j^* \geq 0$. Thus $\Phi^* \geq \log(e^{x_i^* - x_j^*} K_{ij}) \geq \log K_{\min}$.
□

Next, we exactly compute the decrease in potential from an Osborne update on a fixed coordinate $k \in [n]$. This is a simple, direct calculation and is similar to [32, Lemma 2.1].

Lemma 4 (Potential decrease from Osborne update) *Consider any $x \in \mathbb{R}^n$ and update coordinate $k \in [n]$. Let x' denote the output of an Osborne update on x w.r.t. coordinate k, $A := \mathbb{D}(e^x) K \mathbb{D}(e^{-x})$ denote the scaling corresponding to x, and $P := A/(\sum_{ij} A_{ij})$ its normalization. Then*

$$\Phi(x) - \Phi(x') = -\log\left(1 - \left(\sqrt{r_k(P)} - \sqrt{c_k(P)}\right)^2\right). \tag{12}$$

Proof Let $A' := \mathbb{D}(e^{x'}) K \mathbb{D}(e^{-x'})$ denote the scaling corresponding to the next iterate x'. Then $e^{\Phi(x)} - e^{\Phi(x')} = (r_k(A) + c_k(A)) - (r_k(A') + c_k(A')) = (r_k(A) + c_k(A)) - 2\sqrt{r_k(A)}\sqrt{c_k(A)} = (\sqrt{r_k(A)} - \sqrt{c_k(A)})^2 = (\sqrt{r_k(P)} - \sqrt{c_k(P)})^2 e^{\Phi(x)}$. Dividing by $e^{\Phi(x)}$ and re-arranging proves (12).
□

In the sequel, we lower bound the per-iteration progress in (12) by $(\sqrt{r_k(P)} - \sqrt{c_k(P)})^2$ using the elementary inequality $-\log(1 - z) \geq z$. Analyzing this further

requires knowledge of how k is chosen, i.e., the Osborne variant. However, for both Greedy Osborne and Random Osborne, this progress is at least the average

$$\frac{1}{n} \sum_{k=1}^{n} (\sqrt{r_k(P)} - \sqrt{c_k(P)})^2 = \frac{2}{n} \mathsf{H}^2(r(P), c(P)). \tag{13}$$

(For Random Osborne, this statement requires an expectation; see Sect. 5.) The rest of this section establishes the main ingredient in the potential argument: Proposition 6 lower bounds this Hellinger imbalance, and thereby lower bounds the per-iteration progress. Note that Proposition 6 is stated for "nontrivial balancings", i.e., $x \in \mathbb{R}^n$ satisfying $\Phi(x) \leq \Phi(0)$. This automatically holds for any iterate of the Osborne algorithm—regardless of the variant—since the first iterate is initialized to $\mathbf{0}$, and since the potential is monotonically non-increasing by Lemma 4.

Proposition 6 (Lower bound on Hellinger imbalance) *Consider any* $x \in \mathbb{R}^n$. *Let* $A := \mathbb{D}(e^x) K \mathbb{D}(e^{-x})$ *denote the corresponding scaling, and let* $P := A / \sum_{ij} A_{ij}$ *denote its normalization. If* $\Phi(x) \leq \Phi(0)$ *and* A *is not* ε-balanced, then

$$\mathsf{H}^2(r(P), c(P)) \geq \frac{1}{8} \left(\frac{\Phi(x) - \Phi^*}{d \log \kappa} \vee \varepsilon \right)^2. \tag{14}$$

To prove Proposition 6, we collect several helpful lemmas. The first is a standard inequality in statistics which lower bounds the Hellinger distance between two probability distributions by their ℓ_1 distance (or equivalently, up to a factor of 2, their total variation distance) [13]. A short, simple proof via Cauchy-Schwarz is provided for completeness.

Lemma 5 (Hellinger versus ℓ_1 inequality) *If* $\mu, \nu \in \Delta_n$, *then*

$$H(\mu, \nu) \geq \frac{1}{2\sqrt{2}} \|\mu - \nu\|_1. \tag{15}$$

Proof By Cauchy-Schwarz, $\|\mu - \nu\|_1^2 = (\sum_k |\mu_k - \nu_k|)^2 = (\sum_k |\sqrt{\mu_k} - \sqrt{\nu_k}| \cdot |\sqrt{\mu_k} + \sqrt{\nu_k}|)^2 \leq (\sum_k (\sqrt{\mu_k} - \sqrt{\nu_k})^2) \cdot (\sum_k (\sqrt{\mu_k} + \sqrt{\nu_k})^2) = 2\mathsf{H}^2(\mu, \nu) \cdot (\sum_k (\mu_k + \nu_k + 2\sqrt{\mu_k \nu_k}))$. By the AM-GM inequality and the assumption $\mu, \nu \in \Delta_n$, the latter sum is at most $\sum_k (\mu_k + \nu_k + 2\sqrt{\mu_k \nu_k}) \leq 2 \sum_k (\mu_k + \nu_k) = 4$. $\square$

Next, we recall the following standard bound on the variation norm of nontrivial balancings. This bound is often stated only for optimal balancings (e.g., [12, Lemma 4.24])—however, the proof extends essentially without modifications; details are provided briefly for completeness.

Lemma 6 (Variation norm of nontrivial balancings) *If* $x \in \mathbb{R}^n$ *satisfies* $\Phi(x) \leq \Phi(0)$, *then* $\|x\|_{\mathrm{var}} \leq d \log \kappa$.

Proof Consider any $u, v \in [n]$. By definition of d, there exists a path in G_K from u to v of length at most d. For each edge (i, j) on the path, we have $e^{x_i - x_j} K_{ij} \leq \Phi(x) \leq$

$\Phi(0)$, and thus $x_i - x_j \leq \log \kappa$. Summing this inequality along the edges of the path and telescoping yields $x_u - x_v \leq d \log \kappa$. Since this holds for any u, v, we conclude $\|x\|_{\mathrm{var}} = \max_u x_u - \min_v x_v \leq d \log \kappa$. □

From Lemma 6, we deduce the following bound.

Corollary 7 (ℓ_∞ distance of nontrivial balancings to minimizers) *If $x \in \mathbb{R}^n$ satisfies $\Phi(x) \leq \Phi(0)$, then there exists a minimizer x^* of Φ such that $\|x - x^*\|_\infty \leq d \log \kappa$.*

Proof By definition, Φ is invariant under translations of $\mathbf{1}$. Choose any minimizer x^* and translate it by a multiple of $\mathbf{1}$ so that $\max_i (x - x^*)_i = -\min_j (x - x^*)_j$. Then $\|x - x^*\|_\infty = (\max_i (x_i - x_i^*) - \min_j (x_j - x_j^*))/2 \leq ((\max_i x_i - \min_j x_j) + (\max_i x_i^* - \min_j x_j^*))/2 = (\|x\|_{\mathrm{var}} + \|x^*\|_{\mathrm{var}})/2$. By Lemma 6, this is at most $d \log \kappa$. □

We are now ready to prove Proposition 6.

Proof (Proposition 6) Since P is normalized, its marginals $r(P)$ and $c(P)$ are both probability distributions in Δ_n. Thus by Lemma 5,

$$\mathsf{H}^2\big(r(P), c(P)\big) \geq \frac{1}{8} \|r(P) - c(P)\|_1^2. \qquad (16)$$

The claim now follows by lower bounding $\|r(P) - c(P)\|_1$ in two different ways. The first is $\|r(P) - c(P)\|_1 \geq \varepsilon$, which holds since A is not ε-balanced by assumption. The second is

$$\|r(P) - c(P)\|_1 \geq \frac{\Phi(x) - \Phi(x^*)}{d \log \kappa}, \qquad (17)$$

which we show presently. By convexity of Φ (Lemma 2) and then Hölder's inequality,

$$\Phi(x) - \Phi(x^*) \leq \langle \nabla \Phi(x), x - x^* \rangle \leq \|\nabla \Phi(x)\|_1 \|x - x^*\|_\infty \qquad (18)$$

for any minimizer x^* of Φ. Now by Corollary 7, there exists a minimizer x^* such that $\|x - x^*\|_\infty \leq d \log \kappa$; and by (8), the gradient is $\nabla \Phi(x) = r(P) - c(P)$. Rearranging (18) therefore establishes (17). □

4 Greedy Osborne converges quickly

Here we show an improved runtime bound for Greedy Osborne that, for well-connected sparsity patterns, scales (near) linearly in both the total number of entries n^2 and the inverse accuracy ε^{-1}. See Sect. 1.2 for further discussion of the result, and Sect. 1.3.1 for a proof sketch.

Theorem 8 (Convergence of Greedy Osborne) *Given a balanceable matrix $K \in \mathbb{R}_{\geq 0}^{n \times n}$ and accuracy $\varepsilon > 0$, Greedy Osborne solves $\mathrm{ABAL}(K, \varepsilon)$ in $O(\frac{n^2}{\varepsilon}(\frac{1}{\varepsilon} \wedge d) \log \kappa)$ arithmetic operations.*

The key lemma is that each iteration of Greedy Osborne improves the potential significantly.

Lemma 7 (Potential decrease of Greedy Osborne) *Consider any $x \in \mathbb{R}^n$ for which the corresponding scaling $A := \mathbb{D}(e^x) K \mathbb{D}(e^{-x})$ is not ε-balanced. If x' is the next iterate obtained from a Greedy Osborne update, then*

$$\Phi(x) - \Phi(x') \geq \frac{1}{4n} \left(\frac{\Phi(x) - \Phi^*}{d \log \kappa} \vee \varepsilon \right)^2.$$

Proof Using in order Lemma 4, the inequality $-\log(1 - z) \geq z$ which holds for any $z \in \mathbb{R}$, the definition of Greedy Osborne, and then Proposition 6,

$$\Phi(x) - \Phi(x') = -\log(1 - (\sqrt{r_k(P)} - \sqrt{c_k(P)})^2) \tag{19}$$

$$\geq \left(\sqrt{r_k(P)} - \sqrt{c_k(P)} \right)^2 \tag{20}$$

$$\geq \frac{1}{n} \sum_{\ell=1}^{n} \left(\sqrt{r_\ell(P)} - \sqrt{c_\ell(P)} \right)^2 \tag{21}$$

$$\geq \frac{1}{4n} \left(\frac{\Phi(x) - \Phi^*}{d \log \kappa} \vee \varepsilon \right)^2. \tag{22}$$

$\square$

Proof (Theorem 8) Let $x^{(0)} = \mathbf{0}, x^{(1)}, x^{(2)}, \ldots$ denote the iterates, and let τ be the first iteration for which $\mathbb{D}(e^x) K \mathbb{D}(e^{-x})$ is ε-balanced. Since the number of arithmetic operations per iteration is amortized to $O(n)$ by Remark 6, it suffices to show that the number of iterations τ is at most $O(n\varepsilon^{-1}(\varepsilon^{-1} \wedge d) \log \kappa)$. Now by Lemma 7, for each $t \in \{0, 1, \ldots, \tau - 1\}$ we have

$$\Phi(x^{(t)}) - \Phi(x^{(t+1)}) \geq \frac{1}{4n} \left(\frac{\Phi(x^{(t)}) - \Phi^*}{d \log \kappa} \vee \varepsilon \right)^2. \tag{23}$$

Case 1 $\varepsilon^{-1} \leq d$. By the second bound in (23), the potential decreases by at least $\varepsilon^2/4n$ in each iteration. Since the potential is initially at most $\log \kappa$ by Lemma 3 and is always nonnegative by definition, the total number of iterations is at most

$$\tau \leq \frac{\log \kappa}{\varepsilon^2/4n} = \frac{4n \log \kappa}{\varepsilon^2}. \tag{24}$$

Case 2 $\varepsilon^{-1} > d$. For shorthand, denote $\alpha := \varepsilon d \log \kappa$. Let τ_1 be the first iteration for which the potential $\Phi(x^{(t)}) - \Phi^* \leq \alpha$, and let $\tau_2 := \tau - \tau_1$ denote the number of remaining iterations. By an identical argument as in case 1,

$$\tau_2 \leq \frac{\alpha}{\varepsilon^2/4n} = \frac{4nd \log \kappa}{\varepsilon}. \tag{25}$$

To bound τ_1, partition this phase further as follows. Let $\phi_0 := \log \kappa$ and $\phi_i := \phi_{i-1}/2$ for $i = 1, 2, \dots$ until $\phi_N \leq \alpha$. Let $\tau_{1,i}$ be the number of iterations starting from when the potential is first no greater than ϕ_{i-1} and ending when it no greater than ϕ_i. In the i-th subphase, the potential drops by at least $(\frac{\phi_i}{d \log \kappa})^2/4n$ per iteration by (23). Thus

$$\tau_{1,i} \leq \frac{\phi_{i-1} - \phi_i}{(\frac{\phi_i}{d \log \kappa})^2/4n} = \frac{4nd^2 \log^2 \kappa}{\phi_i}. \tag{26}$$

Since $\sum_{i=1}^N \frac{1}{\phi_i} = \frac{1}{\phi_N} \sum_{j=0}^{N-1} 2^{-j} \leq \frac{2}{\phi_N} \leq \frac{4}{\alpha}$, thus

$$\tau_1 = \sum_{i=1}^N \tau_{1,i} \leq \frac{16nd^2 \log \kappa^2}{\alpha} = \frac{16nd \log \kappa}{\varepsilon}. \tag{27}$$

By (25) and (27), the total number of iterations is at most $\tau = \tau_1 + \tau_2 \leq 20nd\varepsilon^{-1} \log \kappa$. □

5 Random Osborne converges quickly

Here we show that Random Osborne has runtime that is (i) near-linear in the input sparsity m; and (ii) also linear in the inverse accuracy ε^{-1} for well-connected sparsity patterns. See Sect. 1.2 for further discussion of the result, and Sect. 1.3.2 for a proof sketch.

Theorem 9 (Convergence of Random Osborne) *Given a balanceable matrix $K \in \mathbb{R}_{\geq 0}^{n \times n}$ and accuracy $\varepsilon > 0$, Random Osborne solves* ABAL(K, ε) *in T arithmetic operations, where*

- *(Expectation guarantee.)* $\mathbb{E}[T] = O(\frac{m}{\varepsilon}(\frac{1}{\varepsilon} \wedge d) \log \kappa)$.
- *(H.p. guarantee.) There exists a universal constant $c > 0$ such that for all $\delta > 0$,*

$$\mathbb{P}\left(T \leq c\left(\frac{m}{\varepsilon}(\frac{1}{\varepsilon} \wedge d) \log \kappa \log \frac{1}{\delta}\right)\right) \geq 1 - \delta.$$

As described in the proof overview in Sect. 1.3.1, the core argument is nearly identical to the analysis of Greedy Osborne in Sect. 4. Below, we detail the additional probabilistic nuances and describe how to overcome them. Remaining details for the proof of Theorem 9 are deferred to Appendix A.2.

5.1 Bounding the number of iterations

Analogous to the proof of Greedy Osborne (c.f. Lemma 7), the key lemma is that each iteration significantly decreases the potential. The statement and proof are nearly identical. The only difference in the statement of the lemma is that for Random Osborne, this improvement is in *expectation*.

Lemma 8 (Potential decrease of Random Osborne) *Consider any $x \in \mathbb{R}^n$ for which the corresponding scaling $A := \mathbb{D}(e^x) K \mathbb{D}(e^{-x})$ is not ε-balanced. If x' is the next iterate obtained from a Random Osborne update, then*

$$\mathbb{E}\left[\Phi(x) - \Phi(x')\right] \geq \frac{1}{4n} \left(\frac{\Phi(x) - \Phi^*}{d \log \kappa} \vee \varepsilon\right)^2 ,$$

where the expectation is over the algorithm's uniform random choice of update coordinate from $[n]$.

Proof The proof is identical to the proof for Greedy Osborne (c.f. Lemma 7), with only two minor differences. The first is that (19) and (20) are in expectation. The second is that (21) holds with equality by definition of the Random Osborne algorithm. □

Lemma 3 shows that the potential is initially bounded, and Lemma 8 shows that each iteration significantly decreases the potential in expectation. In the analysis of Greedy Osborne, this potential drop is deterministic, and so we immediately concluded that the number of iterations is at most the initial potential divided by the per-iteration decrease (see (24) in Sect. 4). Lemma 9 below shows that essentially the same bound holds in our stochastic setting. Indeed, the expectation bound is exactly this quantity (plus one), and the h.p. bound is the same up to a small constant.

Lemma 9 (Per-iteration expected improvement implies few iterations) *Let $A > a$ and $h > 0$. Let $\{Y_t\}_{t \in \mathbb{N}_0}$ be a stochastic process adapted to a filtration $\{\mathcal{F}_t\}_{t \in \mathbb{N}_0}$ such that $Y_0 \leq A$ a.s., each difference $Y_{t-1} - Y_t$ is bounded within $[0, 2(A-a)]$ a.s., and*

$$\mathbb{E}\left[Y_t - Y_{t+1} \mid \mathcal{F}_t, \, Y_t \geq a\right] \geq h \tag{28}$$

for all $t \in \mathbb{N}_0$. Then the stopping time $\tau := \min\{t \in \mathbb{N}_0 : Y_t \leq a\}$ satisfies

- *(Expectation bound.) $\mathbb{E}[\tau] \leq \frac{A-a}{h} + 1$.*
- *(H.p. bound.) For all $\delta \in (0, 1/e)$, it holds that $\mathbb{P}(\tau \leq \frac{6(A-a)}{h} \log \frac{1}{\delta}) \geq 1 - \delta$.*

The expectation bound in Lemma 9 is proved using Doob's Optional Stopping Theorem, and the h.p. bound using Chernoff bounds; details are deferred to Appendix A.1.

Remark 8 (*Sub-exponential concentration*) Lemma 9 shows that the upper tail of τ decays at a sub-exponential rate. This concentration cannot be improved to a sub-Gaussian rate: indeed, consider X_t i.i.d. Bernoulli with parameter $h \in (0, 1)$, $Y_t = 1 - \sum_{i=1}^t X_i$, $A = 1$, and $a = 0$. Then $\mathbb{P}(\tau \leq N) = 1 - \mathbb{P}(X_1 = \cdots = X_N = 0) = 1 - (1-h)^N$ which is $\approx 1 - \delta$ when $N \approx \frac{1}{h} \log \frac{1}{\delta}$.

5.2 Bounding the final runtime

The key reason that Random Osborne is faster than Greedy Osborne (other than bit complexity) is that its per-iteration runtime is faster for sparse matrices: it is $O(m/n)$ by Observation 2 rather than $O(n)$. In the deterministic setting, the final runtime is

at most the product of the per-iteration runtime and the number of iterations (c.f. Sect. 4). However, obtaining a final runtime bound from a per-iteration runtime and an iteration-complexity bound requires additional tools in the stochastic setting. A similar h.p. bound follows from a standard Chernoff bound. But proving an expectation bound is more nuanced. The natural approach is Wald's equation, which states the the sum of a random number τ of i.i.d. random variables $Z_1, \ldots, Z_\tau$ equals $\mathbb{E}\tau\mathbb{E}Z_1$, so long as τ is independent from $Z_1, \ldots, Z_\tau$ [14, Theorem 4.1.5]. However, in our setting the per-iteration runtimes and the number of iterations are *not* independent. Nevertheless, this dependence is weak enough for the identity to still hold. Formally, we require the following minor technical modifications of the per-iteration runtime bound in Observation 2 and Wald's equation.

Lemma 10 (Per-iteration runtime of Random Osborne, irrespective of history) *Let $\mathcal{F}_{t-1}$ denote the sigma-algebra generated by the first $t-1$ iterates of Random Osborne. Conditional on $\mathcal{F}_{t-1}$, the t-th iteration requires $O(m/n)$ arithmetic operations in expectation.*

Lemma 11 (Minor modification of Wald's equation) *Let $Z_1, Z_2, \ldots$ be i.i.d. nonnegative integrable r.v.'s. Let τ be an integrable $\mathbb{N}$-valued r.v. satisfying $\mathbb{E}[Z_t | \tau \geq t] = \mathbb{E}[Z_1]$ for each $t \in \mathbb{N}$. Then $\mathbb{E}[\sum_{t=1}^{\tau} Z_t] = \mathbb{E}\tau\mathbb{E}Z_1$.*

The proof of Lemma 10 is nearly identical to the proof of Observation 2, and is thus omitted. The proof of Lemma 11 is a minor modification of the proof of the standard Wald's equation in [14]; details in Appendix A.1.

6 Random-Reshuffle Cyclic Osborne converges quickly

Here we show a runtime bound for Random-Reshuffle Cyclic Osborne. See Sect. 1.2 for further discussion, and Sect. 1.3.3 for a proof sketch.

Theorem 10 (Convergence of Random-Reshuffle Cyclic Osborne) *Given a balance-able matrix $K \in \mathbb{R}_{\geq 0}^{n \times n}$ and accuracy $\varepsilon > 0$, Random-Reshuffle Cyclic Osborne solves $\mathrm{ABAL}(K, \varepsilon)$ in T arithmetic operations, where*

- *(Expectation guarantee.) $\mathbb{E}[T] = O(\frac{mn}{\varepsilon}(\frac{1}{\varepsilon} \wedge d) \log \kappa)$.*
- *(H.p. guarantee.) There exists a universal constant $c > 0$ such that for all $\delta > 0$,*

$$\mathbb{P}\left(T \leq c\left(\frac{mn}{\varepsilon}(\tfrac{1}{\varepsilon} \wedge d) \log \kappa \log \tfrac{1}{\delta}\right)\right) \geq 1 - \delta.$$

A straightforward coupling argument with Random Osborne shows the following per-cycle potential decrease bound for Random-Reshuffle Cyclic Osborne.

Lemma 12 (Potential decrease of Random-Reshuffle Cyclic Osborne) *Consider any $x \in \mathbb{R}^n$ for which the corresponding scaling $A := \mathbb{D}(e^x)K\mathbb{D}(e^{-x})$ is not ε-balanced. Let x' be the iterate obtained from x after a cycle of Random-Reshuffle Cyclic Osborne. Then*

$$\mathbb{E}\left[\Phi(x) - \Phi(x')\right] \geq \frac{1}{4n}\left(\frac{\Phi(x) - \Phi^*}{d\log\kappa} \vee \varepsilon\right)^2,$$

where the expectation is over the algorithm's random choice of update coordinates.

Proof By monotonicity of Φ w.r.t. Osborne updates (Lemma 4), the expected decrease in Φ from all n updates in a cycle is at least that from the first update in the cycle. This first update index is uniformly distributed from $[n]$, thus is equivalent to an iteration of Random Osborne. We conclude by applying the per-iteration decrease bound for Random Osborne in Lemma 8.   □

The runtime bound for Random-Reshuffle Cyclic Osborne (Theorem 10) given the expected per-cycle potential decrease (Lemma 12) then follows by an identical argument as the runtime bound for Random Osborne (Theorem 9) given that algorithm's expected per-iteration potential decrease (Lemma 8). The straightforward details are omitted for brevity.

7 Parallelized variants of Osborne converge quickly

Here we show fast runtime bounds for parallelized variants of Osborne's algorithm when given a coloring of G_K (see Sect. 2.5). See Sect. 1.2 for a discussion of these results, and Sect. 1.3.4 for a proof sketch.

Theorem 11 (Convergence of Block Osborne variants) *Consider balancing a balance-able matrix $K \in \mathbb{R}_{\geq 0}^{n \times n}$ to accuracy $\varepsilon > 0$ given a coloring of G_K of size p.*

- *Greedy Block Osborne solves* ABAL(K, ε) *in* $O(\frac{p}{\varepsilon}(\frac{1}{\varepsilon} \wedge d)\log\kappa)$ *rounds and* $O(\frac{mp}{\varepsilon}(\frac{1}{\varepsilon} \wedge d)\log\kappa)$ *total work.*
- *Random Block Osborne solves* ABAL(K, ε) *in* $O(\frac{p}{\varepsilon}(\frac{1}{\varepsilon} \wedge d)\log\kappa)$ *rounds and* $O(\frac{m}{\varepsilon}(\frac{1}{\varepsilon} \wedge d)\log\kappa)$ *total work, in expectation and w.h.p.*
- *Random-Reshuffle Cyclic Block Osborne solves* ABAL(K, ε) *in* $O(\frac{p^2}{\varepsilon}(\frac{1}{\varepsilon} \wedge d)\log\kappa)$ *rounds and* $O(\frac{mp}{\varepsilon}(\frac{1}{\varepsilon} \wedge d)\log\kappa)$ *total work, in expectation and w.h.p.*

Note that the h.p. bounds in Theorem 11 have exponentially decaying tails, just as for the non-parallelized variants (c.f., Theorems 9 and 10; see also Remark 8).

The proof of Theorem 11 is nearly identical to the analysis of the analogous non-parallelized variants in Sects. 4, 5, and 6 above. For brevity, we only describe the differences. First, we show the rounds bounds. For Greedy and Random Block Osborne, the only difference is that the per-iteration potential decrease is now n/p times larger than in Lemmas 7 and 8, respectively. Below we show this modification for Greedy Block Osborne; an identical argument applies for Random Block Osborne after taking an expectation (the inequality (29) then becomes an equality).

Lemma 13 (Potential decrease of Greedy Block Osborne) *Consider any $x \in \mathbb{R}^n$ for which the corresponding scaling $A := \mathbb{D}(e^x)K\mathbb{D}(e^{-x})$ is not ε-balanced. If x' is the*

next iterate obtained from a Greedy Block Osborne update, then

$$\Phi(x) - \Phi(x') \geq \frac{1}{4p} \left(\frac{\Phi(x) - \Phi^*}{d \log \kappa} \vee \varepsilon \right)^2.$$

Proof Let S_ℓ be the chosen block. Using in order Lemma 4, the inequality $-\log(1 - z) \geq z$, the definition of Greedy Block Osborne, re-arranging, and then Proposition 6,

$$\begin{aligned}
\Phi(x) - \Phi(x') &= - \sum_{k \in S_\ell} \log\left(1 - \left(\sqrt{r_k(P)} - \sqrt{c_k(P)}\right)^2\right) \\
&\geq \sum_{k \in S_\ell} \left(\sqrt{r_k(P)} - \sqrt{c_k(P)}\right)^2 \\
&\geq \frac{1}{p} \sum_{\ell=1}^{p} \sum_{k \in S_\ell} \left(\sqrt{r_\ell(P)} - \sqrt{c_\ell(P)}\right)^2 \\
&= \frac{1}{p} \sum_{k=1}^{n} \left(\sqrt{r_k(P)} - \sqrt{c_k(P)}\right)^2 \\
&\geq \frac{1}{4p} \left(\frac{\Phi(x) - \Phi^*}{d \log \kappa} \vee \varepsilon \right)^2.
\end{aligned} \tag{29}$$

□

With this n/p times larger per-iteration potential decrease, the number of rounds required by Greedy and Random Block Osborne is then n/p times smaller than the number of Osborne updates required by their non-parallelized counterparts, establishing the desired rounds bounds in Theorem 11. The rounds bound for Random-Reshuffle Cyclic Block Osborne is then p times that of Random Block Osborne by an identical coupling argument as for their non-parallelized counterparts (see Sect. 6).

Next, we describe the total-work bounds in Theorem 11. For Random-Shuffle Cyclic Block Osborne, every p rounds is a full cycle and therefore requires $\Theta(m)$ work. For Greedy and Random Block Osborne, each round takes work proportional to the number of nonzero entries in the updated block. For Random Block Osborne, this is $\Theta(m/p)$ on average by an identical argument to Observation 2. For Greedy Block Osborne, this could be up to $O(m)$ in the worst case. (Although this is of course significantly improvable if the blocks have balanced sizes.)

Finally, we note that combining Theorem 11 with the extensive literature on parallelized algorithms for coloring bounded-degree graphs yields a fast parallelized algorithm for balancing Δ-uniformly sparse matrices, i.e., matrices K for which G_K has max degree[14] Δ.

Corollary 12 (Parallelized Osborne for uniformly sparse matrices) *There is a parallelized algorithm that, given any Δ-uniformly sparse matrix $K \in \mathbb{R}_{\geq 0}^{n \times n}$, computes an*

---

[14] This is the degree in the undirected graph where (i, j) is an edge if either (i, j) or (j, i) is an edge in G_K.

ε-approximate balancing in $O(\frac{\Delta}{\varepsilon}(\frac{1}{\varepsilon} \wedge d) \log \kappa)$ rounds and $O(\frac{m}{\varepsilon}(\frac{1}{\varepsilon} \wedge d) \log \kappa)$ total work, both in expectation and w.h.p.

Proof The algorithm of [6] computes a $\Delta + 1$ coloring in $O(\Delta) + \frac{1}{2} \log^* n$ rounds, where $\log^*$ is the iterated logarithm. Run Random Block Osborne with this coloring, and apply Theorem 11.   □

We remark that a coloring of size $\Delta + 1$ can be alternatively computed by a simple greedy algorithm in $O(m)$ linear time. Although sequential, this simpler algorithm may be more practical.

8 Numerical precision

So far we have assumed exact arithmetic for simplicity of exposition; here we address numerical precision issues. Note that Osborne iterates can have variation norm up to $O(n \log \kappa)$; see [22, §3] and Lemma 6. For such iterates, operations on the current balancing $\mathbb{D}(e^x) K \mathbb{D}(e^{-x})$—namely, computing row and column sums for an Osborne update—naïvely require arithmetic operations on $O(n \log \kappa)$-bit numbers. Here, we show that there is an implementation that uses numbers with only logarithmically few bits and still achieves the same runtime bounds.[15]

Below, we assume for simplicity that each input entry K_{ij} is represented using $O(\log \frac{K_{\max}}{K_{\min}} + \log \frac{n}{\varepsilon})$ bits. (Or $O(\log \log \frac{K_{\max}}{K_{\min}} + \log \frac{n}{\varepsilon})$ bits if input on the logarithmic scale $\log K_{ij}$, for $(i, j) \in \mathrm{supp}(K)$, see Remark 9.) This assumption is made essentially without loss of generality since after a possible rescaling and truncation of entries to $\pm \varepsilon K_{\min}/n$—which does not change the problem of approximately balancing K to $O(\varepsilon)$ accuracy by Lemma 14—all inputs are represented using this many bits.

Theorem 13 (Osborne variants with low bit-complexity) *There is an implementation of Random Osborne (respectively, Random-Reshuffle Cyclic Osborne, Random Block Osborne, and Random-Reshuffle Cyclic Block Osborne) that uses arithmetic operations over $O(\log \frac{n}{\varepsilon} + \log \frac{K_{\max}}{K_{\min}})$-bit numbers and achieves the same runtime bounds as in Theorem 9 (respectively, Theorem 10, 11, and again 11).*

Moreover, if the matrix K is given as input through the logarithms of its entries $\{\log K_{ij}\}_{(i,j) \in \mathrm{supp}(K)}$, this bit-complexity is improvable to $O(\log \frac{n}{\varepsilon} + \log \log \frac{K_{\max}}{K_{\min}})$.

This result may be of independent interest since the aforementioned bit-complexity issues of Osborne's algorithm are well-known to cause numerical precision issues in practice and have been difficult to analyze theoretically. We note that [32, §5] shows similar bit complexity $O(\log(n\kappa/\varepsilon))$ for an Osborne variant they propose; however, that variant has runtime scaling in n^2 rather than m (see footnote 6). Moreover, our analysis is relatively simple and extends to the related Sinkhorn algorithm for Matrix Scaling (see Appendix B).

Before proving Theorem 13, we make several remarks.

---

[15] Note that Theorem 13 outputs only the balancing vector $x \in \mathbb{R}^n$, not the approximately balanced matrix $A = \mathbb{D}(e^x) K \mathbb{D}(e^{-x})$. If applications require A, this can be computed to polynomially small entrywise additive error using only logarithmically many bits; this is sufficient, e.g., for the application of approximating Min-Mean-Cycle [4, §5.3].

Remark 9 *(Log-domain input)* Theorem 13 gives an improved bit-complexity if K is input through the *logarithms* of its entries. This is useful in an application such as Min-Mean-Cycle where the input is a weighted adjacency matrix W, and the matrix K to balance is the entrywise exponential of (a constant times) W [4, §5].

Remark 10 *(Greedy Osborne requires large bit-complexity)* All known implementations of Greedy Osborne require bit-complexity at least $\tilde{\Omega}(n)$ [32]. The obstacle is the computation (10) of the next update coordinate, which requires computing the *difference* of two log-sum-exp's. It can be shown that computing this difference to a constant multiplicative error suffices. However, this still requires at least computing the sign of the difference, which importantly, precludes dropping small summands in each log-sum-exp—a key trick used for computing an individual log-sum-exp to additive error with low bit-complexity (Lemma 17).

We now turn to the proof of Theorem 13. For brevity, we establish this only for Random Osborne; the proofs for the other variants are nearly identical. Our implementation of Random Osborne makes three minor modifications to the exact-arithmetic implementation in Algorithm 1. We emphasize that these modifications are in line with standard implementations of Osborne's algorithm in practice, see Remark 5.

1. In a pre-processing step, compute $\{\log K_{ij}\}_{(i,j)\in\mathrm{supp}(K)}$ to additive accuracy $\gamma = \Theta(\varepsilon/n)$.
2. Truncate each Osborne iterate $x^{(t)}$ entrywise to additive accuracy $\tau = \Theta(\varepsilon^2/n)$.
3. Compute Osborne updates to additive accuracy τ by using log-sum-exp computation tricks (Lemma 17) and using K_{ij} only through the truncated values $\log K_{ij}$ computed in step 1.

Step 1 is performed only when K is not already input on the logarithmic scale, and is responsible for the $O(\log(K_{\max}/K_{\min}))$ bit-complexity. To argue about these modifications, we collect several helpful observations, the proofs of which are simple and deferred to Appendix A.3 for brevity.

Lemma 14 (Approximately balancing an approximate matrix suffices) *Let $K, \tilde{K} \in \mathbb{R}_{\geq 0}^{n\times n}$ such that $\mathrm{supp}(K) = \mathrm{supp}(\tilde{K})$ and the ratio $K_{ij}/\tilde{K}_{ij}$ of nonzero entries is bounded in $[1 - \gamma, 1 + \gamma]$ for some $\gamma \in (0, 1/3)$. If x is an ε-balancing of K, then x is an $(\varepsilon + 6n\gamma)$-balancing of $\tilde{K}$.*

Lemma 15 (Stability of log-sum-exp) *The function $z \mapsto \log(\sum_{i=1}^{n} e^{z_i})$ is 1-Lipschitz with respect to the ℓ_∞ norm on $\mathbb{R}^n$.*

Lemma 16 (Stability of potential function) *Let $K \in \mathbb{R}_{\geq 0}^{n\times n}$. Then $\Phi(x) := \log(\sum_{ij} e^{x_i - x_j} K_{ij})$ is 2-Lipschitz with respect to the ℓ_∞ norm on $\mathbb{R}^n$.*

Lemma 17 (Computing log-sum-exp with low bit-complexity) *Let $z_1, \ldots, z_n \in \mathbb{R}$ and $\tau > 0$ be given as input, each represented using b bits. Then $\log(\sum_{i=1}^{n} e^{z_i})$ can computed to $\pm\tau$ in $O(n)$ operations on $O(b + \log(\frac{n}{\tau}))$-bit numbers.*

Proof (Theorem 13) **Error and runtime analysis.**

1. Let $\tilde{K}$ be the matrix whose ij-th entry is the exponential of the truncated log K_{ij} for $(i, j) \in \mathrm{supp}(K)$, and 0 otherwise. The effect of step (1) is to balance $\tilde{K}$ rather than K. But by Lemma 14, this suffices since an $O(\varepsilon)$ balancing of $\tilde{K}$ is an $O(\varepsilon + n\gamma) = O(\varepsilon)$ balancing of K.

2,3. The combined effect is that: given the previous Osborne iterate $x^{(t-1)}$, the next iterate $x^{(t)}$ differs from the value it would have in the exact-arithmetic implementation by $O(\tau)$ in ℓ_∞ norm. By Lemma 16, this changes $\Phi(x^{(t)})$ by at most $O(\tau)$. By appropriately choosing the constant in the definition of $\tau = \Theta(\varepsilon^2/n)$, this decreases each iteration's expected progress (Lemma 8) by at most a factor of $1/2$. The proof of Theorem 9 then proceeds otherwise unchanged, resulting in a final runtime at most 2 times larger.

Bit-complexity analysis.

1. Consider $(i, j) \in \mathrm{supp}(K)$. Since $\log K_{ij} \in [\log K_{\min}, \log K_{\max}]$ and are stored to additive accuracy $\gamma = \Theta(\varepsilon/n)$, the bit-complexity for storing $\log K_{ij}$ is

$$O\left(\log \frac{\log K_{\max} - \log K_{\min}}{\gamma}\right) = O\left(\log \frac{n}{\varepsilon} + \log \log \frac{K_{\max}}{K_{\min}}\right).$$

2. Since the coordinates of each Osborne iterate are truncated to additive accuracy $\tau = \Theta(\varepsilon^2/n)$ and have modulus at most $d \log \kappa$ by Lemma 6, they require bit-complexity

$$O\left(\log \frac{(d \log \kappa) - (-d \log \kappa)}{\tau}\right) = O\left(\log \frac{n}{\varepsilon} + \log \log \frac{K_{\max}}{K_{\min}}\right).$$

3. By Lemma 17, the Osborne update requires bit-complexity $O(\log \frac{n}{\tau}) = O(\log \frac{n}{\varepsilon})$.

$\square$

9 Conclusion

We conclude with several open questions:

1. Can one establish matching runtime lower bounds for the variants of Osborne's algorithm? The only existing lower bound is [32, Theorem 6.1], and there is a large gap between this and the current upper bounds.

2. Does any variant of Cyclic Osborne run in near-linear time? The best known runtime bound for Round-Robin Cyclic Osborne scales as roughly mn^2 [32], and the runtime bound we show for Random-Reshuffle Cyclic Osborne scales as roughly mn (Theorem 10).

3. Is there a provable gap between the (worst-case) performance of Random Osborne, Random-Reshuffle Cyclic Osborne, and Round-Robin Cyclic Osborne? The existence of such gaps in the more general context of Coordinate Descent for convex optimization is an active area of research with recent breakthroughs [24, 43, 44].

4. Empirically, Osborne's algorithm often significantly outperforms its worst-case bounds. Is it possible to prove faster average-case runtimes for "typical" matrices arising in practice? (This is the analog to the third open question in [40, §6] for Max-Balancing.)

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

A Deferred proofs

A.1 Probabilistic helper lemmas

Several times we make use of the following standard (martingale) version of multiplicative Chernoff bounds, see, e.g., [28, §4].

Lemma 18 (Multiplicative Chernoff Bounds) *Let $X_1, \ldots X_n$ be supported in $[0, 1]$, be adapted to some filtration $\mathcal{F}_0 = \{\emptyset, \Omega\}, \mathcal{F}_1, \ldots, \mathcal{F}_n$, and satisfy $\mathbb{E}[X_i | \mathcal{F}_{i-1}] = p$ for each $i \in [n]$. Denote $X := \sum_{i=1}^{n} X_i$ and $\mu := \mathbb{E}X$. Then*

- *(Lower tail.) For any $\Delta \in (0, 1)$, $\mathbb{P}(X \leq (1 - \Delta)\mu) \leq e^{-\Delta^2 \mu/2}$.*
- *(Upper tail.) For any $\Delta \geq 1$, $\mathbb{P}(X \geq (1 + \Delta)\mu) \leq e^{-\Delta\mu/3}$.*

Proof (Lemma 9) Expectation bound. Define $Z_t := Y_t + ht$. Then $Z_t^{\tau} := Z_{t \wedge \tau}$ is a stopped supermartingale with respect to $\mathcal{F}_t$. Thus by Doob's Optional Stopping Theorem [14] (which may be invoked by a.s. boundedness),

$$A \geq \mathbb{E}Z_0 \geq \mathbb{E}Z_{\tau-1} = \mathbb{E}Y_{\tau-1} + h(\mathbb{E}\tau - 1) \geq a + h(\mathbb{E}\tau - 1)$$

Re-arranging yields $\mathbb{E}[\tau] \leq \frac{A-a}{h} + 1$, as desired.

High probability bound. For shorthand, denote $B := 2(A - a)$ and $N := \lceil 3B/h \log \frac{1}{\delta} \rceil$. By definition of τ, telescoping, and then the bound on Y_0,

$$\mathbb{P}(\tau > N) = \mathbb{P}(Y_N > a) = \mathbb{P}\left(\sum_{t=1}^{N} (Y_{t-1} - Y_t) < Y_0 - a \right)$$

$$\leq \mathbb{P}\left(\sum_{t=1}^{N} (Y_{t-1} - Y_t) < A - a \right) \tag{30}$$

To bound (30), define the process $X_t := (Y_{t-1} - Y_t)/B$. Each X_t is a.s. bounded within $[0, 1]$ by the bounded-difference assumption on Y_t. Thus by an application of the lower-tail Chernoff bound in Lemma 18 (combined with a simple stochastic domination argument since $\mathbb{E}[X_t|\mathcal{F}_{t-1}] \geq h/B$ rather than exactly equal), and then the choice of N, we conclude that

$$\mathbb{P}\left(\sum_{t=1}^{N} (Y_{t-1} - Y_t) < A - a \right) = \mathbb{P}\left(\sum_{t=1}^{N} X_t < \frac{A-a}{B} \right)$$

$$\leq \exp\left(-\left(1 - \frac{A-a}{Nh}\right)^2 \frac{Nh}{2B} \right) \leq \delta. \tag{31}$$

$\square$

Proof (Lemma 11) Observe that

$$\mathbb{E}\left[\sum_{t=1}^{\tau} Z_t \right] = \sum_{T=1}^{\infty} \mathbb{E}\left[\sum_{t=1}^{\tau} Z_t \mathbb{1}_{\tau=T} \right] = \sum_{T=1}^{\infty} \sum_{t=1}^{T} \mathbb{E}[Z_t \mathbb{1}_{\tau=T}]$$

$$= \sum_{t=1}^{\infty} \sum_{T=t}^{\infty} \mathbb{E}[Z_t \mathbb{1}_{\tau=T}] = \sum_{t=1}^{\infty} \mathbb{E}\left[Z_t \mathbb{1}_{\tau \geq t} \right],$$

where the third equality above is because the assumption $Z_i \geq 0$ allows us to invoke Fubini's Theorem. Now since $\mathbb{E}\left[Z_t \mathbb{1}_{\tau \geq t} \right] = \mathbb{E}[Z_t|\tau \geq t]\mathbb{P}(\tau \geq t) = \mathbb{E}[Z_t]\mathbb{P}(\tau \geq t)$ by assumption, we conclude that $\mathbb{E}[\sum_{t=1}^{\tau} Z_t] = \mathbb{E}[Z_1](\sum_{t=1}^{\infty} \mathbb{P}(\tau \geq t)) = \mathbb{E}[Z_1]\mathbb{E}[\tau]$. $\square$

A.2 Proof of Theorem 9

Let $x^{(0)} = \mathbf{0}, x^{(1)}, x^{(2)}, \ldots$ denote the iterates, and $\{\mathcal{F}_t := \sigma(x_1, \ldots, x_t)\}_t$ denote the corresponding filtration. Define the stopping time $\tau := \min\{t \in \mathbb{N}_0 : \mathbb{D}(e^x) K \mathbb{D}(e^{-x}) \text{ is } \varepsilon\text{-balanced}\}$. By Lemma 8,

$$\mathbb{E}\left[\Phi(x^{(t)}) - \Phi(x^{(t+1)}) \mid \mathcal{F}_t, t \leq \tau \right] \geq \frac{1}{4n}\left(\frac{\Phi(x^{(t)}) - \Phi^*}{d \log \kappa} \vee \varepsilon \right)^2. \tag{32}$$

Case 1 $\varepsilon^{-1} \le d$. Here, we establish the $O(m\varepsilon^{-2} \log \kappa)$ runtime bound both in expectation and w.h.p. To this end, let T_t denote the runtime of iteration t, where (solely for analysis purposes) we consider also $t > \tau$ if the algorithm had continued after convergence. Define Y_t to be $\Phi(x^{(t)})$ if $t \le \tau$, and otherwise $\Phi(x^{(t)}) - (t - \tau)\varepsilon^2/4n$ if $t > \tau$. By (32), we have

$$\mathbb{E}\left[Y_t - Y_{t+1} \mid \mathcal{F}_t, Y_t \ge 0\right] \ge \frac{\varepsilon^2}{4n}. \tag{33}$$

For both expected and h.p. bounds below, we apply Lemma 9 to the process Y_t with $A = \log \kappa$ (by Lemma 3), $a = 0$, and $h = \varepsilon^2/4n$ (by (33)).

<u>Expectation bound.</u> The expectation bound in Lemma 9 implies $\mathbb{E}[\tau] \le 4n\varepsilon^{-2} \log \kappa + 1$. Since each iteration has expected runtime $\mathbb{E}[T_t|\mathcal{F}_{t-1}] = O(m/n)$ by Lemma 10, Lemma 11 ensures that the total expected runtime is $\mathbb{E}T = \mathbb{E}[\sum_{t=1}^{\tau} T_t] = \mathbb{E}\tau \mathbb{E}T_1 = O(m\varepsilon^{-2} \log \kappa)$.

<u>H.p. bound.</u> For shortand, denote $U := 24n\varepsilon^{-2} \log \kappa \log \frac{2}{\delta}$. The h.p. bound in Lemma 3 implies that $\mathbb{P}(\tau > U) \le \delta/2$. By Lemma 10, there is some constant $c > 0$ such that $\mathbb{E}[T_t] = cm/n$. Since the T_t are independent, a Chernoff bound (Lemma 18) implies that $\mathbb{P}(\sum_{t=1}^{U} T_t \le 2cUm/n) \le \delta/2$. Therefore, a union bound implies that with probability at least $1 - \delta$, the total runtime $T = \sum_{t=1}^{\tau} T_\tau$ is at most $2cUm/n = 48cm\varepsilon^{-2} \log \kappa \log \frac{2}{\delta}$.

Case 2 $\varepsilon^{-1} \ge d$. Here, we establish the $O(md\varepsilon^{-1} \log \kappa)$ runtime bound in expectation and w.h.p. Define α, τ_1, τ_2, $\tau_{1,i}$, and ϕ_i as in the analysis of Greedy Osborne (see Sect. 4).

<u>Expectation bound.</u> To bound $\mathbb{E}\tau_2$, define Y_t and apply Lemma 9 as in case 1 above (except now with $A = \varepsilon d \log \kappa$) to establish that

$$\mathbb{E}\tau_2 \le \frac{\varepsilon d \log \kappa}{\varepsilon^2/4n} + 1 = \frac{4nd \log \kappa}{\varepsilon} + 1. \tag{34}$$

Next, we bound $\mathbb{E}\tau_1$. Consider subphase $\tau_{1,i}$ for $i \in [N]$. By an application of Lemma 9 on the process $\Phi(x^{(t-\tau_{1,i-1})})$ where $A = \phi_{i-1}$, $a = \phi_i$, and $h = \phi_i^2/(4nd^2 \log^2 \kappa)$ from (32), $\mathbb{E}\tau_{1,i} \le \frac{4nd^2 \log^2 \kappa}{\phi_i} + 1$. Thus $\mathbb{E}\tau_1 = \sum_{i=1}^{N} \mathbb{E}\tau_{1,i} \le 4nd^2 \log^2 \kappa (\sum_{i=1}^{N} \frac{1}{\phi_i}) + N$. Since $\sum_{i=1}^{N} \frac{1}{\phi_i} \le \frac{4}{\varepsilon d \log \kappa}$,

$$\mathbb{E}\tau_1 \le \frac{16nd \log \kappa}{\varepsilon} + \log_2 \left\lceil \frac{1}{\varepsilon d} \right\rceil. \tag{35}$$

Combining (34) and (35) establishes that $\mathbb{E}\tau = \mathbb{E}\tau_1 + \mathbb{E}\tau_2 \le 21nd\varepsilon^{-1} \log \kappa$. By the $O(m/n)$ per-iteration expected runtime bound in Lemma 10 and the variant of Wald's equation in Lemma 11, the total expected runtime is therefore at most $\mathbb{E}T \le O(m/n) \cdot \mathbb{E}\tau = O(md\varepsilon^{-1} \log \kappa)$.

<u>H.p. bound.</u> By Lemma 9, $\mathbb{P}(\tau_2 > 24nd\varepsilon^{-1} \log \kappa \log \frac{4}{\delta}) \le \delta/4$. To bound the first phase, define $p_i := \delta/2^{N-i+3}$ for each $i \in [N]$. By Lemma 9, $\mathbb{P}(\tau_{1,i} > (24nd^2 \log^2 \kappa \log 1/p_i)/\phi_i) \le p_i$. Note that $\sum_{i=1}^{N} \frac{\log 1/p_i}{\phi_i} = \frac{1}{\phi_N} \sum_{j=0}^{N-1} 2^{-j} (\log 8/\delta +$

$j \log 2) \leq \frac{1}{\phi_N} \sum_{j=0}^{\infty} 2^{-j}(\log 8/\delta + j \log 2) = \frac{2 \log 8/\delta + 2 \log 2}{\phi_N} \leq \frac{6 \log 8/\delta}{\varepsilon d \log \kappa}$. Thus by a union bound, with probability at most $\sum_{i=1}^{N} p_i \leq \delta/4$, the first phase has length at most $\tau_1 = \sum_{i=1}^{N} \tau_{1,i} \leq 144 n d \varepsilon^{-1} \log \kappa \log \frac{8}{\delta}$. We conclude by a further union bound that, with probability at least $1 - \delta/2$, the total number of iterations is at most $\tau = \tau_1 + \tau_2 \leq 168 n d \varepsilon^{-1} \log \kappa \log \frac{8}{\delta}$. The proof is complete by an identical Chernoff bound argument as in case 1 above.

A.3 Proofs for Sect. 8

Proof (Lemma 14) Let $A := \mathbb{D}(e^x) K \mathbb{D}(e^{-x})$ denote the corresponding scaling of K, and $P := A/\sum_{ij} A_{ij}$ denote its normalization. Similarly for $\tilde{A}$ and $\tilde{P}$. Note that each nonzero entry $\tilde{P}_{ij}$ approximates P_{ij} to a multiplicative factor within $[(1 - \gamma)/(1 + \gamma), (1+\gamma)/(1-\gamma)] \subset [1-3\gamma, 1+3\gamma]$, where the last step used the assumption that $\gamma < 1/3$. Thus each row marginal $r_k(\tilde{P})$ approximates $r_k(P)$ to the same multiplicative factor, and similarly for the column marginals. Since P and $\tilde{P}$ are normalized, this implies the additive approximations $|r_k(P) - r_k(\tilde{P})| \leq 3\gamma$, and similarly for the columns. Thus by the triangle inequality, $\|r(P) - c(P)\|_1 \leq \|r(\tilde{P}) - c(\tilde{P})\|_1 + 6n\gamma$. □

Proof (Lemma 15) Let $x, y \in \mathbb{R}^n$. Since $\min_i(a_i/b_i) \leq (\sum_{i=1}^{n} a_i)/(\sum_{i=1}^{n} b_i) \leq \max_i(a_i/b_i)$ for any $a, b \in \mathbb{R}_{>0}^n$,

$$\log \sum_{i=1}^{n} e^{x_i} - \log \sum_{i=1}^{n} e^{y_i} = \log \frac{\sum_{i=1}^{n} e^{x_i}}{\sum_{i=1}^{n} e^{y_i}} \leq \log \max_i e^{x_i - y_i}$$

$$= \max_i x_i - y_i \leq \|x - y\|_\infty,$$

and similarly $\log \sum_{i=1}^{n} e^{x_i} - \log \sum_{i=1}^{n} e^{y_i} \geq \log \min_i e^{x_i - y_i} = \min_i x_i - y_i \geq -\|x - y\|_\infty$. We conclude that $|\log \sum_{i=1}^{n} e^{x_i} - \log \sum_{i=1}^{n} e^{y_i}| \leq \|x - y\|_\infty$. □

Proof (Lemma 16) Let $x, y \in \mathbb{R}^n$. Clearly $|(x_i - x_j) - (y_i - y_j)| \leq 2\|x - y\|_\infty$ for any $i, j \in [n]$. Thus by Lemma 15, $|\Phi(x) - \Phi(y)| = |\log(\sum_{(i,j)\in\text{supp}(K)} e^{x_i - x_j + \log K_{ij}}) - \log(\sum_{(i,j)\in\text{supp}(K)} e^{y_i - y_j + \log K_{ij}})| \leq 2\|x - y\|_\infty$. □

Proof (Lemma 17) Since $\log \sum_{i=1}^{n} e^{z_i} = \max_j z_j + \log \sum_{i=1}^{n} e^{z_i - (\max_j z_j)}$, we may assume without loss of generality after translation that each $z_i \leq 0$ and at least one $z_i = 0$. Since we need only approximate $\log \sum_{i=1}^{n} e^{z_i}$ to $\pm\tau$ accuracy, we can truncate each z_i to additive accuracy $\pm O(\tau)$ by Lemma 15, and also drop all z_i below $-\log \frac{n}{O(\tau)}$. To summarize, in order to compute $\log \sum_{i=1}^{n} e^{z_i}$ to $\pm\tau$, it suffices to compute $\log \sum_{i=1}^{k} e^{\tilde{z}_i}$ to $\pm O(\tau)$ where $k \leq n$, each $\tilde{z}_i \in [-\log \frac{n}{O(\tau)}, 0]$, and each $\tilde{z}_i$ is represented by a number with at most $O(\log(\frac{\log(n/\tau)}{\tau})) = O(\log \frac{1}{\tau} + \log \log n)$ bits. Now to compute $\log \sum_{i=1}^{k} e^{\tilde{z}_i}$ to $\pm O(\tau)$, we can tolerate computing each $e^{\tilde{z}_i}$ to multiplicative accuracy $(1 \pm O(\tau))$. Thus since $e^{\tilde{z}_i} \geq O(\tau/n)$, we can tolerate computing each $e^{\tilde{z}_i}$ to additive accuracy $\pm O(\tau^2/n)$. Since $e^{\tilde{z}_i} \in [0, 1]$, it therefore suffices to compute $e^{\tilde{z}_i}$ using $O(\log \frac{1}{\tau^2/n}) = O(\log \frac{n}{\tau})$ bits of precision. □

B Connections to Matrix Scaling and Sinkhorn's algorithm

Here, we continue the discussion in Remark 4 by briefly mentioning two further connections between Osborne's algorithm for Matrix Balancing and Sinkhorn's algorithm for Matrix Scaling.

Parallelizability In contrast to Osborne's algorithm for Matrix Balancing, Sinkhorn's algorithm for Matrix Scaling is so-called "embarassingly parallelizable". We briefly explain this in terms of the connection between parallelizability and graph coloring (see Sect. 2.5). For the Matrix Scaling problem on $K \in \mathbb{R}_{\geq 0}^{m \times n}$, the associated graph has vertex set $L \cup R$ where $|L| = m$ and $|R| = n$, and edge set $\{(i, j) : i \in [m], j \in [n], K_{ij} \neq 0\}$. This graph is *bipartite* and thus trivially 2-*colorable*, which is why Sinkhorn's algorithm can safely update all coordinates in L or R in parallel.

Bit-complexity In Theorem 13, we showed that many variants of Osborne's algorithm can be implemented over numbers with logarithmically few bits, and still achieve the same runtime bounds. By a nearly identical argument, it can be shown that the analogous result applies to Sinkhorn's algorithm. This saves a similar factor of up to roughly $O(n)$ in the bit-complexity for poorly connected inputs. Moreover, this modification is also helpful for well-connected inputs, in particular for the application of Optimal Transport, where the matrix K to scale is dense yet has exponentially large entries which require bit-complexity $O(L(\log n)/\varepsilon)$ in the notation of [3, Remark 1]. This modification reduces the bit-complexity to only logarithmic size $O(\log(Ln/\varepsilon))$.

References

1. Allen-Zhu, Z., Li, Y., Oliveira, R., Wigderson, A.: Much faster algorithms for matrix scaling. In: Symposium on the Foundations of Computer Science (FOCS). IEEE (2017)
2. Allen-Zhu, Z., Qu, Z., Richtárik, P., Yuan, Y.: Even faster accelerated coordinate descent using non-uniform sampling. In: International Conference on Machine Learning (ICML), pp. 1110–1119 (2016)
3. Altschuler, J., Weed, J., Rigollet, P.: Near-linear time approximation algorithms for optimal transport via Sinkhorn iteration. In: Conference on Neural Information Processing Systems (NeurIPS) (2017)
4. Altschuler, J.M., Parrilo, P.A.: Approximating Min-Mean-Cycle for low-diameter graphs in near-optimal time and memory. SIAM J. Optim. (2022, to appear)
5. Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.: LAPACK Users' Guide, 3rd edn. Society for Industrial and Applied Mathematics, Philadelphia, PA (1999)
6. Barenboim, L., Elkin, M., Kuhn, F.: Distributed (Δ+1)-coloring in linear (in Δ) time. SIAM J. Comput. **43**(1), 72–95 (2014)
7. Bertsekas, D.P., Tsitsiklis, J.N.: Parallel and Distributed Computation: Numerical Methods, vol. 23. Prentice Hall Englewood Cliffs, NJ (1989)
8. Bollobás, B.: Random Graphs, vol. 73. Cambridge University Press, Cambridge (2001)
9. Chakrabarty, D., Khanna, S.: Better and simpler error analysis of the Sinkhorn-Knopp algorithm for matrix scaling. In: Symposium on Simplicity in Algorithms (SOSA) (2018)
10. Chen, T.Y.: Balancing sparse matrices for computing eigenvalues. Master's thesis, UC Berkeley (1998)
11. Chen, T.Y., Demmel, J.W.: Balancing sparse matrices for computing eigenvalues. Linear Algebra Appl. **309**(1–3), 261–287 (2000)
12. Cohen, M.B., Madry, A., Tsipras, D., Vladu, A.: Matrix scaling and balancing via box constrained Newton's method and interior point methods. In: Symposium on the Foundations of Computer Science (FOCS), pp. 902–913. IEEE (2017)
13. Deza, M.M., Deza, E.: Encyclopedia of Distances, pp. 1–583. Springer, New York (2009)
14. Durrett, R.: Probability: Theory and Examples. Cambridge University Press, Cambridge (2010)

15. Dvurechensky, P., Gasnikov, A., Kroshnin, A.: Computational optimal transport: complexity by accelerated gradient descent is better than by Sinkhorn's algorithm. In: International Conference on Machine Learning (ICML) (2018)

16. Eaves, B.C., Hoffman, A.J., Rothblum, U.G., Schneider, H.: Line-sum-symmetric scalings of square nonnegative matrices. In: Mathematical Programming Essays in Honor of George B. Dantzig Part II, pp. 124–141. Springer (1985)

17. Garey, M.R., Johnson, D.S.: The complexity of near-optimal graph coloring. J. ACM **23**(1), 43–49 (1976)

18. Goulet, V., Dutang, C., Maechler, M., Firth, D., Shapira, M., Stadelmann, M., et al.: expm: Matrix exponential. R package version 0.99-0 (2013)

19. Gurvits, L., Yianilos, P.N.: The deflation-inflation method for certain semidefinite programming and maximum determinant completion problems. Technical report, NECI (1998)

20. Higham, N.J.: The scaling and squaring method for the matrix exponential revisited. SIAM J. Matrix Anal. Appl. **26**(4), 1179–1193 (2005)

21. Idel, M.: A review of matrix scaling and Sinkhorn's normal form for matrices and positive maps. arXiv preprint arXiv:1609.06349 (2016)

22. Kalantari, B., Khachiyan, L., Shokoufandeh, A.: On the complexity of matrix balancing. SIAM J. Matrix Anal. Appl. **18**(2), 450–463 (1997)

23. Karp, R.M.: Reducibility among combinatorial problems. In: Complexity of Computer Computations, pp. 85–103. Springer (1972)

24. Lee, C.P., Wright, S.J.: Random permutations fix a worst case for cyclic coordinate descent. IMA J. Numer. Anal. **39**(3), 1246–1275 (2019)

25. Mai, V.S., Battou, A.: Asynchronous distributed matrix balancing and application to suppressing epidemic. In: 2019 American Control Conference (ACC), pp. 2177–2182. IEEE (2019)

26. MathWorks: balance: diagonal scaling to improve eigenvalue accuracy. https://www.mathworks.com/help/matlab/ref/balance.html

27. MathWorks: eig: eigenvalues and eigenvectors. https://www.mathworks.com/help/matlab/ref/eig.html

28. Mitzenmacher, M., Upfal, E.: Probability and Computing: Randomization and probabilistic techniques in algorithms and data analysis. Cambridge University Press, Cambridge (2017)

29. Nemirovski, A., Rothblum, U.: On complexity of matrix scaling. Linear Algebra Appl. **302**, 435–460 (1999)

30. Nesterov, Y., Stich, S.U.: Efficiency of the accelerated coordinate descent method on structured optimization problems. SIAM J. Optim. **27**(1), 110–123 (2017)

31. Osborne, E.: On pre-conditioning of matrices. J. ACM **7**(4), 338–345 (1960)

32. Ostrovsky, R., Rabani, Y., Yousefi, A.: Matrix balancing in l_p norms: bounding the convergence rate of Osborne's iteration. In: Symposium on Discrete Algorithms (SODA), pp. 154–169. SIAM (2017)

33. Ostrovsky, R., Rabani, Y., Yousefi, A.: Strictly balancing matrices in polynomial time using Osborne's iteration. In: International Colloquium on Automata, Languages and Programming (ICALP) (2018)

34. Parlett, B.N., Reinsch, C.: Balancing a matrix for calculation of eigenvalues and eigenvectors. Numerische Mathematik **13**(4), 293–304 (1969)

35. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical recipes 3rd edition: the art of scientific computing. Cambridge University Press, Cambridge (2007)

36. RDocumentation: Balance a square matrix via LAPACK's dgebal. https://www.rdocumentation.org/packages/expm/versions/0.99-1.1/topics/balance

37. Rothblum, U.G., Schneider, H., Schneider, M.H.: Scaling matrices to prescribed row and column maxima. SIAM J. Matrix Anal. Appl. **15**(1), 1–14 (1994)

38. Schneider, H., Schneider, M.H.: Max-balancing weighted directed graphs and matrix scaling. Math. Oper. Res. **16**(1), 208–222 (1991)

39. Schneider, M.H., Zenios, S.A.: A comparative study of algorithms for matrix balancing. Oper. Res. **38**(3), 439–455 (1990)

40. Schulman, L.J., Sinclair, A.: Analysis of a classical matrix preconditioning algorithm. J. ACM **64**(2), 9 (2017)

41. Sinkhorn, R.: Diagonal equivalence to matrices with prescribed row and column sums. Am. Math. Mon. **74**(4), 402–405 (1967)

42. Smith, B.T., Boyle, J.M., Garbow, B., Ikebe, Y., Klema, V., Moler, C.: Matrix Eigensystem Routines - EISPACK Guide, vol. 6. Springer, New York (2013)

43. Sun, R., Luo, Z.Q., Ye, Y.: On the efficiency of random permutation for ADMM and coordinate descent. Math. Oper. Res. **45**(1), 233–271 (2020)
44. Sun, R., Ye, Y.: Worst-case complexity of cyclic coordinate descent: $O(n^2)$ gap with randomized version. Math. Prog. **185**(1), 487–520 (2021)
45. Tarjan, R.: Depth-first search and linear graph algorithms. SIAM J. Comput. **1**(2), 146–160 (1972)
46. Tomlin, J.A.: A new paradigm for ranking pages on the world wide web. In: Proceedings of the 12th international conference on World Wide Web, pp. 350–355 (2003)
47. Ward, R.C.: Numerical computation of the matrix exponential with accuracy estimate. SIAM J. Numer. Anal. **14**(4), 600–610 (1977)
48. Wright, S.J.: Coordinate descent algorithms. Math. Progr. **151**(1), 3–34 (2015)
49. Young, N.E., Tarjan, R.E., Orlin, J.B.: Faster parametric shortest path and minimum-balance algorithms. Networks **21**(2), 205–221 (1991)
50. Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number. In: Symposium on the Theory of Computing (STOC), pp. 681–690. ACM (2006)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.