Transport and Beyond: Efficient Optimization over Probability Distributions

by

Jason M. Altschuler

B.S.E., Princeton University, 2016
S.M., Massachusetts Institute of Technology, 2018

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of Doctor of Philosophy
at the Massachusetts Institute of Technology

September 2022

Author: _____

Department of Electrical Engineering and Computer Science
August 26, 2022

Certified by: _____

Pablo A. Parrilo
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by: _____

Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Transport and Beyond: Efficient Optimization over Probability Distributions

by

Jason M. Altschuler

Submitted to the MIT EECS Department in September 2022,
in partial fulfillment of the requirements for the Ph.D. degree

Abstract

The core of classical optimization focuses on the setting where decision variables are vectors in $\mathbb{R}^n$. However, modern applications throughout machine learning, applied mathematics, and engineering demand high-dimensional optimization problems where decision variables are probability distributions. Can such optimization problems be solved efficiently? This thesis presents two interrelated lines of work in this direction through the common thread of Optimal Transport. A unifying theme is the optimization of joint probability distributions with constrained marginals.

Part I of this thesis considers Optimal Transport and other optimization problems over joint distributions with two constrained marginals. Such tasks are fundamental in alignment problems, matrix problems, graph problems, and more. Chapters 2-4 establish near-linear runtimes for approximation algorithms for several classical problems under this umbrella: Optimal Transport, Minimum-Mean-Cycle, Matrix Balancing, and Matrix Scaling. Two recurring key themes are the use of entropic regularization for exploiting separability of optimization constraints, and the use of probabilistic inequalities for obtaining dimension-free convergence bounds. A dictionary is presented that unifies these various problems, which were historically studied in disparate communities.

Part II of this thesis considers Multimarginal Optimal Transport (MOT) and other optimization problems over joint distributions with many constrained marginals. Despite the syntactic similarities with the problems in part I, these problems require fundamentally different algorithms and analyses. The key issue limiting the many applications of MOT is that in general, MOT requires exponential time in the number of marginals k and their support sizes n. Chapters 5-6 develop a general theory about what "structure" makes MOT solvable in time

that is polynomial in n and k. We demonstrate this general theory on applications in diverse fields ranging from operations research to data science to fluid dynamics to quantum chemistry. Chapter 7 dedicates special attention to the popular MOT application of Wasserstein barycenters—resolving the complexity of this problem and uncovering the subtle dependence of the dimension on the answer.

Thesis Supervisor: Pablo A. Parrilo
Title: Professor of Electrical Engineering and Computer Science

Dedicated to Matthew Brennan (1994-2021)

"*How's the research going?*" — Pablo
"*I'm really confused.*" — Jason
"*Fantastic! From confusion comes enlightment, or something like that.*" — Pablo

(A common start to our wonderful meetings, slightly paraphrased.)

Contents

Acknowledgements

I am forever grateful to my advisor Pablo Parrilo, who has gone so far above and beyond what I could have ever hoped for from a PhD advisor. Working so closely with someone as brilliant, creative, incisive, and supportive as Pablo has been a truly inspiring experience. I am amazed by how Pablo sees the world of mathematics so cohesively, and it has been a particularly delightful recurrence in our meetings when Pablo shows me new connections between areas of mathematics that I would never have expected. I also wish to specially thank Pablo for his extraordinary generosity with his time and advice, and for his seemingly unbounded compassion and thoughtfulness as a mentor, confidant, and friend these past years.

I am deeply thankful to my committee members: Philippe Rigollet and Guy Bresler. I have worked with Philippe and his group since I started at MIT, and throughout Philippe has been so incredibly kind to me. I wish to thank Philippe for welcoming me into his group meetings, for bringing me onto fun research projects, for always finding time to chat with me about research, and most of all for being extraordinarily encouraging. Philippe is a visionary, somehow able to see far beyond the current frontiers of research, and it has been an honor to learn from him and watch him do this in real time. Guy has taught me so much—everything from probability to ski tricks. I wish to thank Guy for the seasons pass to his group meetings, for being a constant source of encouragement, and also for being so thoughtful. I am so impressed by how Guy seeks moonshot projects and is not discouraged when others in the field have long since given up. This tenacity is incredible and is a model for the type of researcher I hope to become.

I've been very fortunate to work with fantastic collaborators: Francis Bach, Aditya Bhaskara, Enric Boix, Matt Brennan, Victor-Emmanuel Brunel, Sinho Chewi, Thomas Fu, Patrik Gerber, Alan Malek, Vahab Mirrokni, Jon Niles-Weed, Pablo Parrilo, Philippe Rigollet, Afshin Rostamizadeh, Alessandro Rudi, Austin

Stromme, Kunal Talwar, Elizabeth Yang, Morteza Zadimoghaddam—and many more who I've learned so much from but haven't written a manuscript with yet (hopefully soon!). Who I am as a researcher is hugely influenced by all these wonderful collaborations.

I want to specially thank three collaborators since I've spent thousands of hours with each on the whiteboard: Jon, Victor, and Enric. I thank Jon for his endless encouragement, for his fantastic humor, for sharing his brilliance, and for his empathy. Jon is a role model not just for how I hope to grow as a researcher, but also for how I hope to grow as a person. I thank Victor for his relentless positivity, for teaching me so many things, and for repeatedly hosting me in Paris during our collaborations. It is delightful to collaborate with someone who finds mathematics so beautiful. I thank Enric for somehow making everything fun and for all the great memories together these past few years, as collaborators, friends, and roommates. Collaborating with Enric over our home whiteboard was just about the only good thing about getting stuck at home during COVID lockdown—and what fun we had.

I am grateful to many wonderful MIT faculty members for interactions ranging from stimulating research conversations to amusing anecdotes, including Michel Goemans, Jonathan Kelner, Sasha Megretski, Ankur Moitra, Asu Özdağlar, Yury Polyanskiy, Sasha Rakhlin, Suvrit Sra, and Ashia Wilson. A big thank you to my incredibly inspiring undergraduate mentors from Princeton, especially Emmanuel Abbe, Amir Ali Ahmadi, Michael Damron, Elad Hazan, and Yoram Singer. (Special thanks to Amir Ali who quite literally changed my life by introducing me to Pablo and the field of optimization.) I would also like to thank Marco Cuturi, Mark Low, Gabriel Peyré, Jim Renegar, David Shmoys, and Joel Tropp for their unexpected kindness and encouragement. It means a lot to a young researcher.

A major reason I have enjoyed graduate school so much is the great community, including: Pablo's group, Philippe's group, Guy's group, Yury's group, Suvrit's group, my officemates (in each of the several offices I've been in), and the broader community here at the LIDS lab (which feels like home and will be hard to leave). Special thanks to James Siderius[1] and Denizcan Vanli who have been with me since the beginning here at MIT, to Kyle Dhillon, Nicholas Keeley, and Justin Vogel for many great memories, to Elizabeth Yang for the uplifting memes,

[1]It is fitting to devote at least a footnote to salute the illustrious Crossroads Pub, which in its glorious heyday offered $10 pizza and pitcher deals on Wednesdays. It went out of business shortly after James and I moved to Boston; the timing was not a coincidence.

to Kris Frey and Scarlett Barker for introducing me to the good folk at CABLE house, to Chris Bradley for sharing the fruits of his BBQ pitmastery, to Alex Katz and Pari Negi for the many blitz games, and to the MIT chess team, MIT tennis club, and MIT water polo IMs for letting me continue some of my favorite hobbies in such fun ways. Thanks also to so many more—as Fermat once said (slightly paraphrased), the margins here are too small to provide all the wonderful details.

I thank my family for raising me in a home with so much love (and good food). I am grateful to my grandparents for their endless support, to my parents for being with me every step of the way, and to my brilliant little brother for his wisdom far beyond his years. I look forward to disambiguating myself as Dr. Altschuler $\#n$, for some sufficiently large $n \geqslant 5$.

The work in this thesis was partially supported by a TwoSigma PhD Fellowship, NSF Graduate Fellowship 1122374, NSF grant 6933939, Berkeley grant 6934982, and Air Force grant 6943130.

I wish to dedicate this thesis to Matthew Brennan. Before Matt passed away, he was also a PhD student here at LIDS and in fact was the first person I met at MIT. We quickly bonded because we had so much common, and in the first year of grad school we made sure we took the same classes, from there becoming collaborators, officemates, and even roommates too. Matt was my best friend during the first few years of grad school. I miss him.

Chapter 1

Introduction

Broadly speaking, the research field of optimization provides a mathematical formalism for decision making. Expressed succinctly, the central question of this field is how to solve problems of the form

$$\min_{x \in \mathcal{X}} f(x). \tag{1.1}$$

Here, the variable x represents the decision to be made, the set $\mathcal{X}$ represents the constraints on the decision, and the function f represents the objective for which the decision should be optimized. The generality of such a formalism enables the recasting of problems from diverse fields in a unified framework—for example designing the most cost-efficient way to ship widgets from factories to stores, or training a machine learning classifier in a way that best fits the observed data.

Of course, a formalism like (1.1) is primarily useful if one can arrive at its solution. Typically a closed-form expression is impossible, and as such the goal is to design algorithms which compute solutions numerically. The goal of the field of optimization is the design and analysis of such algorithms. The most basic questions are: when can the optimization be performed efficiently and accurately? How efficiently? How accurately? And in what sense? In practice, in theory, in both?

The core of classical optimization focuses on understanding these questions in the setting where the decision variable x is an arbitrary vector, say in $\mathbb{R}^n$. However, modern applications throughout machine learning, applied mathematics, and engineering demand high-dimensional optimization problems where decision variables are probability distributions. Can such optimization problems be solved efficiently?

This thesis presents two interrelated lines of work in this direction through the common thread of Optimal Transport. A unifying theme is the optimization of (discrete but very high-dimensional) joint probability distributions with constrained marginals.

1.1 Outline

Here, we briefly summarize the main contributions of this thesis. See the individual chapters for a more detailed discussion of the results and related work.

This thesis is thematically divided into two parts: the first part considers Optimal Transport and other matrix-like problems; the second part considers Multimarginal Optimal Transport and other tensor-like problems. Despite their syntactic similarities, the problems in these two parts exhibit fundamentally different algorithmic phenomena and are thus treated separately.

1.1.1 Part I: Optimal Transport and matrix-like problems

Vignette: Optimal Transport. We begin by introducing Optimal Transport (OT) since it is a canonical example of the broader class of problems studied in Part I—not to mention a significant problem in its own right. OT was originally proposed by Monge in the 1700s when he asked: what is the least-effort way to move a pile of dirt into a nearby ditch of equal volume [155]? In the language of modern mathematics, OT is a distance between two probability distributions on a metric space, defined as the least effort required to move one distribution to the other. In recent years, OT has become central to diverse applications in machine learning, computer graphics, and the sciences because it provides a geometrically-meaningful measure of distance between complex data distributions beyond just piles of dirt—e.g., data distributions arising from point clouds, images, object meshes, document embeddings, or fMRI brain scans. See the monograph [178] for an overview of the many modern applications of OT.

A central challenge in modern OT applications is scalable computation. Indeed, the definition of OT implicitly demands the resolution of an optimization problem: how much mass P_{xy} should one send from each point x in one distribution μ to each point y in the other distribution ν? That is, solve

$$\min_{P \in \mathcal{M}(\mu,\nu)} \sum_{xy} P_{xy} C_{xy} \tag{1.2}$$

where C_{xy} denotes the cost of moving one unit of mass from point x to point y, and $\mathcal{M}(\mu,\nu)$ denotes the transportation polytope which consists of joint probability distributions P_{xy} that have marginals μ and ν. These marginal constraints encode the fact that μ is transported to ν: for each point x in distribution μ, the total mass sent from x is $\sum_y P_{xy} = \mu(x)$; and similarly, for each point y in distribution ν, the total mass sent to y is $\sum_x P_{xy} = \nu(y)$.

Data-driven applications require the computation of OT between discrete distributions (i.e., empirical data distributions), each with $\leqslant n$ data points, for n

large. In this setting, joint distributions P_{xy} with constrained marginals can be identified with $n \times n$ non-negative matrices with fixed row and column sums. The optimization problem (1.2) defining OT is thus a linear program with n^2 variables, n^2 non-negativity constraints, and $2n$ equality constraints. The challenge is that, of course, this is a very large linear program when the number of data points n is large. Solving such a linear program is challenging when n is in, say, the many thousands, let alone millions, and is it a longstanding challenge to develop algorithms that can compute OT in a reasonable amount of time for large values of n. See Chapter 2 for a discussion of the extensive literature on OT algorithms which dates back to Jacobi in the 19th century.

A more general story. Part I of this thesis is about a more general story, in which OT serves as one example. The unifying theme is optimization problems over joint probability distributions P_{xy} with constrained marginals. As mentioned above, OT is of this form—and in the discrete setting of interest, OT can be equivalently re-interpreted as an optimization problem over non-negative matrices with constrained row and column sums. This correspondence can be reversed: classical matrix problems can also be viewed under the lens of optimization over probability distributions. This includes classical periodic optimization problems such as Min-Mean-Cycle, and ubiquitous pre-conditioning problems such as Matrix Balancing and Matrix Scaling.

The relationship between these three problems and OT is summarized in Table 1.1. The relationship between OT and Matrix Scaling is classical and dates back to the 1960s [232]: briefly, the latter can be reformulated as an optimization problem that is identical to the one defining OT (1.2), modulo a certain entropic regularization term. This fact is well-known to the modern OT community as it forms the basis for what is by-now the standard algorithmic approach for computing OT at scale—namely using an algorithm for Matrix Scaling (the Sinkhorn algorithm) to compute approximate OT solutions. Much less well-known are the relationships between the other problems in Table 1.1. Briefly, Min-Mean-Cycle and Matrix Balancing—just like OT and Matrix Scaling—both have natural reformulations as optimization problems over non-negative matrices with constrained row and column sums (or equivalently, joint probability distributions P_{xy} with constrained marginals). The key difference is replacing the transportation polytope $\{P \in \mathbb{R}^{n\times n}_{\geqslant 0} : P1 = \mu, P^T1 = \nu\}$ by the circulation polytope $\{P \in \mathbb{R}^{n\times n}_{\geqslant 0} : P1 = P^T1\}$—i.e., replacing the constraint of fixed row and column sums with the constraint of symmetric row and column sums.

Scalable algorithms? Scalability is the key issue for all four of these optimization problems. Each has a long line of algorithmic work dating back more than half a century. However, at the beginning of the author's PhD, the state-of-the-

	Fixed marginals	Symmetric marginals
Linear program	**Optimal Transport**	**Min-Mean-Cycle**
Entropic regularization	**Matrix Scaling**	**Matrix Balancing**
Polytope	Transportation polytope	Circulation polytope
Simple algorithm	Sinkhorn algorithm	Osborne algorithm
Algorithm in dual	Block coordinate descent	Entrywise coordinate descent
Per-iteration progress	KL divergence	Hellinger divergence

Table 1.1: Although not traditionally viewed in this way, the four bolded optimization problems can be viewed under the same lens: each optimizes a probability distribution with similar constraints and objectives. Part I of this thesis fully exposes the unifying connections in this table and exploits them to obtain near-optimal runtimes for all four problems. These results are a combination of the papers [17, 18, 19].

art algorithms in practice either had no provable guarantees, or had extremely prohibitive $\tilde{O}(n^3)$ runtime[1] which is far too slow for modern applications.

Part I of this thesis establishes $\tilde{O}(n^2)$ runtimes for approximation algorithms for all four problems. These algorithms have nearly-optimal runtime: they solve in roughly the same amount of time as just reading the $n \times n$ matrix input. As such, these $\tilde{O}(n^2)$ runtimes are often called "near-linear" since they are linear in the input size n^2 up to lower-order logarithmic factors.

Moreover, we emphasize that these algorithms are not just theoretical benchmarks, but are actually effective in practice. Indeed, the runtime improvement from $\tilde{O}(n^3)$ to $\tilde{O}(n^2)$ enables solving Min-Mean-Cycle problems $\sim$10$\times$ larger in the same amount of time. And for the other three problems, our results establish near-optimal runtimes for $\sim$60-year-old algorithms that have long been the go-to algorithms of practitioners. For example, the (entropic) OT algorithm we study is the default in standard software packages (e.g., POT, OTT, OTJulia, and GeomLoss), and the Matrix Balancing and Scaling algorithms we study are used in standard numerical packages (e.g., MATLAB, R, Lapack, and Eispack) for pre-conditioning before ubiquitous computations like eigenvalue decomposition and matrix exponentiation. Our results provide theoretical justification for the practical efficacy of these widespread algorithms.

At a high level, our approach to all four problems has several key recurring themes, as outlined in Table 1.1. On the algorithmic side, solve the entropically regularized program rather than the linear program (followed possibly by a rounding post-processing step), because the entropic regularization allows us to algorithmically exploit the separability of the marginal constraints. And on the analysis side, interpret the resulting algorithm as a form of dual coordinate

[1]Throughout, the notation $\tilde{O}(\cdot)$ suppresses lower-order polylogarithmic factors for simplicity.

descent, observe that each iteration improves an appropriate dual potential by an amount that can be quantified by the current imbalance—either in Kullback-Leibler or Hellinger divergence—and use probabilistic inequalities to convert this per-iteration improvement to a final runtime analysis in a dimension-free way, which is crucial for near-linear runtimes. However, instantiating this high-level flow differs between the individual problems in several important ways, for example how to quantify the per-iteration progress, how to deal with the fact that fixed marginal constraints are completely separable whereas symmetric marginal constraints are not, and how to round to a feasible solution after optimization. Part I explains the algorithms and analyses for these four problems.

We now briefly outline the individual contributions of the chapters in part I.

Chapter 2. This chapter revisits the problem of Optimal Transport, which as mentioned above is central to modern applications spanning computer science, data science, and the natural sciences. However, despite the recent introduction of several algorithms with good empirical performance, it was unknown whether general OT distances can be approximated in near-linear time.

We demonstrate that, with an appropriate choice of parameters, the Sinkhorn algorithm—originally designed for Matrix Scaling, and with a rich history dating back nearly a century (for a historical perspective see the survey [118])—is in fact a near-linear time approximation algorithm for computing OT distances between discrete measures. This is the first proof that such near-linear time results are achievable for OT. Core to our work is a new analysis of the classical Sinkhorn algorithm which we show converges in a number of iterations independent of the dimension n of the matrix to scale.

Chapter 3. This chapter revisits the problem of Matrix Balancing, a pre-conditioning primitive 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 and in fact polynomial runtimes were not known for more than half a century. We show that a simple random variant of Osborne's algorithm converges in the ℓ_1 norm in near-linear time in the input sparsity. 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.

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. Our

analysis technique is robust in that it also enables us to establish significantly improved runtime bounds for other variants of Osborne's algorithm, establish polynomial bit complexity, and establish improved runtimes if an associated graph is moderately connected.

Chapter 4. This chapter revisits Min-Mean-Cycle, a classical problem in graph theory with extensive applications in periodic optimization, algorithm design, control theory, and max-plus algebra. We propose an approximation algorithm that, for graphs with polylogarithmic diameter, achieves a near-linear runtime. In particular, this is the first algorithm whose runtime scales in the number of vertices n as $\tilde{O}(n^2)$ for the complete graph. Moreover, unconditionally on the diameter, the algorithm only uses $O(n)$ memory beyond reading the input, making it "memory-optimal." Our approach is based on solving a linear programming relaxation using entropic regularization, which modulo a post-processing rounding step, reduces the problem to Matrix Balancing—á la the classical reduction of OT to Matrix Scaling.

The proposed algorithm is practical and simple to implement. Preliminary numerical simulations demonstate that in practice, the proposed algorithm can compute high-quality solutions in essentially linear runtime and for $\sim 10\times$ larger problem sizes than the state-of-the-art algorithms implemented in the popular, heavily-optimized C++ software package LEMON.

■ 1.1.2 Part II: Multimarginal Optimal Transport and tensor-like problems

From two marginals to many marginals. The optimization problems described above involve joint probability distributions with two marginals. Part II of the thesis tackles problems with many marginals k. These optimization problems—often called *Multimarginal Optimal Transport (MOT)* in the setting of linear optimization and constrained marginals—are central to diverse applications. Examples range from averaging k data distributions in data science, to performing inference from k measurements in machine learning, to simulating incompressible fluids over k timesteps in fluid dynamics, to analyzing stable systems of k electrons in quantum chemistry.

A central challenge in all of these problems is that in general, it is intractable to even store a k-variate distribution, let alone to solve for the optimal one. Indeed, a k-variate joint distribution in which each variable takes n possible values is in correspondence with a k-order tensor that has n^k entries. This complexity n^k grows exponentially, and is intractable beyond tiny input sizes like $n = k = 10$. Here, nearly-linear runtime in n and k seem too much to hope for—but can we at least lower the complexity from exponential to polynomial?

Remarkably, specially tailored algorithms with polynomial runtime were known

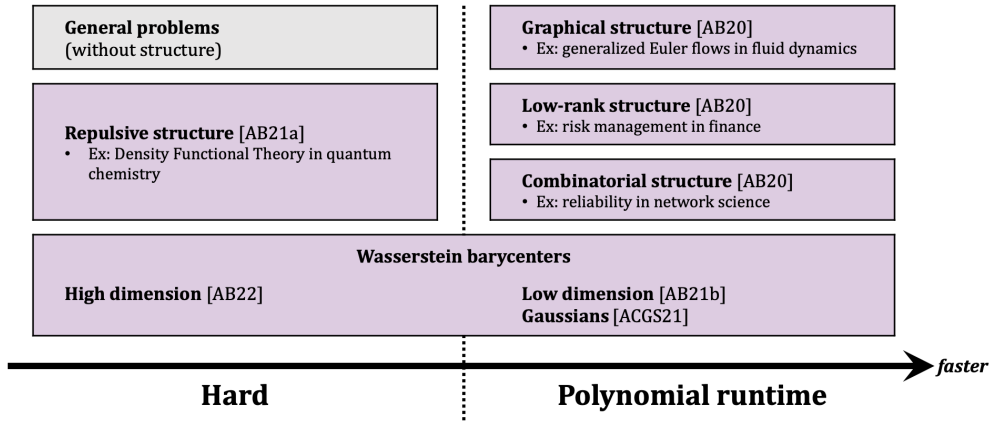


Figure 1.1: Overview of Part II of this thesis: elucidating the boundary between tractability and intractability for Multimarginal Optimal Transport and other optimization problems over probability distributions with many marginals (equivalently: tensors). Here, the fundamental goal to achieve is polynomial runtime. These results are a combination of the papers [11, 12, 13, 14].

for several MOT applications, see Chapter 6 for details. But these algorithms do not extend to the many other MOT applications. Can those be solved efficiently too? More generally:

What optimization problems over joint probability distributions are solvable in polynomial time in the number of marginals?

In nearly all applications of MOT, the input has a concise implicit representation of size that is polynomial in k rather than exponential. In such setups, is exponential runtime avoidable? What "structure" of an MOT problem enables solving it in polynomial time? How do you check if that structure is present? What candidate algorithms might exploit that structure? Do some algorithms require strictly more structure than other algorithms?

Part II of this thesis attack these foundational questions from both fronts—by designing polynomial-time algorithms when possible, or by proving rigorous hardness results otherwise. We outline these results below; see Figure 1.1 for a summary.

Chapter 5. In this chapter, we investigate the fundamental limitations of the rapidly growing line of research that seeks polynomial-time algorithms for structured MOT problems. Specifically, we establish the intractability of a number of MOT problems studied in the literature that have resisted previous algorithmic efforts. For instance, we demonstrate that in the absence of further structure,

MOT is intractable even if the objective function decomposes into pairwise interactions between the variables, or if it decomposes in a low-rank manner with super-constant rank. As another example, we provide evidence that repulsive costs make MOT intractable by showing that several such problems of interest are NP-hard to solve, including, notably, the popular MOT formulation of Density Functional Theory when using the Coulomb-Buckingham potential. See Figure 1.1, left.

All of our hardness results hold even for approximate computation—and to the best of our knowledge, these are the first inapproximability results for any MOT problems. Together, these intractability results help guide the search for efficient MOT algorithms since they establish that these commonly occuring structural properties of MOT problems are, by themselves, insufficient for developing efficient algorithms.

Chapter 6. This chapter complements the intractability results in the previous chapter by developing a unified algorithmic framework for MOT. This framework builds upon classical ideas from oracle-based optimization in order to characterize the "structure" that an MOT problem must possess in order to be solvable in polynomial time. This framework has several benefits. First, it enables us to show that the Sinkhorn algorithm, which is currently the most popular MOT algorithm, requires strictly more structure than other algorithms do to solve MOT in polynomial time. Second, our framework makes it much simpler to develop polynomial time algorithms for a given MOT problem. In particular, it is necessary and sufficient to (approximately) solve the dual feasibility oracle—which is much more amenable to standard algorithmic techniques.

We illustrate this ease-of-use by developing polynomial-time algorithms for three general classes of MOT cost structures: (1) graphical structure; (2) set-optimization structure; and (3) low-rank plus sparse structure. For structure (1), we recover the known result that Sinkhorn has polynomial runtime; moreover, we provide the first polynomial-time algorithms for computing solutions that are exact and sparse. For structures (2)-(3), we give the first polynomial-time algorithms, even for approximate computation. Together, these three structures encompass many—if not most—current applications of MOT. We demonstrate this general theory on applications in diverse fields ranging from operations research to data science to fluid dynamics. See Figure 1.1, right. Two particularly notable applications are the first polynomial-time algorithms that compute high-precision solutions for the 30-year-old problem of generalized Euler flows and for the popular geometric problem of computing low-dimensional Wasserstein barycenters (detailed in Chapter 7).

Chapter 7. This chapter highlights the application of our general MOT framework to one particularly popular instance of MOT: the computation of Wasserstein barycenters (a.k.a., Optimal Transport barycenters). Wasserstein barycenters provide a geometrically-meaningful notion of average between probability distributions, and over the past decade have emerged as a central tool in data science for manipulating and interpreting complicated geometric data. However, despite considerable attention, the most fundamental question about the complexity of Wasserstein barycenters remained open: can they be computed in polynomial time?

In this chapter, we resolve this problem and uncover that the answer depends subtly on the dimension due to the continuous nature of the problem. Specifically, we establish that Wasserstein barycenters are NP-hard to compute in high dimension, but can be computed in polynomial time in any fixed dimension (e.g., as in physical, imaging, or graphics applications). This uncovers a "curse of dimensionality" for Wasserstein barycenter computation which does not occur for Optimal Transport computation.

■ 1.2 Related publications

The contents of this thesis are based on the following published papers.[2]

Chapter 2. J.M. Altschuler, J. Weed, and P. Rigollet. Near-linear time approximation algorithms for Optimal Transport via Sinkhorn iteration. *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. [19]

Chapter 3. J.M. Altschuler and P.A. Parrilo. Approximating Min-Mean-Cycle for low-diameter graphs in near-optimal time and memory. *SIAM Journal on Optimization (SIOPT)*, 2022. [18]

Chapter 4. J.M. Altschuler and P.A. Parrilo. Near-linear convergence of the Random Osborne algorithm for Matrix Balancing. *Mathematical Programming (MAPR)*, 2022. [18]

Chapter 5. J.M. Altschuler and E. Boix-Adserá. Hardness results for Multimarginal Optimal Transport problems. *Dicrete Optimization (DISOPT)*, 2021. [12]

Chapter 6. J.M. Altschuler and E. Boix-Adserá. Polynomial-time algorithms for Multimarginal Optimal Transport problems with structure. *Mathematical Programming (MAPR)*, 2022. [13]

[2] Also, a unified exposition of the results in Part I will appear in an invited paper to SIAG/OPT Views and News.

Chapter 7.

- J.M. Altschuler and E. Boix-Adserá. Wasserstein barycenters are NP-hard to compute. *SIAM Journal on Mathematics of Data Science (SIMODS)*, 2022. [11]

- J.M. Altschuler and E. Boix-Adserá. Wasserstein barycenters can be computed in polynomial time in fixed dimension. *Journal of Machine Learning Research (JMLR)*, 2021. [14]

Three remarks about how we have organized the thesis for the purpose of readability. First, we have omitted several details and auxiliary results from the above papers in the interest of brevity. Second, some parts of the above papers are moved to different chapters in order to avoid redundancy. Third, the notation is largely consistent between chapters, and for the convenience of the reader, each chapter introduces the notation relevant to that chapter.

Part I

Entropic OT and Matrix-Like Problems

Chapter 2

Near-linear time approximation algorithms for Optimal Transport via Sinkhorn iteration

Computing Optimal Transport distances such as the earth mover's distance is a fundamental problem in machine learning, statistics, and computer vision. Despite the recent introduction of several algorithms with good empirical performance, it is unknown whether general Optimal Transport distances can be approximated in near-linear time. This chapter demonstrates that this ambitious goal is in fact achieved by the Sinkhorn algorithm—a classical algorithm that was recently popularized by the influential 2013 paper of Cuturi—and provides guidance towards parameter tuning for this algorithm. This result relies on a new analysis of Sinkhorn iterations that also directly suggests a new algorithm `GREENKHORN` with the same theoretical guarantees. Numerical simulations illustrate that in terms of the number of arithmetic operations, `GREENKHORN` significantly outperforms the classical `SINKHORN` algorithm in practice.

2.1 Introduction

Computing distances between probability measures on metric spaces, or more generally between point clouds, plays an increasingly preponderant role in machine learning [24, 122, 146, 157, 192], statistics [38, 88, 171, 214] and computer vision [44, 189, 206]. A prominent example of such distances is the *earth mover's distance* introduced in [230] (see also [189]), which is a special case of Wasserstein distance, or Optimal Transport (OT) distance [225].

While OT distances exhibit a unique ability to capture geometric features of the objects at hand, they suffer from a heavy computational cost that has been prohibitive in large scale machine-learning applications until the recent re[1]-

[1]This algorithm and its core entropic regularization ideas have a long history in the trans-

popularization of *Sinkhorn Distances* by Cuturi [73]. Combined with other numerical tricks, these recent advances have enabled the treatment of large clouds of points in computer graphics such as triangle meshes [206] and high-resolution neuroimaging data [99]. Sinkhorn Distances mainly rely on the idea of *entropy penalization*, which has been implemented in similar problems at least since Schrödinger [142, 197]. This powerful idea has been successfully applied to a variety of contexts not only as a statistical tool for model selection [123, 186, 187] and online learning [56], but also as an optimization gadget in first-order optimization methods such as mirror descent and proximal methods [51].

Related work. Computing an OT distance amounts to solving the following linear system:

$$\min_{P \in \mathcal{U}_{r,c}} \langle P, C \rangle \,, \tag{2.1}$$

where

$$\mathcal{U}_{r,c} := \left\{ P \in \mathbb{R}_+^{n \times n} \,:\, P\mathbf{1} = r \,, P^\top \mathbf{1} = c \right\},$$

is the *transport polytope*, $\mathbf{1}$ is the all-ones vector in $\mathbb{R}^n$, $C \in \mathbb{R}_+^{n \times n}$ is a given *cost matrix*, and $r \in \mathbb{R}^n, c \in \mathbb{R}^n$ are given vectors with positive entries that sum to one. Typically C is a matrix containing pairwise distances, but in this chapter we allow C to be any positive dense matrix with bounded entries since our results are more general. For brevity, this chapter focuses on square matrices C and P, since extensions to the rectangular case are straightforward.

This chapter is at the intersection of two lines of research: a practical one that pursues fast algorithms for Optimal Transport problems and a theoretical one that aims at finding (near) linear time approximation algorithms for simple problems that are already known to run in polynomial time.

Noticing that (2.1) is a linear program with $O(n)$ linear constraints and certain graphical structure, one can use the recent Lee-Sidford linear solver to find a solution in time $\widetilde{O}(n^{2.5})$ [140], improving over the previous standard of $O(n^{3.5})$ [185]. While no practical implementation of the Lee-Sidford algorithm is known, it provides a theoretical benchmark for our methods. Their result is part of a long line of work initiated by the seminal paper of Spielman and Teng [207] on solving linear systems of equations, that has provided a building block for near-linear time approximation algorithms in a variety of combinatorially structured linear problems. Our work fits into this line of work in the sense that it provides the first near-linear time guarantee to approximate (2.1). However, our work presents a striking difference: we analyze algorithms that are also practically efficient.

portation community dating back more than half a century [232]; for a historical perspective, see, e.g., [118, 195] and the references within.

Practical algorithms for computing OT distances include Orlin's algorithm for the *Uncapacitated Minimum Cost Flow* problem via a standard reduction. Akin to interior point methods, it has a provable complexity of $O(n^3 \log n)$. This cubic dependence on the dimension is also observed in practice, thereby preventing large-scale applications. To overcome the limitations of such general solvers, various ideas ranging from graph sparsification [177] to metric embedding [100, 119, 200] have been proposed over the years to deal with particular cases of OT distance. At the same time, recent years have witnessed the development of scalable methods for general OT that leverage the idea of entropic regularization [32, 73, 93]. However, their apparent practical efficacy still lacks theoretical guarantees. In particular, the existence of algorithms to compute or approximate general OT distances in time nearly linear in the input size n^2 is an open question. Therefore, new tools are needed to develop provably near-linear time algorithms for OT distance computation.

Our contribution. The contribution of this chapter is twofold. First we demonstrate that, with an appropriate choice of parameters, the classical Sinkhorn algorithm is in fact a *near-linear time* approximation algorithm for computing OT distances between discrete measures. This is the first proof that such near-linear time results are achievable for Optimal Transport. Core to our work is a new analysis of the Sinkhorn iteration algorithm, which we show converges in a number of iterations independent of the dimension n of the matrix to balance using a new and arguably more natural analysis of these iterations. In particular, this analysis directly suggests a greedy variant of Sinkhorn iterations that also provably runs in near-linear time and, while less parallelizable, significantly outperforms the classical algorithm in practice in terms of arithmetic operations. Finally, while most approximation algorithms output an approximation of the optimum *value* of the linear program (2.1), we also describe a simple rounding algorithm that provably outputs a feasible solution to (2.1). Specifically, for any $\varepsilon > 0$ and bounded, positive cost matrix C, we describe an algorithm that runs in time $\widetilde{O}(n^2/\varepsilon^4)$ and outputs $\hat{P} \in \mathcal{U}_{r,c}$ such that

$$\langle \hat{P}, C \rangle \leq \min_{P \in \mathcal{U}_{r,c}} \langle P, C \rangle + \varepsilon$$

Notation. We denote non-negative real numbers by $\mathbb{R}_+$, the set of integers $\{1, \ldots, n\}$ by $[n]$, and the d-dimensional simplex by $\Delta_d := \{x \in \mathbb{R}_+^d : \sum_{i=1}^d x_i = 1\}$. For two probability distributions $p, q \in \Delta_d$ such that p is absolutely continuous w.r.t. q, we define the entropy $H(p)$ of p and the Kullback-Leibler divergence

$D_{\mathsf{KL}}(p\|q)$ between p and q respectively by

$$H(p) = \sum_{i=1}^{d} p_i \log \frac{1}{p_i}, \qquad D_{\mathsf{KL}}(p\|q) := \sum_{i=1}^{d} p_i \log\left(\frac{p_i}{q_i}\right).$$

We use $\mathbf{1}$ to denote the all-ones vector in $\mathbb{R}^n$. For a matrix $A = (A_{ij})$, we denote by $\exp(A)$ the matrix with entries $(e^{A_{ij}})$. For $A \in \mathbb{R}^{n\times n}$, we denote its row and columns sums by $r(A) := A\mathbf{1} \in \mathbb{R}^n$ and $c(A) := A^\top \mathbf{1} \in \mathbb{R}^n$, respectively. We write $\|A\|_{\max} = \max_{ij} |A_{ij}|$. For two matrices of the same dimensions, we denote the Frobenius inner product of A and B by $\langle A, B\rangle = \sum_{ij} A_{ij}B_{ij}$. For a vector $x \in \mathbb{R}^n$, we write $\mathrm{diag}(x) \in \mathbb{R}^{n\times n}$ to denote the diagonal matrix with entries $(\mathrm{diag}(x))_{ii} = x_i$.

For any two nonnegative sequences $(u_n)_n, (v_n)_n$, we write $u_n = \widetilde{O}(v_n)$ if there exist positive constants C, c such that $u_n \le Cv_n(\log n)^c$. For any two real numbers, we write $a \wedge b = \min(a, b)$.

■ 2.2 Optimal Transport in near-linear time

In this section, we describe the main algorithm studied in this chapter. Pseudocode appears in Algorithm 2.1.

$\eta \leftarrow \frac{4\log n}{\varepsilon}$, $\varepsilon' \leftarrow \frac{\varepsilon}{4\|C\|_{\max}}$
\\ Step 1: Approximately project onto $\mathcal{U}_{r,c}$
1: $A \leftarrow \exp(-\eta C)$
2: $B \leftarrow \texttt{PROJ}(A, \mathcal{U}_{r,c}, \varepsilon')$

\\ Step 2: Round to feasible point in $\mathcal{U}_{r,c}$
3: Output $\hat{P} \leftarrow \texttt{ROUND}(B, \mathcal{U}_{r,c})$

Algorithm 2.1: APPROXOT(C, r, c, ε)

The core of our algorithm is the computation of an *approximate Sinkhorn projection* of the entrywise-exponentiated matrix $A = \exp(-\eta C)$ (Step 1). We discuss this step and its connection to entropic penalization in §2.2.1. Since our approximate Sinkhorn projection is not guaranteed to lie in the feasible set, we round our approximation to ensure that it lies in $\mathcal{U}_{r,c}$ (Step 2). More details about the rounding procedure appear in §2.2.2.

Our main theorem about Algorithm 2.1 is the following accuracy and runtime guarantee.

Theorem 2.2.1. *Algorithm 2.1 returns a point $\hat{P} \in \mathcal{U}_{r,c}$ satisfying*

$$\langle \hat{P}, C \rangle \leqslant \min_{P \in \mathcal{U}_{r,c}} \langle P, C \rangle + \varepsilon$$

in $O(n^2 + S)$ operations, where S is the number of operations of the subroutine $\texttt{PROJ}(A, \mathcal{U}_{r,c}, \varepsilon')$. In particular, if $\|C\|_{\max} \leqslant L$, then S can be $O(n^2 L^3 (\log n)\, \varepsilon^{-3})$, so that Algorithm 2.1 requires $O(n^2 L^3 (\log n)\, \varepsilon^{-3})$ operations.

For simplicity, we state Theorem 2.2.1 in terms of elementary arithmetic operations, and do not consider bit complexity issues arising from the taking of exponentials in Step 1. It can be easily shown [127] that the maximum bit complexity throughout the execution of our algorithm is $O(L(\log n)/\varepsilon)$. As a result, factoring in bit complexity leads to a runtime of $O(n^2 L^4 (\log n)^2\, \varepsilon^{-4})$, which is truly near-linear.[2]

■ 2.2.1 Approximate Sinkhorn projection

The core of our algorithm is the entropic penalty

$$P^\eta := \operatorname*{arg\,min}_{P \in \mathcal{U}_{r,c}} \left\{ \langle P, C \rangle - \eta^{-1} H(P) \right\}, \tag{2.2}$$

where H is the entrywise entropy. Using this entropic penalty for transportation dates back at least to [232]. The key benefit of this entropic penalty is that the solution to (2.2) can be characterized explicitly by analyzing its first-order conditions for optimality (see, e.g., [73] for a proof).

Theorem 2.2.2. *For any cost matrix C and $r, c \in \Delta_n$, the minimization progam (2.2) has a unique minimum at $P^\eta \in \mathcal{U}_{r,c}$ of the form $P^\eta = XAY$, where $A = \exp(-\eta C)$ and $X, Y \in \mathbb{R}_+^{n \times n}$ are both diagonal matrices. The matrices (X, Y) are unique up to a constant factor.*

We call the matrix P^η appearing in Theorem 2.2.2 the *Sinkhorn projection* of A, denoted $\Pi_{\mathcal{S}}(A, \mathcal{U}_{r,c})$, after Sinkhorn, who proved uniqueness in [203]. Computing $\Pi_{\mathcal{S}}(A, \mathcal{U}_{r,c})$ exactly is impractical, so we implement instead an approximate version $\texttt{PROJ}(A, \mathcal{U}_{r,c}, \varepsilon')$ that outputs a matrix $B = XAY$ which may not lie in $\mathcal{U}_{r,c}$ but satisfies the condition $\|r(B) - r\|_1 + \|c(B) - c\|_1 \leqslant \varepsilon'$. We stress that this condition is very natural from a statistical standpoint, since it requires that $r(B)$ and $c(B)$ are close to the target marginals r and c in *total variation distance*. Prior work on approximate Sinkhorn projection focuses on the weaker condition

[2] We remark that with a more careful analysis, this bit complexity can be reduced from $O(L(\log n)/\varepsilon)$ to logarithmic size $O(\log(Ln/\varepsilon))$, see §3.10.4.

$\|r(B) - r\|_2 + \|c(B) - c\|_2 \leqslant \varepsilon'$. Not only do such bounds lack statistical meaning, but they also fail to yield useful approximation guarantees for OT distances. We discuss this issue and give an algorithm to compute $\texttt{PROJ}(A, \mathcal{U}_{r,c}, \varepsilon')$ in §2.3.

Theorem 2.2.3. *Let $\|C\|_{\max} \leqslant L$. There exists an implementation of the procedure $\texttt{PROJ}(A, \mathcal{U}_{r,c}, \varepsilon')$ requiring $O(n^2 L^3 (\log n)\, \varepsilon^{-3})$ elementary arithmetic operations that outputs a matrix $B = XAY$ where $X, Y \in \mathbb{R}_+^{n \times n}$ are diagonal matrices and*

$$\|r(B) - r\|_1 + \|c(B) - c\|_1 \leqslant \varepsilon' \ .$$

Proof. Theorems 2.3.1 and 2.3.5 imply that both the SINKHORN and GREENKHORN algorithms yield a matrix B of the desired accuracy in $O(n^2 (\varepsilon')^{-2} \log \frac{s}{\ell})$ elementary arithmetic operations, where s is the sum of the entries of A and ℓ is the smallest entry of A. Since the matrix C is nonnegative, $s \leqslant n^2$. The smallest entry of A is $e^{-\eta \|C\|_{\max}}$, so $\log 1/\ell = \eta \|C\|_{\max}$. We obtain

$$S = O(n^2 (\varepsilon')^{-2} (\log n + \eta \|C\|_{\max}) \eta \|C\|_{\max}) \,,$$

and plugging in the values of η and ε' finishes the proof. □

■ 2.2.2 Rounding to a feasible point

The rounding procedure we implement is very simple, and is based on the observation that calculating the Optimal Transport with respect to the total variation distance is computationally cheap.

1: $X \leftarrow$ diagonal with $X_{ii} = \frac{r_i}{r_i(F)} \wedge 1$
2: $F \leftarrow XF$
3: $Y \leftarrow$ diagonal with $Y_{jj} = \frac{c_j}{c_j(F)} \wedge 1$
4: $F \leftarrow FY$
5: $\mathrm{err}_r \leftarrow r - r(F)$, $\mathrm{err}_c \leftarrow c - c(F)$
6: Output $G \leftarrow F + \mathrm{err}_r \mathrm{err}_c^\top / \|\mathrm{err}_r\|_1$

Algorithm 2.2: $\texttt{ROUND}(F, \mathcal{U}_{r,c})$

Theorem 2.2.4. *If $r, c \in \Delta_n$ and $F \in \mathbb{R}_+^{n \times n}$, then there exists $G \in \mathcal{U}_{r,c}$ satisfying*

$$\|G - F\|_1 \leqslant \|r(F) - r\|_1 + \|c(F) - c\|_1 \ .$$

Such a G can be computed in $O(n^2)$ time by Algorithm 2.2.

A proof of Theorem 2.2.4 is deferred to §2.5.

2.2.3 Proof of Theorem 2.2.1

We have already established that Steps 1 and 2 run in S and $O(n^2)$ time, respectively, so the runtime guarantee is immediate.

Let B be the output of $\texttt{PROJ}(A, \mathcal{U}_{r,c}, \varepsilon')$, and let $P^* \in \arg\min_{P \in \mathcal{U}_{r,c}} \langle P, C \rangle$ be an optimal solution to the original OT program.

We first show that $\langle B, C \rangle$ is not much larger than $\langle P^*, C \rangle$. To that end, write $r' := r(B)$ and $c' := c(B)$. Since $B = XAY$ for positive diagonal matrices X and Y, Theorem 2.2.2 implies B is the optimal solution to

$$\min_{P \in \mathcal{U}_{r',c'}} \langle P, C \rangle - \eta^{-1} H(P) \,. \tag{2.3}$$

By Theorem 2.2.4, there exists a matrix $P' \in \mathcal{U}_{r',c'}$ such that

$$\|P' - P^*\|_1 \leqslant \|r' - r\|_1 + \|c' - c\|_1 \,.$$

Moreover, since B is an optimal solution of (2.3), we have

$$\langle B, C \rangle - \eta^{-1} H(B) \leqslant \langle P', C \rangle - \eta^{-1} H(P') \,.$$

Thus, by Hölder's inequality

$$\begin{aligned} \langle B, C \rangle - \langle P^*, C \rangle &= \langle B, C \rangle - \langle P', C \rangle + \langle P', C \rangle - \langle P^*, C \rangle \\ &\leqslant \eta^{-1}(H(B) - H(P')) + (\|r' - r\|_1 + \|c' - c\|_1)\|C\|_{\max} \\ &\leqslant 2\eta^{-1} \log n + (\|r' - r\|_1 + \|c' - c\|_1)\|C\|_{\max} \,, \end{aligned} \tag{2.4}$$

where we have used the fact that $0 \leq H(B), H(P') \leq 2 \log n$.

Theorem 2.2.4 implies that the output $\hat{P}$ of $\texttt{ROUND}(B, \mathcal{U}_{r,c}, \varepsilon')$ satisfies

$$\|B - \hat{P}\|_1 \leqslant \|r' - r\|_1 + \|c' - c\|_1 \,.$$

Together with (2.4) and Hölder's inequality, it yields

$$\langle \hat{P}, C \rangle \leqslant \min_{P \in \mathcal{U}_{r,c}} \langle P, C \rangle + 2\eta^{-1} \log n + 2(\|r' - r\|_1 + \|c' - c\|_1)\|C\|_{\max} \,.$$

Applying the guarantee of $\texttt{PROJ}(A, \mathcal{U}_{r,c})$ yields

$$\langle \hat{P}, C \rangle \leqslant \min_{P \in \mathcal{U}_{r,c}} \langle P, C \rangle + \frac{2 \log n}{\eta} + 2\,\varepsilon'\,\|C\|_{\max} \,.$$

Plugging in the values of η and ε' prescribed in Algorithm 2.1 yields the claim.

2.3 Linear-time approximate Sinkhorn projection

Given a matrix A, Sinkhorn proposed a simple iterative algorithm to approximate the Sinkhorn projection $\Pi_{\mathcal{S}}(A, \mathcal{U}_{r,c})$, which is commonly known as the Sinkhorn-Knopp algorithm or RAS method. Despite the simplicity of this algorithm and its good performance in practice, it has been difficult to analyze. As a result, recent work showing that $\Pi_{\mathcal{S}}(A, \mathcal{U}_{r,c})$ can be approximated in near-linear time [9, 67] has bypassed the Sinkhorn-Knopp algorithm entirely. Though these results come with provable convergence guarantees, the algorithms they propose are not currently implementable in practice. In our work, we obtain a new analysis of the simple and practical Sinkhorn-Knopp algorithm, showing that it also approximates $\Pi_{\mathcal{S}}(A, \mathcal{U}_{r,c})$ in near-linear time.

Pseudocode for the Sinkhorn-Knopp algorithm appears in Algorithm 2.3. In brief, it is an alternating projection procedure which renormalizes the rows and columns of A in turn so that they match the desired row and column marginals r and c. At each step, it prescribes to either modify all the rows by multiplying row i by $r_i / r_i(A)$ for $i \in [n]$, or to do the analogous operation on the columns. (We interpret the quantity $0/0$ as 1 in this algorithm if ever it occurs.)

Initialize $k \leftarrow 0$
$A^{(0)} \leftarrow A / \|A\|_1$, $X^{(0)} \leftarrow I$, $Y^{(0)} \leftarrow I$
while $\mathrm{dist}(A^{(k)}, \mathcal{U}_{r,c}) > \varepsilon$ **do**
    $k \leftarrow k + 1$
    if k odd **then**
        $X \leftarrow$ diagonal with $X_{ii} = \frac{r_i}{r_i(A^{(k-1)})}$
        $X^{(k)} \leftarrow X^{(k-1)} X$, $Y^{(k)} \leftarrow Y^{(k-1)}$
    else
        $Y \leftarrow$ diagonal with $Y_{jj} = \frac{c_j}{c_j(A^{(k-1)})}$
        $Y^{(k)} \leftarrow Y^{(k-1)} Y$, $X^{(k)} \leftarrow X^{(k-1)}$
    $A^{(k)} = X^{(k)} A Y^{(k)}$
Output $B \leftarrow A^{(k)}$

Algorithm 2.3: $\texttt{SINKHORN}(A, \mathcal{U}_{r,c}, \varepsilon')$

It is clear that, if Algorithm 2.3 terminates, then its output B satisfies $\mathrm{dist}_{\mathcal{M}}(B, \mathcal{U}_{r,c}) \leqslant \varepsilon'$. (The choice of metric in which to measure $\mathrm{dist}_{\mathcal{M}}(A, \mathcal{U}_{r,c})$ will be discussed further below.) If m is the number of nonzero entries in A, then each iteration of Sinkhorn can be performed in $O(m)$ time. Therefore the total running time of Algorithm 2.3 is linear in m so long as the number of iterations depends only on ε' but not on n or m.

Before this work, the best analysis of the RAS method appeared in [127]. They defined

$$\operatorname{dist}_{\mathcal{M}}(A, \mathcal{U}_{r,c}) = \|r(A) - r\|_2 + \|c(A) - c\|_2 \,,$$

and showed that if A is strictly positive with $\min_{ij} A_{ij} = \ell$ and $\sum_{ij} A_{ij} = s$, then Algorithm 2.3 outputs a matrix B satisfying

$$\|r(B) - r\|_2 + \|c(B) - c\|_2 \leqslant \varepsilon' \tag{2.5}$$

in $O\big(\rho(\varepsilon')^{-2} \log(s/\ell)\big)$ iterations, where $\rho > 0$ is such that $r_i, c_i \leq \rho$ for all $i \in [n]$. While this result appears to show that we can obtain an ε'-approximate scaling of A in only $\widetilde{O}((\varepsilon')^{-2})$ iterations, this impression is misleading. Indeed, the ℓ_2 norm is not an appropriate measure of closeness between probability vectors, since very different distributions on large alphabets can nevertheless have small ℓ_2 distance: for example, $(n^{-1}, \dots, n^{-1}, 0, \dots, 0)$ and $(0, \dots, 0, n^{-1}, \dots, n^{-1})$ in Δ_{2n} have ℓ_2 distance $\sqrt{2/n}$ even though they have disjoint support. As noted above, for statistical problems, including computation of the OT distance, it is more natural to measure distance in ℓ_1 norm.

The best ℓ_1 guarantee available from previous work implies that a matrix B can be obtained satisfying

$$\|r(B) - r\|_1 + \|c(B) - c\|_1 \leqslant \varepsilon'$$

in $O\big(n\rho(\varepsilon')^{-2} \log(s/\ell)\big)$ iterations, where the extra factor of n is the price to pay to convert an ℓ_2 bound to an ℓ_1 bound. Note that $\rho \geqslant 1/n$, so $n\rho$ is always larger than 1. In the extreme where r or c contains an entry of constant size, $n\rho = \Omega(n)$. However, if $r = c = \mathbf{1}_n/n$ are uniform distributions, then $n\rho = 1$ and no dependence on the dimension appears. Our new analysis allows to a dimension-independent bound on the number of iterations beyond the uniform case.

Theorem 2.3.1. *Algorithm 2.3 with* $\operatorname{dist}_{\mathcal{M}}(A, \mathcal{U}_{r,c}) = \|r(A) - r\|_1 + \|c(A) - c\|_1$ *outputs a matrix* B *satisfying* $\operatorname{dist}_{\mathcal{M}}(B, \mathcal{U}_{r,c}) \leqslant \varepsilon'$ *in* $O\big((\varepsilon')^{-2} \log(s/\ell)\big)$ *iterations.*

Comparing our result with the bound on ℓ_2 distance, we see that our bound is always stronger, by up to a factor of n. Moreover, our analysis is extremely short. Our improved results and simplified proof follow directly from the fact that we carry out the analysis entirely with respect to the Kullback–Leibler divergence, a common measure of statistical distance. This measure possesses a close connection to the total-variation distance via Pinsker's inequality (Lemma 2.3.4, below), from which we obtain the desired ℓ_1 bound. A full proof appears in §2.3.1.

We also propose a new algorithm, `GREENKHORN` (for "Greedy Sinkhorn"), which enjoys precisely the same bound as `SINKHORN`, but which works better in many

practical situations (see §2.4 for experimental results). We emphasize that previous analyses of Sinkhorn iteration did not apply to `GREENKHORN`, but our new analysis handles the `GREENKHORN` algorithm with only trivial modifications.

■ 2.3.1 New analysis of Sinkhorn iteration

We analyze Sinkhorn iterations by considering the following auxiliary function, which has appeared in much of the literature on Sinkhorn projections [67, 124, 125, 127]. Given a matrix A and desired row and column sums r and c, we define $f : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ by

$$f(x, y) = \sum_{ij} A_{ij} e^{x_i + y_j} - \langle r, x \rangle - \langle c, y \rangle .$$

It is easy to check that a minimizer (x^*, y^*) of f yields the Sinkhorn projection of A: writing $X = \operatorname{diag}(\exp(x^*))$ and $Y = \operatorname{diag}(\exp(y^*))$, first order optimality conditions imply that XAY lies in $\mathcal{U}_{r,c}$, and therefore $XAY = \Pi_{\mathcal{S}}(A, \mathcal{U}_{r,c})$.

Since the first step of Algorithm 2.3 renormalizes A to have total mass 1, we can assume in our analysis that $s = 1$ at the price of replacing ℓ by ℓ/s. This simplification is valid because our bound involves the scale-invariant quantity s/ℓ. Likewise, if r_i or c_j is zero for some $i, j \in [n]$, then the corresponding rows and columns of $A^{(k)}$ contain only zeroes throughout the execution of the algorithm. We therefore restrict our attention to the submatrix indexed by positive entries of r and c.

Algorithm 2.3 updates one of the diagonal matrices $X^{(k)}$ and $Y^{(k)}$ at each step. Write x^k for the vector whose ith entry is $\log X^{(k)}_{ii}$, and similarly let y^k be the vector with entries $\log Y^{(k)}_{jj}$. We call these vectors the *scaling vectors* corresponding to $A^{(k)}$.

The proof of Theorem 2.3.1 relies on the following Lemmas that relate the successive improvements of the function f to the Kullback-Leibler divergence between target and current row/column sums. Similar ideas can be traced back at least to [105] where an analysis of Sinkhorn iterations for bi-stochastic targets is sketched in the context of a different problem, detecting the existence of a perfect matching in a bipartite graph.

Lemma 2.3.2. *If $k \geqslant 2$, then*

$$f(x^{k-1}, y^{k-1}) - f(x^k, y^k) = D_{\mathsf{KL}}(r \| r(A^{(k-1)})) + D_{\mathsf{KL}}(c \| c(A^{(k-1)})) .$$

Proof. Assume without loss of generality that k is odd, so that $c(A^{(k-1)}) = c$ and

$r(A^{(k)}) = r$. (If k is even, interchange the roles of r and c.) By definition,

$$\begin{aligned} f(x^{k-1}, y^{k-1}) - f(x^k, y^k) &= \sum_{ij} \left(A_{ij}^{(k-1)} - A_{ij}^{(k)} \right) + \langle r, x^k - x^{k-1} \rangle + \langle c, y^k - y^{k-1} \rangle \\ &= \sum_i r_i (x_i^k - x_i^{k-1}) \\ &= D_{\mathsf{KL}}(r \| r(A^{(k-1)}) + D_{\mathsf{KL}}(c \| c(A^{(k-1)})) \,, \end{aligned}$$

where we have used that: $\|A^{(k-1)}\|_1 = \|A^{(k)}\|_1 = 1$ and $Y^{(k)} = Y^{(k-1)}$; for all i,

$$r_i(x_i^k - x_i^{k-1}) = r_i \log \frac{X_{ii}^{(k)}}{X_{ii}^{(k-1)}} = r_i \log \frac{r_i}{r_i(A^{(k-1)})} \,;$$

and $D_{\mathsf{KL}}(c \| c(A^{(k-1)})) = 0$ since $c = c(A^{(k-1)})$. □

The next lemma has already appeared in the literature and we defer its proof to §2.5.

Lemma 2.3.3. *If A is a positive matrix with total mass s, then*

$$f(x^1, y^1) - \min_{x,y \in \mathbb{R}} f(x, y) \leqslant f(0, 0) - \min_{x,y \in \mathbb{R}} f(x, y) \leqslant \log \frac{s}{\ell} \,.$$

Lemma 2.3.4 (Pinsker's Inequality [219])**.** *For any $p, q \in \Delta_{n^2}$ such that p is absolutely continuous with respect to q, we have*

$$\|p - q\|_1 \leqslant \sqrt{2 D_{\mathsf{KL}}(p \| q)} \,.$$

Proof of Theorem 2.3.1. Let k^* be the first iteration such that

$$\|r(A^{(k^*)}) - r\|_1 + \|c(A^{(k^*)}) - c\|_1 \leqslant \varepsilon \,.$$

Pinsker's inequality implies that for any $k < k^*$, we have

$$\varepsilon^2 < (\|r(A^{(k)}) - r\|_1 + \|c(A^{(k)}) - c\|_1)^2 \leqslant 4(D_{\mathsf{KL}}(r \| r(A^{(k)})) + D_{\mathsf{KL}}(c \| c(A^{(k)})) \,,$$

so Lemmas 2.3.2 and 2.3.3 implies that we terminate in

$$k^* \leq 4 \, \varepsilon^{-2} \log \left(\frac{s}{\ell} \right)$$

steps, as claimed. □

2.3.2 Greedy sinkhorn

In addition to a new analysis of SINKHORN, we propose a new algorithm which enjoys the same convergence guarantee but with better performance in practice in terms of the number of arithmetic operations. Instead of alternating updates of all rows and columns of A, GREENKHORN simply updates the best single row or column at each step, thus updating only $O(n)$ entries of A, rather than $O(n^2)$ per iteration. Our analysis shows GREENKHORN might require n times more iterations than SINKHORN, so that the runtime guarantees of the two algorithms are the same. However, GREENKHORN tends to make much faster progress in practice.

This algorithm is an extremely natural modification of the RAS method, but previous analyses of RAS cannot be modified to extract any meaningful performance guarantees. On the other hand, our new analysis applies to GREENKHORN with only trivial modifications.

$A \leftarrow A/\|A\|_1$
while $\mathrm{dist}_{\mathcal{M}}(A, \mathcal{U}_{r,c}) > \varepsilon$ **do**
    $I \leftarrow \arg\max_i \rho(r_i, r_i(A))$
    $J \leftarrow \arg\max_j \rho(c_j, c_j(A))$
    if $\rho(r_I, r_I(A)) > \rho(c_J, c_J(A))$ **then**
        Rescale Ith row of A by $r_I/r_I(A)$
    else
        Rescale Jth row of A by $c_J/c_J(A)$
Output $B \leftarrow A$

Algorithm 2.4: GREENKHORN$(A, \mathcal{U}_{r,c}, \varepsilon')$

Pseudocode for GREENKHORN appears in Algorithm 2.4. As in SINKHORN,

$$\mathrm{dist}_{\mathcal{M}}(A, \mathcal{U}_{r,c}) = \|r(A) - r\|_1 + \|c(A) - c\|_1 .$$

Violations of the row and column constraints are measured by the distance function $\rho : \mathbb{R}_+ \times \mathbb{R}_+ \to [0, +\infty]$ given by

$$\rho(a, b) = b - a + a \log \frac{a}{b} .$$

Since ρ is not symmetric, it is not a metric; however, the function ρ is nonnegative and satisfies $\rho(a, b) = 0$ iff $a = b$.

We note that after $r(A)$ and $c(A)$ are computed once at the beginning of the algorithm, GREENKHORN can be implemented such that each iteration runs in only $O(n)$ time.

Theorem 2.3.5. *The algorithm* ***GREENKHORN*** *yields a matrix satisfying* $\operatorname{dist}_{\mathcal{M}}(B, \mathcal{U}_{r,c}) \leqslant \varepsilon'$ *in* $O(n(\varepsilon')^{-2}\log(s/\ell))$ *iterations. Since each iteration requires* $O(n)$ *operations, such a matrix can be found in* $O(n^2(\varepsilon')^{-2}\log(s/\ell))$ *arithmetic operations.*

The analysis requires the following Lemma, which is an easy modification of Lemma 2.3.2.

Lemma 2.3.6. *Let* A' *and* A'' *be successive iterates of* ***GREENKHORN****, with corresponding scaling vectors* (x', y') *and* (x'', y'')*. If* A'' *was obtained from* A' *by updating row* I*, then*

$$f(x', y') - f(x'', y'') = \rho(r_I, r_I(A'))\,,$$

and if it was obtained by updating column J*, then*

$$f(x', y') - f(x'', y'') = \rho(c_J, c_J(A'))\,.$$

We also require the following extension of Pinsker's inequality (the proof is deferred to §2.5).

Lemma 2.3.7. *For any* $\alpha \in \Delta_n, \beta \in \mathbb{R}^n_+$*, define* $\rho(\alpha, \beta) = \sum_i \rho(\alpha_i, \beta_i)$*. If* $\rho(\alpha, \beta) \leqslant 1$*, then*

$$\|\alpha - \beta\|_1 \leqslant \sqrt{7\rho(\alpha, \beta)}\,.$$

Proof of Theorem 2.3.5. We follow the proof of Theorem 2.3.1. If $\|r(A) - r\|_1 + \|c(A) - c\|_1 > \varepsilon$, then we make at least

$$\frac{1}{2n}(\rho(r, r(A)) + \rho(c, c(A))) \geqslant \frac{1}{14n}(\|r(A) - r\|_1^2 + \|c(A) - c\|_1^2) \geqslant \frac{1}{28n}\,\varepsilon^2$$

progress at each step, so we terminate in at most $28n\,\varepsilon^{-2}\log(s/\ell)$ iterations. □

■ 2.4 Empirical results

Cuturi [73] already gave experimental evidence that using `SINKHORN` to solve (2.2) outperforms state-of-the-art techniques for Optimal Transport. In this section, we provide strong empirical evidence that our proposed `GREENKHORN` algorithm significantly outperforms `SINKHORN` in terms of the number of arithmetic operations.

Remark 2.4.1 (Parallelizability)**.** ***GREENKHORN****, at least as written, is less parallelizable than* ***SINKHORN*** *since the latter can perform* n *row/column updates simultaneously using matrix-vector multiplication. On the other hand, as demonstrated numerically here,* ***GREENKHORN*** *can provide significant savings by only updating*

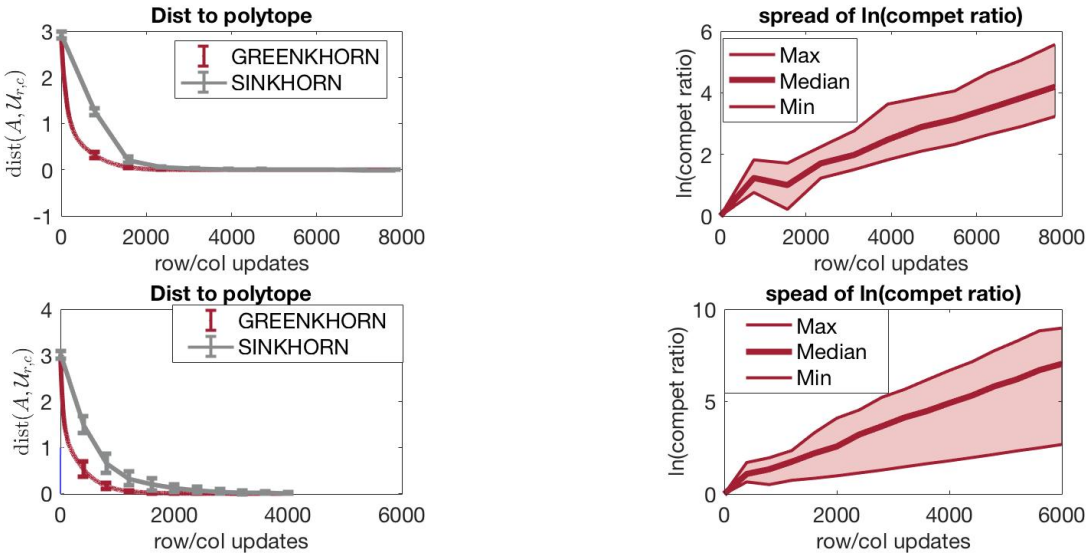


Figure 2.1: Comparison of GREENKHORN and SINKHORN on pairs of MNIST images of dimension 28×28 (top) and random images of dimension 20×20 with 20% foreground (bottom). Left: distance $\text{dist}(A, \mathcal{U}_{r,c})$ to the transport polytope (average over 10 random pairs of images). Right: maximum, median, and minimum values of the competitive ratio $\ln\left(\text{dist}(A_S, \mathcal{U}_{r,c})/\text{dist}(A_G, \mathcal{U}_{r,c})\right)$ over 10 runs.

the most salient rows/columns. In practice, the most prudent option may be a combination of the two, for example running ***GREENKHORN*** *with k row/column updates per iteration, where $1 \ll k \ll n$, in order to balance both the benefits of both parallelizability and fewer arithmetic operations. Here, we ignore parallelizability questions and focus solely on the hardware-independent question of the number of arithmetic operations.*

We consider transportation between pairs of $m \times m$ greyscale images, normalized to have unit total mass. The target marginals r and c represent two images in a pair, and $C \in \mathbb{R}^{m^2 \times m^2}$ is the matrix of ℓ_1 distances between pixel locations. Therefore, we aim to compute the earth mover's distance.

We run experiments on two datasets: *real images*, from MNIST, and *synthetic images*, as in Figure 2.2.

■ 2.4.1 MNIST

We first compare the behavior of GREENKHORN and SINKHORN on real images. To that end, we choose 10 random pairs of images from the MNIST dataset, and for each one analyze the performance of APPROXOT when using both GREENKHORN and SINKHORN for the approximate projection step. We add negligible noise 0.01

to each background pixel with intensity 0. Figure 2.1 paints a clear picture: GREENKHORN significantly outperforms SINKHORN both in the short and long term.

2.4.2 Random images

To better understand the empirical behavior of both algorithms in a number of different regimes, we devised a synthetic and tunable framework whereby we generate images by choosing a randomly positioned "foreground" square in an otherwise black background. The size of this square is a tunable parameter varied between 20%, 50%, and 80% of the total image's area. Intensities of background pixels are drawn uniformly from $[0, 1]$; foreground pixels are drawn uniformly from $[0, 50]$. Such an image is depicted in Figure 2.2, and results appear in Figure 2.1.

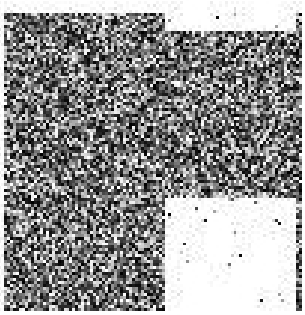


Figure 2.2: Synthetic image.

We perform two other experiments with random images in Figure 2.3. In the first, we vary the number of background pixels and show that GREENKHORN performs better when the number of background pixels is larger. We conjecture that this is related to the fact that GREENKHORN only updates salient rows and columns at each step, whereas SINKHORN wastes time updating rows and columns corresponding to background pixels, which have negligible impact. This demonstrates that GREENKHORN is a better choice especially when data is sparse, which is often the case in practice.

In the second, we consider the role of the regularization parameter η. Our analysis requires taking η of order $\log n/\varepsilon$, but Cuturi [73] observed that in practice η can be much smaller. Cuturi showed that SINKHORN outperforms state-of-the art techniques for computing OT distance even when η is a small constant, and Figure 2.3 shows that GREENKHORN runs faster than SINKHORN in this regime with no loss in accuracy.

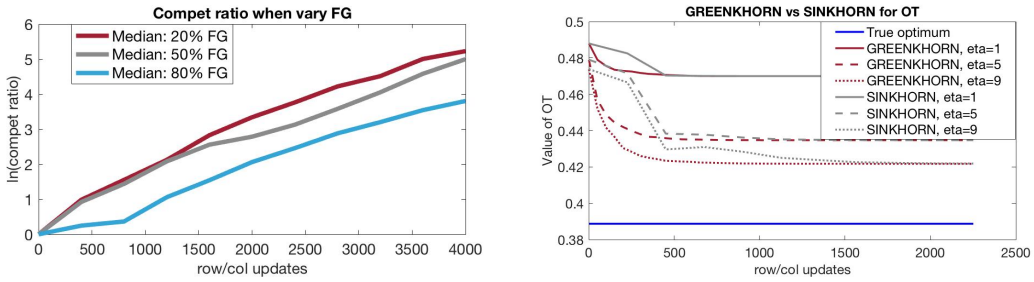


Figure 2.3: Left: Comparison of median competitive ratio for random images containing 20%, 50%, and 80% foreground. Right: Performance of `GREENKHORN` and `SINKHORN` for small values of η.

2.5 Deferred details

2.5.1 Proof of Theorem 2.2.4

Let G be the output of $\texttt{ROUND}(F, \mathcal{U}_{r,c})$. The entries of F are nonnegative throughout, and at the end of the algorithm err_r and err_c are both nonnegative, with $\|\mathrm{err}_r\|_1 = \|\mathrm{err}_c\|_1 = 1 - \|F\|_1$. Therefore the entries of G are nonnegative and

$$r(G) = r(F) + r(\mathrm{err}_r \mathrm{err}_c^\top / \|\mathrm{err}_r\|_1) = r(F) + \mathrm{err}_r = r\,,$$

and likewise $c(G) = c$. This establishes that $G \in \mathcal{U}_{r,c}$.

Let $\Delta = 1 - \|F\|_1$ be the total amount of mass subtracted from F during the course of the algorithm. Since we only remove mass from F from rows and columns which are over weight, we have

$$\begin{aligned}\Delta &\leqslant \sum_{i=1}^{n} (r(F)_i - r_i)_+ + \sum_{j=1}^{n} (c(F)_j - c_j)_+ \\ &\leqslant \frac{1}{2}(\|r(F) - r\|_1 + \|c(F) - c\|_1)\,.\end{aligned}$$

We obtain

$$\begin{aligned}\|G - F\|_1 &\leqslant \Delta + \|\mathrm{err}_r \mathrm{err}_c^\top\|_1 / \|\mathrm{err}_r\|_1 \\ &= 2\Delta \leqslant \|r(F) - r\|_1 + \|c(F) - c\|_1\,.\end{aligned}$$

Finally, we prove the $O(n^2)$ runtime bound follows by observing that each rescaling and computing the matrix $\mathrm{err}_r \mathrm{err}_c^\top / \|\mathrm{err}_r\|_1$ both require at most $O(n^2)$ time. □

2.5.2 Proof of Lemma 2.3.3

The first inequality follows from the fact that rescaling the rows or columns of A always leads to improvement in the value of f. Then as in the proof of Lemma 2.3.2,

$$\begin{aligned} f(x^{(0)}, y^{(0)}) - f(x^{(1)}, y^{(1)}) &= \langle r, x^{(1)} \rangle + \langle c, y^{(1)} \rangle \\ &= \sum_{ij} A^{(1)}_{ij} \log \frac{A^{(1)}_{ij}}{A_{ij}} \\ &= D_{\mathsf{KL}}(A^{(1)} \| A) \geqslant 0 \,. \end{aligned}$$

We now prove the second claim. Recall that we assume that we have rescaled A in such a way that $\|A\|_1 = 1$ and its smallest entry is ℓ/s. Since A is positive, [203] shows that $\Pi_{\mathcal{S}}(A)$ exists and is unique. Let (x^*, y^*) be corresponding scaling factors. Then

$$f(x^{(0)}, y^{(0)}) - f(x^*, y^*) = \langle r, x^* \rangle + \langle c, y^* \rangle \,.$$

Since

$$A_{ij} e^{x^*_i + y^*_j} \leqslant \sum_{ij} A_{ij} e^{x^*_i + y^*_j} = 1 \,,$$

we have

$$x^*_i + y^*_j \leqslant \log \frac{s}{\ell} \,,$$

for all $i, j \in [n]$. Because r and c are both probability vectors,

$$\langle r, x^* \rangle + \langle c, y^* \rangle \leqslant \log \frac{s}{\ell} \,.$$

□

2.5.3 Proof of Lemma 2.3.6

We prove only the case where a row was updated, since the column case is exactly the same.

By definition,

$$f(x', y') - f(x'', y'') = \sum_{ij} (A'_{ij} - A''_{ij}) + \langle r, x'' - x' \rangle + \langle c, y'' - y' \rangle \,.$$

Observe that A' and A'' differ only in the Ith row, and x'' and x' differ only in the Ith entry, and $y'' = y'$. Hence

$$\begin{aligned} f(x', y') - f(x'', y'') &= r_I(A') - r_I(A'') + r_I(x''_I - x'_I) \\ &= \rho(r_I, r_I(A')) \,, \end{aligned}$$

where we have used the fact that $r_I(A'') = r_I$ and $x''_I - x'_I = \log(r_I / r_I(A'))$. □

■ 2.5.4 Proof of Lemma 2.3.7

Let $s = \sum_i \beta_i$, and write $\bar{\beta} = \beta/s$. The definition of ρ implies

$$\begin{aligned}\rho(\alpha, \beta) &= \sum_i (\beta_i - \alpha_i) + \alpha_i \log \frac{\alpha_i}{\beta_i} \\ &= s - 1 + \sum_i \alpha_i \log \frac{\alpha_i}{s\bar{\beta}_i} \\ &= s - 1 - (\log s) \sum_i \alpha_i + D_{\mathsf{KL}}(\alpha \| \bar{\beta}) \\ &= s - 1 - \log s + D_{\mathsf{KL}}(\alpha \| \bar{\beta}) \,.\end{aligned}$$

Note that both $s - 1 - \log s$ and $D_{\mathsf{KL}}(\alpha \| \bar{\beta})$ are nonnegative. If $\rho(\alpha, \beta) \leqslant 1$, then in particular $s - 1 - \log s \leqslant 1$, and it can be seen that $s - 1 - \log s \geqslant (s-1)^2/5$ in this range. Applying Lemma 2.3.4 (Pinsker's inequality) yields

$$\rho(\alpha, \beta) \geqslant \frac{1}{5}(s-1)^2 + \frac{1}{2}\|\alpha - \bar{\beta}\|_1^2 \,.$$

By the triangle inequality and convexity,

$$\begin{aligned}\|\alpha - \beta\|_1^2 &\leqslant (\|\bar{\beta} - \beta\|_1 + \|\alpha - \bar{\beta}\|_1)^2 \\ &= (|s-1| + \|\alpha - \bar{\beta}\|_1)^2 \\ &\leqslant \frac{7}{5}(s-1)^2 + \frac{7}{2}\|\alpha - \bar{\beta}\|_1^2 \,.\end{aligned}$$

The claim follows from the above two displays. □

Chapter 3

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

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 $K \in \mathbb{R}^{n\times n}_{\geqslant 0}$ 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 significantly improved runtime bounds for cyclic, greedy, and parallelized variants of Osborne's algorithm.

3.1 Introduction

Let $\mathbf{1}$ denote the all-ones vector in $\mathbb{R}^n$. A nonnegative square matrix $A \in \mathbb{R}^{n\times n}_{\geqslant 0}$ 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.

$$r(A) = c(A). \tag{3.1}$$

This chapter revisits the classical problem of *Matrix Balancing*—sometimes also called *diagonal similarity scaling* or *line-sum-symmetric scaling*—which asks: given a nonnegative matrix $K \in \mathbb{R}^{n\times n}_{\geqslant 0}$, 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 [166, 175] 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 [198] for $p = \infty$ and then [167] for p finite. See Remark 3.1.5 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 chapter 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 [166, 175] and matrix exponentiation [113, 228]. 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 [166]. Moreover, in practice, the runtime of (approximate) Matrix Balancing is essentially negligible compared to the runtime of these downstream tasks [180, §11.6.1]. The ubiquity of these applications has led to the implementation of Matrix Balancing in most linear algebra software packages, including EISPACK [205], LAPACK [23], R [184], and MATLAB [150]. In fact, Matrix Balancing is performed by default in the command for eigenvalue decomposition in MATLAB [151] and in the command for matrix exponentation for R [98]. Matrix Balancing also has other diverse applications in economics [195], information retrieval [217], and combinatorial optimization [18].

In practice, Matrix Balancing is performed approximately rather than exactly, since this can be done efficiently and typically suffices for applications. Specifically,

[1] K can be balanced if and only if K is irreducible [83]. This can be efficiently checked in linear time [215].

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.,

$$\frac{\|r(A) - c(A)\|_1}{\sum_{ij} A_{ij}} \leqslant \varepsilon\,. \tag{3.2}$$

Remark 3.1.1 (ℓ_1 versus ℓ_2 error criterion)**.** *Several papers [126, 167] study approximate Matrix Balancing with ℓ_2 error criterion—rather than ℓ_1 as done here in* (3.2) *and in e.g., [161]—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 [18]—in large part due to its natural interpretations in both probabilistic problems (as total variation imbalance) and graph theoretic problems (as netflow imbalance) [18, Remarks 2.1 and 5.8].*[2] *Note also that the approximate balancing criterion* (3.2) *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 [167], 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 3.1.5.)*

■ 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.

■ 3.1.1.1 Practical state-of-the-art

Ever since its invention in 1960, *Osborne's algorithm* has been the algorithm of choice for practitioners [166, 175]. 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

[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 [19] and Bipartite Perfect Matching [57].

[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.

column sum $c_k(A)$ of the current balancing $A = DKD^{-1}$ agree.[4] A more precise statement is in Algorithm 3.1 later.

The classical version of Osborne's algorithm, henceforth called *Round-Robin Cyclic Osborne*, chooses the update coordinates by repeatedly cycling through $\{1, \dots, n\}$. 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 [167]. The paper [167] 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 $\arg\max_k(\sqrt{r_k(A)}-\sqrt{c_k(A)})^2$. 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 chapter seeks to address are:

[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 [175], some implementations have two minor modifications: a preprocessing 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 [215], and the latter is solely to safeguard against numerical precision issues in practice.

[6] Note that in [167], bounds are written for the ℓ_2 error criterion; see Remark 3.1.1.

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 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?

■ 3.1.1.2 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 [126], 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, [67][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 $\{(i,j) : K_{ij} > 0\}$ [67, 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.

■ 3.1.2 Contributions

Random Osborne converges in near-linear time. Our main result (Theorem 3.5.1) 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 [167], while a different, impractical algorithm has theoretical runtime which is (conditionally) near-linear in m [67]. Property (ii) shows that improving

[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}.

[8] Similar runtimes were also developed by [9].

Variant	Best runtime (arithmetic ops)	Polylog bits
Cyclic (Round-Robin)	$\tilde{O}(mn^2/\varepsilon^2)$ [167]	No
Cyclic (Random-Reshuffle)	$\boldsymbol{\tilde{O}(mn/\varepsilon)}$ **(Theorem 3.6.1)**	**Yes (Theorem 3.8.1)**
Weighted Random	$\tilde{O}(n^2/\varepsilon^2)$ [167]	Yes [167]
Greedy	$\tilde{O}((n^2/\varepsilon^2) \wedge (n^3/\varepsilon))$ [167] $\longrightarrow$ $\boldsymbol{\tilde{O}(n^2/\varepsilon)}$ **(Theorem 3.4.1)**	No
Random	$\boldsymbol{\tilde{O}(m/\varepsilon)}$ **(Theorem 3.5.1)**	**Yes (Theorem 3.8.1)**

Table 3.1: Variants of Osborne's algorithm for balancing a matrix $K \in \mathbb{R}^{n\times n}_{\geqslant 0}$ with m nonzeros to ε ℓ_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 [167], bounds are written for the ℓ_2 error criterion; see Remark 3.1.1. See the main text for descriptions of each variant, and also §3.2.4 for more details on Random-Reshuffle Cyclic, Greedy, and Random Osborne. Our new bounds are in bold. Theorems 3.4.1 and 3.6.1 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 3.5.1, provides the first near-linear runtime for any variant of Osborne's algorithm.

the runtime dependence in ε from ε^{-2} to ε^{-1} does not require paying a costly factor of n (c.f., [167]).

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 3.1.2 (Informal version of Theorem 3.5.1)**.** *Random Osborne solves the approximate Matrix Balancing problem on input $K \in \mathbb{R}^{n\times n}_{\geqslant 0}$ to accuracy $\varepsilon > 0$ after*

$$O\left(\frac{m}{\varepsilon}\left(\frac{1}{\varepsilon} \wedge d\right)\log\kappa\right), \tag{3.3}$$

arithmetic operations, both in expectation and with high probability.

We make several remarks about Theorem 3.1.2. First, we interpret the runtime (3.3). 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., [43, Theorem 10.10]). Note that the complexity of Matrix Balancing is intimately related to

[9]Not to be confused with the different randomized variant of Osborne's algorithm in [167, §5], which draws coordinates with non-uniform probabilities. We call that algorithm Weighted Random Osborne to avoid confusion.

Variant	Best runtime (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

Table 3.2: Parallelized variants of Osborne's algorithm for balancing a matrix $K \in \mathbb{R}^{n\times n}_{\geqslant 0}$ with m nonzeros to ε ℓ_1 accuracy, given a partitioning of the dataset into p blocks (see §3.2.5 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 3.7.1, and the bit-complexity bounds are in Theorem 3.8.1.

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) [166]. 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 3.1.2 has tails that decay exponentially fast. This is optimal with our analysis, see Remark 3.5.4.

Next, we comment on the $\log\kappa$ term in the runtime. This term appears in all other state-of-the-art runtimes [67, 167] and is mild: indeed, $\log\kappa \leqslant \log m + \log(\max_{ij} K_{ij}/\min_{ij:K_{ij}>0} K_{ij})$, 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 3.5.1 (a.k.a., the formal version of Theorem 3.1.2) gives a faster runtime than any existing practical algorithm, see Table 3.1. If comparing to the (impractical) algorithm of [67] on a purely theoretical plane, neither runtime dominates the other, and which is faster depends on the precise parameter regime: [67] 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.

Finally, we remark about bit-complexity. In §3.8, 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 3.8.1 for formal statement.

Simple, streamlined analysis for different Osborne variants. We prove Theorem 3.1.2 using an intuitive potential argument (overviewed in §3.1.3 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 3.1 and 3.2

[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 [175].

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 [167]). Specifically, in Theorem 3.4.1, 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 [167]. (The other improved $\log n$ factor comes from simplifying the data structure used for efficient greedy updates, see Remark 3.2.7.)

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 3.6.1). 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 [167]. Although the version of Cyclic Osborne we study is different than the one studied in [167], 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 3.8.1), whereas the analysis of Round-Robin Cyclic Osborne in [167] 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 §3.2.5. These runtimes bounds are summarized in Table 3.2. 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 3.1.2 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 §3.2.5 for further details). Property (ii) shows that the speedup of parallelization comes at no cost in the total work.

■ 3.1.3 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 rig-

orous 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 §3.2.3. Note also that Osborne's algorithm is equivalent to Exact Coordinate Descent on this function—which, importantly, is convex after a re-parameterization; see §3.2.4. In the interest of accessibility, the below overview describes our approach at an informal level that does not require further background. Later, §3.2 provides these preliminaries, and §3.3 gives the technical details of the potential argument.

■ 3.1.3.1 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 §3.4. Since each Greedy Osborne iteration takes $O(n)$ arithmetic operations (see §3.2.4), 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

$$(\text{per-iteration decrease in potential}) \gtrsim \frac{\mathsf{H}^2\left(r(P), c(P)\right)}{n}, \tag{3.4}$$

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 $\mathsf{H}^2\left(r(P), c(P)\right)$ 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 (3.2)). Therefore

$$(\text{per-iteration decrease in potential}) \gtrsim \frac{\varepsilon^2}{n} \tag{3.5}$$

for each iteration before convergence. Since the potential is initially not very large (at most $\log \kappa$, see Lemma 3.3.1) 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 (3.5)

$$(\text{per-iteration decrease in potential}) \gtrsim \frac{1}{n} \left[\frac{(\text{current potential})}{R} \vee \varepsilon \right]^2 \tag{3.6}$$

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 §3.4).

■ 3.1.3.2 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., (3.4), (3.5), and (3.6)), 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 §3.5.2).

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

Observation 3.1.3 (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 §3.5.1.

■ 3.1.3.3 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 §3.6. 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., [212, 233].

■ 3.1.3.4 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 (3.4), (3.5), and (3.6), 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 §3.7.

■ 3.1.3.5 Key differences from previous approaches

The only other polynomial-time analysis of Osborne's algorithm also uses a potential argument [167]. 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 3.4.2) is specifically tailored to Greedy Osborne[11] and seems unextendable to other variants such as Random Osborne. In particular, this precludes obtaining the near-linear runtime shown in this chapter. Another key difference is that they do *not* use convexity of their potential (explicitly written on [167, page 157]), whereas we exploit not only convexity but also *log-convexity* (note our potential is the logarithm of theirs). Specifically, they use [167, 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 3.3.3) that saves this factor of n for well-connected graphs by exploiting log-convexity of their potential.

3.1.4 Other related work

We briefly remark about several related lines of work. Reference [63] gives heuristics for speeding up Osborne's algorithm on sparse matrices in practice, but does not provide runtime bounds. Reference [168] 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 [148] gives an asynchronous distributed version of Osborne's algorithm with applications to epidemic suppression.

Remark 3.1.4 (Fast Coordinate Descent)**.** *Since Osborne's algorithm is Exact Coordinate Descent on a certain associated convex optimization problem (details in §3.2.4), 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*

[11]Specifically, to prove their Lemma 3.1, [167] uses in (3.6) the inequality $\max_{i\in[n]} a_i/b_i \geqslant (\frac{1}{n}\sum_{j=1}^n a_j)/(\frac{1}{n}\sum_{j=1}^n b_j)$ for positive $a_1,\ldots,a_n,b_1,\ldots,b_n$. Extending their analysis of Greedy 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 (3.21) and Lemma 3.5.2).

[12]E.g., consider applying the state-of-the-art guarantees of [8, 163] 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 §3.2.3) to δ additive accuracy, where L_i is the smoothness of Φ on coordinate i. By [126, 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 $L_i \leqslant 1$ and $\|x^*\|_2 \leqslant \sqrt{n}d\log\kappa$ by Corollary 3.3.6 therefore yields a bound

of the function. In order to achieve the near-linear time bounds in this chapter, we heavily exploit the further global structure of the specific convex optimization problem at hand.

Remark 3.1.5 (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 3.1.1.) The Matrix Balancing problem studied in this chapter is a special case of this: it is ℓ_1 balancing a nonnegative matrix. However, 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., [188]. 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 [194, 238] as well as a natural analog of Osborne's algorithm [166]. 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 [62], and the first runtime bound was shown only a few years ago [198]. 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 [62]), but also the algorithms are quite different (even the analogous versions of Osborne's algorithm) and their analyses do not appear to carry over [167].*

Remark 3.1.6 (Matrix Scaling and Sinkhorn's algorithm)**.** *Here we contextualize the Matrix Balancing problem studied in this chapter with the related problem of Matrix Scaling studied in Chapter 2. Recall that the Matrix Scaling problem is: given $K \in \mathbb{R}^{n\times n}_{\geqslant 0}$ and vectors $\mu, \nu \in \mathbb{R}^n_{\geqslant 0}$ 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 [118]. In analog to Osborne's algorithm for Matrix Balancing, there is a simple iterative procedure (Sinkhorn's algorithm) for Matrix*

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.

Scaling [203]. Chapter 2 shows that Sinkhorn's algorithm converges in near-linear time [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 chapter are deferred to §3.10.4.*

3.1.5 Roadmap

§3.2 recalls preliminary background. §3.3 establishes the key lemmas in the potential argument. §3.4, §3.5, §3.6, and §3.7 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 §3.8. §3.9 concludes with several open questions.

3.2 Preliminaries

3.2.1 Notation

For the convenience of the reader, we collect here the notation used commonly throughout the chapter. We reserve $K \in \mathbb{R}^{n\times n}_{\geqslant 0}$ 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 $\mathrm{diag}(v) \in \mathbb{R}^{n\times n}$. The set of discrete probability distributions on n atoms is identified with the simplex $\Delta_n = \{p \in \mathbb{R}^n_{\geqslant 0} : \sum_{i=1}^n p_i = 1\}$. Let $\mu, \nu \in \Delta_n$. Their Hellinger distance is $\mathsf{H}(\mu,\nu) = \sqrt{\frac{1}{2}\sum_{\ell=1}^n (\sqrt{\mu_\ell} - \sqrt{\nu_\ell})^2}$, and their total variation distance is $\mathsf{TV}(\mu,\nu) = \|\mu - \nu\|_1/2$. 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.

■ 3.2.2 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 $\operatorname{diag}(e^x)$). This is in part to avoid bit-complexity issues (see §3.8).

Definition 3.2.1 (Matrix Balancing)**.** *The* Matrix Balancing problem $\texttt{BAL}(K)$ *for input* $K \in \mathbb{R}^{n\times n}_{\geqslant 0}$ *is to compute a vector* $x \in \mathbb{R}^n$ *such that* $\operatorname{diag}(e^x)K\operatorname{diag}(e^{-x})$ *is balanced.*

Definition 3.2.2 (Approximate Matrix Balancing)**.** *The* approximate Matrix Balancing problem $\texttt{ABAL}(K,\varepsilon)$ *for inputs* $K \in \mathbb{R}^{n\times n}_{\geqslant 0}$ *and* $\varepsilon > 0$ *is to compute a vector* $x \in \mathbb{R}^n$ *such that* $\operatorname{diag}(e^x)K\operatorname{diag}(e^{-x})$ *is* ε*-balanced (see* (3.1)*).*

$K \in \mathbb{R}^{n\times n}_{\geqslant 0}$ is said to be *balanceable* if $\texttt{BAL}(K)$ 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 $K \in \mathbb{R}^{n\times n}_{\geqslant 0}$ and $\varepsilon > 0$), and moreover that this is efficiently reducible to approximately balancing balanceable matrices, see, e.g., [63, 67]. 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 3.2.3 (Characterization of balanceability)**.** $K \in \mathbb{R}^{n\times n}_{\geqslant 0}$ *is balanceable if and only if it is irreducible—i.e., if and only if* G_K *is strongly connected [166].*

■ 3.2.3 Matrix Balancing as convex optimization

Key to to our analysis—as well as much of the other Matrix Balancing literature (e.g., [67, 126, 161, 167])—is the classical connection between (approximately) balancing a matrix $K \in \mathbb{R}^{n\times n}_{\geqslant 0}$ and (approximately) solving the convex optimization problem

$$\min_{x\in\mathbb{R}^n} \Phi(x) := \log \sum_{ij} e^{x_i - x_j} K_{ij}. \tag{3.7}$$

In words, balancing K is equivalent to scaling DKD^{-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:

$$\nabla\Phi(x) = \frac{A\mathbf{1} - A^T\mathbf{1}}{\sum_{ij} A_{ij}}, \quad \text{where } A := \operatorname{diag}(e^x)K\operatorname{diag}(e^{-x}). \tag{3.8}$$

Indeed, solutions of $\texttt{BAL}(K)$ are points where this gradient vanishes, and thus are in correspondence with minimizers of Φ. This also holds approximately: solutions of $\texttt{ABAL}(K, \varepsilon)$ 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 $\|\nabla\Phi(x)\|_1 \leqslant \varepsilon$. The following lemma summarizes these classical connections; for a proof see, e.g., [126].

Lemma 3.2.4 (Matrix Balancing as convex optimization)**.** *Let $K \in \mathbb{R}^{n\times n}_{\geqslant 0}$ and $\varepsilon > 0$. Then:*

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

■ 3.2.4 Osborne's algorithm as coordinate descent

Lemma 3.2.4 equates the *problems* of (approximate) Matrix Balancing and (approximate) optimization of (3.7). This correspondence extends to *algorithms*. In particular, in the sequel, we repeatedly leverage the following known connection, which appears in, e.g., [167].

Observation 3.2.5 (Osborne's algorithm as Cordinate Descent)**.** *Osborne's algorithm for Matrix Balancing is equivalent to Exact Coordinate Descent for optimizing* (3.7)*.*

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

$$x \leftarrow \operatorname*{arg\,min}_{z \in \{x+\alpha e_k \,:\, \alpha\in\mathbb{R}\}} \Phi(z), \tag{3.9}$$

where e_k denotes the k-th standard basis vector in $\mathbb{R}^n$. In words, this update (3.9) 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 §3.1, is an iterative algorithm for Matrix Balancing that repeatedly balances row/column pairs. Algorithm 3.1 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 $\operatorname{diag}(e^x)$ themselves. The connection in Observation 3.2.5 is thus, stated more precisely, that

Osborne's algorithm is a specification of the Exact Coordinate Descent algorithm to minimizing the function Φ *in* (3.7) *with initialization of* $\mathbf{0}$. 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 (3.8) 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.

Input: Matrix $K \in \mathbb{R}^{n \times n}_{\geqslant 0}$ and accuracy $\varepsilon > 0$
Output: Vector $x \in \mathbb{R}^n$ that solves $\texttt{ABAL}(K, \varepsilon)$

1: $x \leftarrow \mathbf{0}$ ▷ Initialize
2: **while** $\operatorname{diag}(e^x) K \operatorname{diag}(e^{-x})$ is not ε-balanced **do**
3: Choose update coordinate $k \in [n]$
4: $x_k \leftarrow x_k + \frac{\log(c_k(\operatorname{diag}(e^x) K \operatorname{diag}(e^{-x}))) - \log(r_k(\operatorname{diag}(e^x) K \operatorname{diag}(e^{-x})))}{2}$ ▷ Update
5: **return** x

Algorithm 3.1: Osborne's algorithm for Matrix Balancing. The variant (e.g., Greedy, Random, etc.) depends on how the update coordinate k is chosen in Line 3.

We note that besides elucidating Observation 3.2.5, the log-domain implementation of Osborne's Algorithm in Algorithm 3.1 is also critical for numerical precision, both in theory and practice.

Remark 3.2.6 (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* $\operatorname{diag}(e^x)$*, operate on* K *through* $\log K_{ij}$ *(see Remark 3.8.2), 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 §3.8, we show that essentially just these modifications enable a provably logarithmic bit-complexity for several variants of Osborne's algorithm (Theorem 3.8.1).*

It remains to discuss the choice of update coordinate in Osborne's algorithm (Line 3 of Algorithm 3.1), 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 $A := \operatorname{diag}(e^x) K \operatorname{diag}(e^{-x})$ disagree most, as measured by

$$\arg\max_{k \in [n]} \left| \sqrt{r_k(A)} - \sqrt{c_k(A)} \right|. \tag{3.10}$$

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

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

Remark 3.2.7 (Efficient implementation of Greedy)**.** *In order to efficiently compute* (3.10)*, 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).*

■ 3.2.5 Parallelizing Osborne's algorithm via graph coloring

For scalability, parallelization of Osborne's algorithm can be critical. It is well-known (see, e.g., [36]) 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 3.2.8 (Block Osborne as Block Coordinate Descent)**.** *Extending Observation 3.2.5,* 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

$$\frac{1}{|S_\ell|} \sum_{k \in S_\ell} \left(\sqrt{r_k(A)} - \sqrt{c_k(A)} \right)^2 \tag{3.11}$$

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

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

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 §3.7). However, finding the (approximately) smallest coloring is NP-hard [90, 129, 242]. 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 3.7.3.

■ 3.3 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 §3.1.3 for a high-level overview of the argument.

First, we recall the following standard bound on the initial potential. This appears in, e.g., [67, 167]. For completeness, we briefly recall the simple proof. Below, we denote the optimal value of the convex optimization problem (3.7) by $\Phi^* := \min_{x \in \mathbb{R}^n} \Phi(x)$.

Lemma 3.3.1 (Bound on initial potential)**.** $\Phi(\mathbf{0}) - \Phi^* \leqslant \log \kappa$.

Proof. It suffices to show $\Phi^* \geqslant \log K_{\min}$. Since K is balanceable, G_K is strongly connected (Lemma 3.2.3), thus G_K contains a cycle. By an averaging argument, this cycle contains an edge (i, j) such that $x^*_i - x^*_j \geqslant 0$. Thus $\Phi^* \geqslant \log(e^{x^*_i - x^*_j} K_{ij}) \geqslant \log K_{\min}$. □

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 [167, Lemma 2.1].

Lemma 3.3.2 (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, $A := \operatorname{diag}(e^x) K \operatorname{diag}(e^{-x})$ denote the scaling corresponding to x, and $P := A/(\sum_{ij} A_{ij})$ its normalization. Then*

$$\Phi(x) - \Phi(x') = -\log\left(1 - \left(\sqrt{r_k(P)} - \sqrt{c_k(P)}\right)^2\right). \tag{3.12}$$

Proof. Let $A' := \operatorname{diag}(e^{x'}) K \operatorname{diag}(e^{-x'})$ 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 (3.12). □

In the sequel, we lower bound the per-iteration progress in (3.12) by $(\sqrt{r_k(P)} - \sqrt{c_k(P)})^2$ using the elementary inequality $-\log(1-z) \geqslant z$. 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

$$\frac{1}{n}\sum_{k=1}^{n} (\sqrt{r_k(P)} - \sqrt{c_k(P)})^2 = \frac{2}{n}\mathsf{H}^2\big(r(P), c(P)\big). \tag{3.13}$$

(For Random Osborne, this statement requires an expectation; see §3.5.) The rest of this section establishes the main ingredient in the potential argument: Proposition 3.3.3 lower bounds this Hellinger imbalance, and thereby lower bounds the per-iteration progress. Note that Proposition 3.3.3 is stated for "nontrivial balancings", i.e., $x \in \mathbb{R}^n$ satisfying $\Phi(x) \leqslant \Phi(\mathbf{0})$. 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 3.3.2.

Proposition 3.3.3 (Lower bound on Hellinger imbalance)**.** *Consider any $x \in \mathbb{R}^n$. Let $A := \operatorname{diag}(e^x) K \operatorname{diag}(e^{-x})$ denote the corresponding scaling, and let $P := A/\sum_{ij} A_{ij}$ denote its normalization. If $\Phi(x) \leqslant \Phi(\mathbf{0})$ and A is not ε-balanced, then*

$$\mathsf{H}^2\big(r(P), c(P)\big) \geqslant \frac{1}{8}\left(\frac{\Phi(x) - \Phi^*}{d \log \kappa} \vee \varepsilon\right)^2. \tag{3.14}$$

To prove Proposition 3.3.3, 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) [78]. A short, simple proof via Cauchy-Schwarz is provided for completeness.

Lemma 3.3.4 (Hellinger versus ℓ_1 inequality)**.** *If $\mu, \nu \in \Delta_n$, then*

$$\mathsf{H}(\mu, \nu) \geqslant \frac{1}{2\sqrt{2}} \|\mu - \nu\|_1. \tag{3.15}$$

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

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., [67, Lemma 4.24])—however, the proof extends essentially without modifications; details are provided briefly for completeness.

Lemma 3.3.5 (Variation norm of nontrivial balancings)**.** *If $x \in \mathbb{R}^n$ satisfies $\Phi(x) \leqslant \Phi(0)$, then $\|x\|_{\mathrm{var}} \leqslant d \log \kappa$.*

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

From Lemma 3.3.5, we deduce the following bound.

Corollary 3.3.6 (ℓ_∞ distance of nontrivial balancings to minimizers)**.** *If $x \in \mathbb{R}^n$ satisfies $\Phi(x) \leqslant \Phi(0)$, then there exists a minimizer x^* of Φ such that $\|x - x^*\|_\infty \leqslant d \log \kappa$.*

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 $\|x - x^*\|_\infty = (\max_i (x_i - x_i^*) - \min_j (x_j - x_j^*))/2 \leqslant ((\max_i x_i - \min_j x_j) + (\max_i x_i^* - \min_j x_j^*))/2 = (\|x\|_{\mathrm{var}} + \|x^*\|_{\mathrm{var}})/2$. By Lemma 3.3.5, this is at most $d \log \kappa$. □

We are now ready to prove Proposition 3.3.3.

Proposition 3.3.3. Since P is normalized, its marginals $r(P)$ and $c(P)$ are both probability distributions in Δ_n. Thus by Lemma 3.3.4,

$$\mathsf{H}^2\big(r(P), c(P)\big) \geqslant \frac{1}{8} \|r(P) - c(P)\|_1^2. \tag{3.16}$$

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

$$\|r(P) - c(P)\|_1 \geqslant \frac{\Phi(x) - \Phi(x^*)}{d \log \kappa}, \tag{3.17}$$

which we show presently. By convexity of Φ (Lemma 3.2.4) and then Hölder's inequality,

$$\Phi(x) - \Phi(x^*) \leqslant \langle \nabla\Phi(x), x - x^* \rangle \leqslant \|\nabla\Phi(x)\|_1 \|x - x^*\|_\infty \tag{3.18}$$

for any minimizer x^* of Φ. Now by Corollary 3.3.6, there exists a minimizer x^* such that $\|x - x^*\|_\infty \leqslant d \log \kappa$; and by (3.8), the gradient is $\nabla\Phi(x) = r(P) - c(P)$. Re-arranging (3.18) therefore establishes (3.17). □

■ 3.4 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 §3.1.2 for further discussion of the result, and §3.1.3.1 for a proof sketch.

Theorem 3.4.1 (Convergence of Greedy Osborne)**.** *Given a balanceable matrix $K \in \mathbb{R}^{n\times n}_{\geqslant 0}$ and accuracy $\varepsilon > 0$, Greedy Osborne solves* `ABAL`(K, ε) *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 3.4.2 (Potential decrease of Greedy Osborne)**.** *Consider any $x \in \mathbb{R}^n$ for which the corresponding scaling $A := \operatorname{diag}(e^x) K \operatorname{diag}(e^{-x})$ is not ε-balanced. If x' is the next iterate obtained from a Greedy Osborne update, then*

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

Proof. Using in order Lemma 3.3.2, the inequality $-\log(1 - z) \geqslant z$ which holds

for any $z \in \mathbb{R}$, the definition of Greedy Osborne, and then Proposition 3.3.3,

$$\Phi(x) - \Phi(x') = -\log(1 - \left(\sqrt{r_k(P)} - \sqrt{c_k(P)}\right)^2) \tag{3.19}$$

$$\geqslant \left(\sqrt{r_k(P)} - \sqrt{c_k(P)}\right)^2 \tag{3.20}$$

$$\geqslant \frac{1}{n}\sum_{\ell=1}^{n}\left(\sqrt{r_\ell(P)} - \sqrt{c_\ell(P)}\right)^2 \tag{3.21}$$

$$\geqslant \frac{1}{4n}\left(\frac{\Phi(x) - \Phi^*}{d\log\kappa} \vee \varepsilon\right)^2. \tag{3.22}$$

□

Theorem 3.4.1. Let $x^{(0)} = \mathbf{0}, x^{(1)}, x^{(2)}, \ldots$ denote the iterates, and let τ be the first iteration for which $\operatorname{diag}(e^x)K\operatorname{diag}(e^{-x})$ is ε-balanced. Since the number of arithmetic operations per iteration is amortized to $O(n)$ by Remark 3.2.7, 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 3.4.2, for each $t \in \{0, 1, \ldots, \tau - 1\}$ we have

$$\Phi(x^{(t)}) - \Phi(x^{(t+1)}) \geqslant \frac{1}{4n}\left(\frac{\Phi(x^{(t)}) - \Phi^*}{d\log\kappa} \vee \varepsilon\right)^2. \tag{3.23}$$

Case 1: $\varepsilon^{-1} \leqslant d$**.** By the second bound in (3.23), the potential decreases by at least $\varepsilon^2/4n$ in each iteration. Since the potential is initially at most $\log\kappa$ by Lemma 3.3.1 and is always nonnegative by definition, the total number of iterations is at most

$$\tau \leqslant \frac{\log\kappa}{\varepsilon^2/4n} = \frac{4n\log\kappa}{\varepsilon^2}. \tag{3.24}$$

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

$$\tau_2 \leqslant \frac{\alpha}{\varepsilon^2/4n} = \frac{4nd\log\kappa}{\varepsilon}. \tag{3.25}$$

To bound τ_1, partition this phase further as follows. Let $\phi_0 := \log\kappa$ and $\phi_i := \phi_{i-1}/2$ for $i = 1, 2, \ldots$ until $\phi_N \leqslant \alpha$. 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 (3.23). Thus

$$\tau_{1,i} \leqslant \frac{\phi_{i-1} - \phi_i}{(\frac{\phi_i}{d \log \kappa})^2/4n} = \frac{4nd^2 \log^2 \kappa}{\phi_i}. \tag{3.26}$$

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

$$\tau_1 = \sum_{i=1}^{N} \tau_{1,i} \leqslant \frac{16nd^2 \log \kappa^2}{\alpha} = \frac{16nd \log \kappa}{\varepsilon}. \tag{3.27}$$

By (3.25) and (3.27), the total number of iterations is at most $\tau = \tau_1 + \tau_2 \leqslant 20nd\,\varepsilon^{-1} \log \kappa$. □

■ 3.5 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 §3.1.2 for further discussion of the result, and §3.1.3.2 for a proof sketch.

Theorem 3.5.1 (Convergence of Random Osborne)**.** *Given a balanceable matrix $K \in \mathbb{R}^{n \times n}_{\geqslant 0}$ and accuracy $\varepsilon > 0$, Random Osborne solves $\texttt{ABAL}(K, \varepsilon)$ 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 $\delta > 0$,*

$$\mathbb{P}\left(T \leqslant c\left(\tfrac{m}{\varepsilon}(\tfrac{1}{\varepsilon} \wedge d) \log \kappa \log \tfrac{1}{\delta}\right)\right) \geqslant 1 - \delta.$$

As described in the proof overview in §3.1.3.1, the core argument is nearly identical to the analysis of Greedy Osborne in §3.4. Below, we detail the additional probabilistic nuances and describe how to overcome them. Remaining details for the proof of Theorem 3.5.1 are deferred to §3.10.2.

■ 3.5.1 Bounding the number of iterations

Analogous to the proof of Greedy Osborne (c.f. Lemma 3.4.2), 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 3.5.2 (Potential decrease of Random Osborne)**.** *Consider any $x \in \mathbb{R}^n$ for which the corresponding scaling $A := \operatorname{diag}(e^x) K \operatorname{diag}(e^{-x})$ is not ε-balanced. If x' is the next iterate obtained from a Random Osborne update, then*

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

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 3.4.2), with only two minor differences. The first is that (3.19) and (3.20) are in expectation. The second is that (3.21) holds with equality by definition of the Random Osborne algorithm. □

Lemma 3.3.1 shows that the potential is initially bounded, and Lemma 3.5.2 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 (3.24) in §3.4). Lemma 3.5.3 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 3.5.3 (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 $Y_0 \leqslant A$ a.s., each difference $Y_{t-1} - Y_t$ is bounded within $[0, 2(A-a)]$ a.s., and*

$$\mathbb{E}\left[Y_t - Y_{t+1} \,|\, \mathcal{F}_t,\, Y_t \geqslant a\right] \geqslant h \tag{3.28}$$

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

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

The expectation bound in Lemma 3.5.3 is proved using Doob's Optional Stopping Theorem, and the h.p. bound using Chernoff bounds; details are deferred to §3.10.1.

Remark 3.5.4 (Sub-exponential concentration)**.** *Lemma 3.5.3 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 $\mathbb{P}(\tau \leqslant N) = 1 - \mathbb{P}(X_1 = \cdots = X_N = 0) = 1 - (1-h)^N$ which is $\approx 1 - \delta$ when $N \approx \frac{1}{h} \log \frac{1}{\delta}$.*

■ 3.5.2 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 3.1.3 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. §3.4). 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 $Z_1, \dots, Z_\tau$ equals $\mathbb{E}\,\tau\,\mathbb{E}\,Z_1$, so long as τ is independent from $Z_1, \dots, Z_\tau$ [81, 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 3.1.3 and Wald's equation.

Lemma 3.5.5 (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 3.5.6 (Minor modification of Wald's equation)**.** *Let $Z_1, Z_2, \dots$ be i.i.d. nonnegative integrable r.v.'s. Let τ be an integrable $\mathbb{N}$-valued r.v. satisfying $\mathbb{E}[Z_t | \tau \geqslant t] = \mathbb{E}[Z_1]$ 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 3.5.5 is nearly identical to the proof of Observation 3.1.3, and is thus omitted. The proof of Lemma 3.5.6 is a minor modification of the proof of the standard Wald's equation in [81]; details in §3.10.1.

■ 3.6 Random-Reshuffle Cyclic Osborne converges quickly

Here we show a runtime bound for Random-Reshuffle Cyclic Osborne. See §3.1.2 for further discussion, and §3.1.3.3 for a proof sketch.

Theorem 3.6.1 (Convergence of Random-Reshuffle Cyclic Osborne)**.** *Given a balanceable matrix $K \in \mathbb{R}^{n\times n}_{\geqslant 0}$ and accuracy $\varepsilon > 0$, Random-Reshuffle Cyclic Osborne solves $\texttt{ABAL}(K, \varepsilon)$ 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 $\delta > 0$,*
$$\mathbb{P}\left(T \leqslant c\left(\tfrac{mn}{\varepsilon}(\tfrac{1}{\varepsilon} \wedge d)\log\kappa \log\tfrac{1}{\delta}\right)\right) \geqslant 1 - \delta.$$

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

Lemma 3.6.2 (Potential decrease of Random-Reshuffle Cyclic Osborne)**.** *Consider any $x \in \mathbb{R}^n$ for which the corresponding scaling $A := \operatorname{diag}(e^x) K \operatorname{diag}(e^{-x})$ is not ε-balanced. Let x' be the iterate obtained from x after a cycle of Random-Reshuffle Cyclic Osborne. Then*
$$\mathbb{E}\left[\Phi(x) - \Phi(x')\right] \geqslant \frac{1}{4n}\left(\frac{\Phi(x) - \Phi^*}{d\log\kappa} \vee \varepsilon\right)^2,$$
where the expectation is over the algorithm's random choice of update coordinates.

Proof. By monotonicity of Φ w.r.t. Osborne updates (Lemma 3.3.2), 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 3.5.2. □

The runtime bound for Random-Reshuffle Cyclic Osborne (Theorem 3.6.1) given the expected per-cycle potential decrease (Lemma 3.6.2) then follows by an identical argument as the runtime bound for Random Osborne (Theorem 3.5.1) given that algorithm's expected per-iteration potential decrease (Lemma 3.5.2). The straightforward details are omitted for brevity.

■ 3.7 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 §3.2.5). See §3.1.2 for a discussion of these results, and §3.1.3.4 for a proof sketch.

Theorem 3.7.1 (Convergence of Block Osborne variants)**.** *Consider balancing a balanceable matrix $K \in \mathbb{R}^{n\times n}_{\geqslant 0}$ 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* $\texttt{ABAL}(K, \varepsilon)$ *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 3.7.1 have exponentially decaying tails, just as for the non-parallelized variants (c.f., Theorems 3.5.1 and 3.6.1; see also Remark 3.5.4).

The proof of Theorem 3.7.1 is nearly identical to the analysis of the analogous non-parallelized variants in §3.4, §3.5, and §3.6 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 3.4.2 and 3.5.2, respectively. Below we show this modification for Greedy Block Osborne; an identical argument applies for Random Block Osborne after taking an expectation (the inequality (3.29) then becomes an equality).

Lemma 3.7.2 (Potential decrease of Greedy Block Osborne)**.** *Consider any* $x \in \mathbb{R}^n$ *for which the corresponding scaling* $A := \operatorname{diag}(e^x) K \operatorname{diag}(e^{-x})$ *is not* ε*-balanced. If* x' *is the next iterate obtained from a Greedy Block Osborne update, then*

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

Proof. Let S_ℓ be the chosen block. Using in order Lemma 3.3.2, the inequality $-\log(1-z) \geqslant z$, the definition of Greedy Block Osborne, re-arranging, and then Proposition 3.3.3,

$$\begin{aligned}
\Phi(x) - \Phi(x') &= -\sum_{k \in S_\ell} \log(1 - \left(\sqrt{r_k(P)} - \sqrt{c_k(P)} \right)^2 \right) \\
&\geqslant \sum_{k \in S_\ell} \left(\sqrt{r_k(P)} - \sqrt{c_k(P)} \right)^2 \\
&\geqslant \frac{1}{p} \sum_{\ell=1}^{p} \sum_{k \in S_\ell} \left(\sqrt{r_\ell(P)} - \sqrt{c_\ell(P)} \right)^2 \qquad (3.29) \\
&= \frac{1}{p} \sum_{k=1}^{n} \left(\sqrt{r_k(P)} - \sqrt{c_k(P)} \right)^2 \\
&\geqslant \frac{1}{4p} \left(\frac{\Phi(x) - \Phi^*}{d \log \kappa} \vee \varepsilon \right)^2 .
\end{aligned}$$

□

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 3.7.1. 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 §3.6).

Next, we describe the total-work bounds in Theorem 3.7.1. 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 3.1.3. 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 3.7.1 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 3.7.3 (Parallelized Osborne for uniformly sparse matrices)**.** *There is a parallelized algorithm that, given any Δ-uniformly sparse matrix $K \in \mathbb{R}^{n\times n}_{\geqslant 0}$, computes an 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 [31] 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 3.7.1. □

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.

■ 3.8 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 [126, §3] and Lemma 3.3.5. For such iterates, operations on the current balancing $\operatorname{diag}(e^x)K\operatorname{diag}(e^{-x})$—namely, computing row and

[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.

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 $(i,j) \in \operatorname{supp}(K)$, see Remark 3.8.2.) 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 3.8.4—all inputs are represented using this many bits.

Theorem 3.8.1 (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 3.5.1 (respectively, Theorem 3.6.1, 3.7.1, and again 3.7.1).*

Moreover, if the matrix K is given as input through the logarithms of its entries $\{\log K_{ij}\}_{(i,j)\in\operatorname{supp}(K)}$, 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 [167, §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's algorithm for Matrix Scaling (see §3.10.4).

Before proving Theorem 3.8.1, we make several remarks.

Remark 3.8.2 (Log-domain input)**.** *Theorem 3.8.1 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 [18, §5].*

Remark 3.8.3 (Greedy Osborne requires large bit-complexity)**.** *All known implementations of Greedy Osborne require bit-complexity at least $\tilde{\Omega}(n)$ [167]. The*

[15] Note that Theorem 3.8.1 outputs only the balancing vector $x \in \mathbb{R}^n$, not the approximately balanced matrix $A = \operatorname{diag}(e^x) K \operatorname{diag}(e^{-x})$. 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 [18, §5.3].

obstacle is the computation (3.10) *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 3.8.7).*

We now turn to the proof of Theorem 3.8.1. 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 3.1. We emphasize that these modifications are in line with standard implementations of Osborne's algorithm in practice, see Remark 3.2.6.

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 3.8.7) 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 §3.10.3 for brevity.

Lemma 3.8.4 (Approximately balancing an approximate matrix suffices)**.** *Let $K, \tilde{K} \in \mathbb{R}^{n\times n}_{\geqslant 0}$ such that supp(K) = 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 3.8.5 (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 3.8.6 (Stability of potential function)**.** *Let $K \in \mathbb{R}^{n\times n}_{\geqslant 0}$. 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 3.8.7 (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.*

Theorem 3.8.1. **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 3.8.4, 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 3.8.6, 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 3.5.2) by at most a factor of $1/2$. The proof of Theorem 3.5.1 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 3.3.5, 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 3.8.7, the Osborne update requires bit-complexity $O(\log \frac{n}{\tau}) = O(\log \frac{n}{\varepsilon})$.

□

■ 3.9 Discussion

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 [167, 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 [167], and the runtime bound we show for Random-Reshuffle Cyclic Osborne scales as roughly mn (Theorem 3.6.1).

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 [139, 212, 213].

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 [198, §6] for Max-Balancing.)

3.10 Deferred details

3.10.1 Probabilistic helper lemmas

Several times we make use of the following standard (martingale) version of multiplicative Chernoff bounds, see, e.g., [154, §4].

Lemma 3.10.1 (Multiplicative Chernoff Bounds)**.** *Let $X_1, \dots X_n$ be supported in $[0,1]$, be adapted to some filtration $\mathcal{F}_0 = \{\emptyset, \Omega\}, \mathcal{F}_1, \dots, \mathcal{F}_n$, 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 $\Delta \in (0,1)$, $\mathbb{P}\left(X \leqslant (1-\Delta)\mu\right) \leqslant e^{-\Delta^2\mu/2}$.*
- *(Upper tail.) For any $\Delta \geqslant 1$, $\mathbb{P}\left(X \geqslant (1+\Delta)\mu\right) \leqslant e^{-\Delta\mu/3}$.*

Lemma 3.5.3. 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 [81] (which may be invoked by a.s. boundedness),

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

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

$$\begin{aligned}\mathbb{P}(\tau > N) &= \mathbb{P}(Y_N > a) \\ &= \mathbb{P}\left(\sum_{t=1}^{N}(Y_{t-1} - Y_t) < Y_0 - a\right) \\ &\leqslant \mathbb{P}\left(\sum_{t=1}^{N}(Y_{t-1} - Y_t) < A - a\right) \qquad (3.30)\end{aligned}$$

To bound (3.30), 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 3.10.1 (combined with a simple stochastic domination argument since $\mathbb{E}[X_t|\mathcal{F}_{t-1}] \geqslant h/B$ rather than exactly equal), and then the choice of N, we conclude that

$$\begin{aligned}\mathbb{P}\left(\sum_{t=1}^{N}(Y_{t-1} - Y_t) < A - a\right) &= \mathbb{P}\left(\sum_{t=1}^{N} X_t < \frac{A-a}{B}\right) \\ &\leqslant \exp\left(-\left(1 - \frac{A-a}{Nh}\right)^2 \frac{Nh}{2B}\right) \\ &\leqslant \delta. \qquad (3.31)\end{aligned}$$

□

Lemma 3.5.6. Observe that

$$\begin{aligned}\mathbb{E}\left[\sum_{t=1}^{\tau} Z_t\right] &= \sum_{T=1}^{\infty}\mathbb{E}\left[\sum_{t=1}^{\tau} Z_t \mathbb{1}_{\tau=T}\right] = \sum_{T=1}^{\infty}\sum_{t=1}^{T}\mathbb{E}\left[Z_t \mathbb{1}_{\tau=T}\right] \\ &= \sum_{t=1}^{\infty}\sum_{T=t}^{\infty}\mathbb{E}\left[Z_t \mathbb{1}_{\tau=T}\right] = \sum_{t=1}^{\infty}\mathbb{E}\left[Z_t \mathbb{1}_{\tau\geqslant t}\right],\end{aligned}$$

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

■ 3.10.2 Proof of Theorem 3.5.1

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 $\tau := \min\{t \in \mathbb{N}_0 :$

$\operatorname{diag}(e^x) K \operatorname{diag}(e^{-x})$ is ε-balanced$\}$. By Lemma 3.5.2,

$$\mathbb{E}\left[\Phi(x^{(t)}) - \Phi(x^{(t+1)}) \,|\, \mathcal{F}_t, t \leqslant \tau\right] \geqslant \frac{1}{4n}\left(\frac{\Phi(x^{(t)}) - \Phi^*}{d \log \kappa} \vee \varepsilon\right)^2. \tag{3.32}$$

Case 1: $\varepsilon^{-1} \leqslant d$**.** 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 $t \leqslant \tau$, and otherwise $\Phi(x^{(t)}) - (t - \tau)\,\varepsilon^2/4n$ if $t > \tau$. By (3.32), we have

$$\mathbb{E}\left[Y_t - Y_{t+1} \,|\, \mathcal{F}_t, Y_t \geqslant 0\right] \geqslant \frac{\varepsilon^2}{4n}. \tag{3.33}$$

For both expected and h.p. bounds below, we apply Lemma 3.5.3 to the process Y_t with $A = \log \kappa$ (by Lemma 3.3.1), $a = 0$, and $h = \varepsilon^2/4n$ (by (3.33)).

Expectation bound. The expectation bound in Lemma 3.5.3 implies $\mathbb{E}[\tau] \leqslant 4n\,\varepsilon^{-2} \log \kappa + 1$. Since each iteration has expected runtime $\mathbb{E}[T_t | \mathcal{F}_{t-1}] = O(m/n)$ by Lemma 3.5.5, Lemma 3.5.6 ensures that the total expected runtime is $\mathbb{E}\,T = \mathbb{E}[\sum_{t=1}^{\tau} T_t] = \mathbb{E}\,\tau\,\mathbb{E}\,T_1 = O(m\,\varepsilon^{-2} \log \kappa)$.

H.p. bound. For shortand, denote $U := 24n\,\varepsilon^{-2} \log \kappa \log \frac{2}{\delta}$. The h.p. bound in Lemma 3.3.1 implies that $\mathbb{P}(\tau > U) \leqslant \delta/2$. By Lemma 3.5.5, there is some constant $c > 0$ such that $\mathbb{E}[T_t] = cm/n$. Since the T_t are independent, a Chernoff bound (Lemma 3.10.1) implies that $\mathbb{P}(\sum_{t=1}^{U} T_t \leqslant 2cUm/n) \leqslant \delta/2$. 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: $\varepsilon^{-1} \geqslant d$**.** Here, we establish the $O(md\,\varepsilon^{-1} \log \kappa)$ runtime bound in expectation and w.h.p. Define $\alpha, \tau_1, \tau_2, \tau_{1,i}$, and ϕ_i as in the analysis of Greedy Osborne (see §3.4).

Expectation bound. To bound $\mathbb{E}\,\tau_2$, define Y_t and apply Lemma 3.5.3 as in case 1 above (except now with $A = \varepsilon\,d \log \kappa$) to establish that

$$\mathbb{E}\,\tau_2 \leqslant \frac{\varepsilon\,d \log \kappa}{\varepsilon^2/4n} + 1 = \frac{4nd \log \kappa}{\varepsilon} + 1. \tag{3.34}$$

Next, we bound $\mathbb{E}\,\tau_1$. Consider subphase $\tau_{1,i}$ for $i \in [N]$. By an application of Lemma 3.5.3 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 (3.32), $\mathbb{E}\,\tau_{1,i} \leqslant \frac{4nd^2 \log^2 \kappa}{\phi_i} + 1$. Thus $\mathbb{E}\,\tau_1 = \sum_{i=1}^{N} \mathbb{E}\,\tau_{1,i} \leqslant 4nd^2 \log^2 \kappa (\sum_{i=1}^{N} \frac{1}{\phi_i}) + N$. Since $\sum_{i=1}^{N} \frac{1}{\phi_i} \leqslant \frac{4}{\varepsilon\,d \log \kappa}$,

$$\mathbb{E}\,\tau_1 \leqslant \frac{16nd \log \kappa}{\varepsilon} + \log_2 \left\lceil \frac{1}{\varepsilon\,d} \right\rceil. \tag{3.35}$$

Combining (3.34) and (3.35) establishes that $\mathbb{E}\,\tau = \mathbb{E}\,\tau_1 + \mathbb{E}\,\tau_2 \leqslant 21nd\,\varepsilon^{-1}\log\kappa$. By the $O(m/n)$ per-iteration expected runtime bound in Lemma 3.5.5 and the variant of Wald's equation in Lemma 3.5.6, the total expected runtime is therefore at most $\mathbb{E}\,T \leqslant O(m/n)\cdot\mathbb{E}\,\tau = O(md\,\varepsilon^{-1}\log\kappa)$.

H.p. bound. By Lemma 3.5.3, $\mathbb{P}(\tau_2 > 24nd\,\varepsilon^{-1}\log\kappa\log\frac{4}{\delta}) \leqslant \delta/4$. To bound the first phase, define $p_i := \delta/2^{N-i+3}$ for each $i \in [N]$. By Lemma 3.5.3, $\mathbb{P}(\tau_{1,i} > (24nd^2\log^2\kappa\log 1/p_i)/\phi_i) \leqslant p_i$. Note that $\sum_{i=1}^N \frac{\log 1/p_i}{\phi_i} = \frac{1}{\phi_N}\sum_{j=0}^{N-1} 2^{-j}(\log 8/\delta + j\log 2) \leqslant \frac{1}{\phi_N}\sum_{j=0}^{\infty} 2^{-j}(\log 8/\delta + j\log 2) = \frac{2\log 8/\delta + 2\log 2}{\phi_N} \leqslant \frac{6\log 8/\delta}{\varepsilon\, d\log\kappa}$. Thus by a union bound, with probability at most $\sum_{i=1}^N p_i \leqslant \delta/4$, the first phase has length at most $\tau_1 = \sum_{i=1}^N \tau_{1,i} \leqslant 144nd\,\varepsilon^{-1}\log\kappa\log\frac{8}{\delta}$. We conclude by a further union bound that, with probability at least $1-\delta/2$, the total number of iterations is at most $\tau = \tau_1 + \tau_2 \leqslant 168nd\,\varepsilon^{-1}\log\kappa\log\frac{8}{\delta}$. The proof is complete by an identical Chernoff bound argument as in case 1 above.

■ 3.10.3 Proofs for §3.8

Lemma 3.8.4. Let $A := \operatorname{diag}(e^x)K\operatorname{diag}(e^{-x})$ 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 $|r_k(P) - r_k(\tilde{P})| \leqslant 3\gamma$, and similarly for the columns. Thus by the triangle inequality, $\|r(P) - c(P)\|_1 \leqslant \|r(\tilde{P}) - c(\tilde{P})\|_1 + 6n\gamma$. □

Lemma 3.8.5. Let $x, y \in \mathbb{R}^n$. Since $\min_i(a_i/b_i) \leqslant (\sum_{i=1}^n a_i)/(\sum_{i=1}^n b_i) \leqslant \max_i(a_i/b_i)$ for any $a, b \in \mathbb{R}^n_{>0}$,

$$\log\sum_{i=1}^n e^{x_i} - \log\sum_{i=1}^n e^{y_i} = \log\frac{\sum_{i=1}^n e^{x_i}}{\sum_{i=1}^n e^{y_i}} \leqslant \log\max_i e^{x_i-y_i} = \max_i x_i - y_i \leqslant \|x-y\|_\infty,$$

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

Lemma 3.8.6. Let $x, y \in \mathbb{R}^n$. Clearly $|(x_i - x_j) - (y_i - y_j)| \leqslant 2\|x-y\|_\infty$ for any $i, j \in [n]$. Thus by Lemma 3.8.5, $|\Phi(x) - \Phi(y)| = |\log(\sum_{(i,j)\in\operatorname{supp}(K)} e^{x_i - x_j + \log K_{ij}}) - \log(\sum_{(i,j)\in\operatorname{supp}(K)} e^{y_i - y_j + \log K_{ij}})| \leqslant 2\|x-y\|_\infty$. □

Lemma 3.8.7. 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 $z_i \leqslant 0$ 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 3.8.5, 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 $k \leqslant n$, 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 $e^{\tilde{z}_i} \geqslant O(\tau/n)$, 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. □

■ 3.10.4 Connections to Matrix Scaling and Sinkhorn's algorithm

Here, we continue the discussion in Remark 3.1.6 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 §3.2.5). For the Matrix Scaling problem on $K \in \mathbb{R}_{\geqslant 0}^{m \times n}$, 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 3.8.1, 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 the discussion after Theorem 2.2.1. This modification reduces the bit-complexity to only logarithmic size $O(\log(Ln/\varepsilon))$.

Chapter 4

Approximating Min-Mean-Cycle for low-diameter graphs in near-optimal time and memory

We revisit Min-Mean-Cycle, the classical problem of finding a cycle in a weighted directed graph with minimum mean weight. Despite an extensive algorithmic literature, previous work failed to achieve a near-linear runtime in the number of edges m. We propose an algorithm with near-linear runtime $\tilde{O}(m(W_{\max}/\varepsilon)^2)$ for computing an ε additive approximation on graphs with polylogarithmic diameter and weights of magnitude at most $W_{\max}$. In particular, this is the first algorithm whose runtime scales in the number of vertices n as $\tilde{O}(n^2)$ for the complete graph. Moreover—unconditionally on the diameter—the algorithm uses only $O(n)$ memory beyond reading the input, making it "memory-optimal". Our approach is based on solving a linear programming relaxation using entropic regularization, which reduces Min-Mean-Cycle to a Matrix Balancing problem—á la the popular reduction of Optimal Transport to Matrix Scaling. We round the fractional linear program solution using a variant of the classical Cycle-Cancelling algorithm that is sped up to near-linear runtime at the expense of being approximate, and implemented in a memory-optimal manner. The algorithm is simple to implement and is competitive with the state-of-the-art methods in practice.

4.1 Introduction

Let $G = (V, E, w)$ be a weighted directed graph (digraph) with vertices V, directed edges $E \subseteq V \times V$, and edge weights $w : E \to \mathbb{R}$. The *mean weight* of a cycle σ is the arithmetic mean of the weights of the cycle's constituent edges, denoted $\bar{w}(\sigma) := \frac{1}{|\sigma|} \sum_{e \in \sigma} w(e)$. The *Min-Mean-Cycle* problem (MMC for short) is to find

a cycle of minimum mean weight. The corresponding value is denoted

$$\mu(G) := \min_{\text{cycle } \sigma \text{ in G}} \bar{w}(\sigma). \tag{MMC}$$

Over the past half century, MMC has received significant attention due to its numerous fundamental applications in periodic optimization, algorithm design, and max-plus algebra. Applications in periodic optimization include deterministic Markov Decision Processes and mean-payoff games [243], financial arbitrage [69], cyclic scheduling problems [131], and performance analysis of digital systems [76], among many others. In algorithm design, MMC provides a tractable option for the bottleneck step in the network simplex algorithm. This has led to the use of MMC in algorithms for several graph theory problems [5, 169]—including, notably, a strongly polynomial algorithm for the Minimum Cost Circulation problem, which includes Maximum Flow as a special case [96]. In max-plus algebra, which commonly arises in operations research and control theory problems, MMC characterizes the fundamental spectral theoretic quantities [30, 103]. More recently, MMC has also arisen in control theory since it captures the growth rate of switched linear dynamical systems with rank-one updates [4, 15].

These myriad applications have motivated a long line of algorithmic work with the goal of solving MMC efficiently. Remarkably, MMC is solvable in polynomial time [130], despite the fact that many seemingly similar optimization problems over cycles are not. Indeed, in sharp contrast, the problem of finding the cycle with minimum *total* weight $\sum_{e\in\sigma} w(e)$ is NP-complete since it can encode the Hamiltonian Cycle problem [196, §8.6b].

Algorithmic advancements over the past half century have led to many efficient algorithms for MMC; details in the prior work section §4.1.3 below. However, previous work falls short of a near-linear runtime in the input sparsity $m := |E|$. For instance, even in the "simple" case where the edge weights are in $\{-1, 0, 1\}$, the best known runtimes are $O(m\sqrt{n}\log n)$ from [165], $m^{11/8+o(1)}$ implicit from [26], and $O(n^\omega \text{polylog} n)$ implicit from [193, 240], where $n := |V|$ is the number of vertices, and $\omega \approx 2.37$ is the current matrix multiplication exponent [231]. These runtimes are incomparable in the sense that which is fastest depends on the graph sparsity (i.e., the ratio of m to n). Nevertheless, in all parameter settings, these runtimes are far from linear in m. An important algorithmic barrier is that *any* faster runtime—let alone a linear runtime—for solving a natural LP relaxation of MMC would constitute a major breakthrough in algorithmic graph theory, as it would imply faster algorithms for many well-studied problems (e.g., Shortest Paths with negative weights [196, §8.2]).

A primary motivation of this chapter is the observation that this complexity barrier is only for *exactly* computing (this LP relaxation of) MMC. Indeed,

our main result is that for graphs with polylogarithmic diameter, MMC can be *approximated* in near-linear[1] time.

4.1.1 Contributions

Henceforth, G is assumed strongly connected; this is without loss of generality for MMC after a trivial $O(m)$ pre-processing step; see §4.2. We denote the *unweighted* diameter of G by d. The notation $\tilde{O}(\cdot)$ suppresses polylogarithmic factors in the number of vertices n, the inverse accuracy ε^{-1}, and the maximum modulus edge weight $w_{\max}$.

We give the first approximation algorithm for MMC that, for graphs with polylogarithmic diameter, has near-linear runtime in the input sparsity m. In particular, this is the first near-linear time algorithm for the important special cases of complete graphs, expander graphs, and random graphs. (Note also that if the diameter is larger than polylogarithmic, this runtime can still be much faster than the state-of-the-art, depending on the parameter regime.) Moreover, unconditionally on the diameter, this new algorithm requires only $O(n)$ additional memory beyond reading the input[2], which means it is so-called "memory-optimal" in the sense that its memory usage is of the same order as the (maximum possible) output size.

Theorem 4.1.1 (Informal version of Theorem 4.6.2)**.** *There is a randomized algorithm (AMMC, see Algorithm 4.4) that given a weighted digraph $G = (V, E, w)$ and an accuracy $\varepsilon > 0$, finds a cycle σ in G satisfying $\bar{w}(\sigma) \leqslant \mu(G) + \varepsilon$ using $O(n)$ memory beyond reading the input and $O(md^2(\frac{w_{\max}}{\varepsilon})^2 \log n)$ time, both in expectation and with exponentially high probability.*

This algorithm AMMC is based on approximately solving an entropically regularized version of an LP relaxation of MMC, followed by rounding the obtained fractional LP solution using a fast, approximate version of the classical Cycle-Cancelling algorithm; details in the overview section §4.1.2. The entropic regularization approach has two key benefits. First, it effectively reduces the optimization problem to Matrix Balancing—a well-studied problem in scientific computing for which near-linear time algorithms were recently developed [9, 17, 67, 167] (see Chapter 3). At a high-level, this parallels the popular entropic-regularization reduction of Optimal Transport to Matrix Scaling [73, 232] (see Chapter 2). Second,

[1] 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} and the maximum modulus edge weight $w_{\max}$.

[2] Storing the input graph takes $\Theta(m)$ memory. To design an algorithm with $o(m)$ memory, we assume G is input implicitly through two oracles: one for finding an adjacent edge of a vertex, and one for querying the weight of an edge; details in §4.6.1.2.

it enables a compact $O(n)$-size implicit representation of the (naïvely $O(m)$-size) fractional solution to the LP relaxation.

Discussion:

Practicality. `AMMC` is practical and simple to implement. This is in contrast to the aforementioned state-of-the-art theoretical algorithms, which rely on (currently) impractical subroutines such as Fast Matrix Multiplication or fast Laplacian solvers, and/or have large constants in their runtimes which can be prohibitive in practice; see the experimental surveys [58, 75, 76, 94, 116]. Indeed, there is currently a large discrepancy between the state-of-the-art MMC algorithms in theory and in practice: the aforementioned empirical surveys point out that the algorithms with best empirical performance have worst-case runtimes no better than $\Omega(mn)$. In §4.7, we provide preliminary numerical simulations demonstrating that in practice, `AMMC` can compute high-quality solutions in essentially $O(m)$ linear runtime and for larger problem sizes than the state-of-the-art algorithms implemented in the popular, heavily-optimized C++ software package LEMON [79].

Multiplicative approximation. If all edge weights are positive, then the *additive* approximation of `AMMC` also yields a *multiplicative* approximation. (If the edge weights are not all positive, then it is impossible to compute any multiplicative approximation in near-linear time, barring a major breakthrough in algorithmic graph theory, namely faster algorithms for the classical Negative Cycle Detection problem [59, §1.2].) Specifically, if all edge weights lie in $[w_{\min}, w_{\max}]$ for $w_{\min} > 0$, then we can find a cycle σ satisfying $\bar{w}(\sigma) \leqslant (1+\varepsilon)\mu(G)$ in $O(md^2(\frac{w_{\max}}{\varepsilon\, w_{\min}})^2 \log n)$ time since $\mu(G) \geqslant w_{\min}$.

Weighted vs unweighted diameter. For simplicity, our runtime is written in terms of the unweighted diameter d. However, $w_{\max}d$ can be replaced by the weighted diameter of the graph with weights $w(e) - w_{\min}$ which are translated to be all nonnegative.[3] This yields tighter bounds since this weighted diameter is at most d times the weight range.

Implications. Our improved approximation algorithm for MMC immediately implies similarly improved algorithms for several related problems. For instance,

[3]This weighted diameter is a natural quantity since it is invariant under the simultaneous translation of all edge weights—a transformation which does not change the complexity of (additively approximating) MMC. To get such bounds, the only change to our algorithms is to compute Single Source Shortest Paths using these translated weights (rather than unit weights), which can be done in near-linear time since they are nonnegative.

the Min-GeoMean-Cycle problem—in which weights are strictly positive, and we seek a cycle σ minimizing $(\prod_{e\in\sigma} w(e))^{1/|\sigma|}$—can be multiplicatively approximated by using our algorithms to additively approximate MMC with weights $\tilde{w}(e) := \log w(e)$. Another immediate implication is the first near-linear time algorithm (again assuming moderate connectedness) for approximating fundamental quantities in max-plus spectral theory. Specifically, let A be an $n \times n$ matrix with entries in $\mathbb{R}_{\max} = \mathbb{R} \cup \{-\infty\}$. It is known that the max-plus eigenvalues and the cycle-time vector of A are characterizeable in terms of the Min-Mean-Cycles of the strongly connected components of the associated digraph $G = (\{1, \ldots, n\}, \{(i, j) : A_{ij} \neq -\infty\})$, see, e.g., [30, 103]. Thus, after topologically sorting the components of G in linear time, we can compute both the max-plus spectrum and the cycle-time vector of A to ℓ_∞ error ε in $\tilde{O}(md^2(\frac{w_{\max}}{\varepsilon})^2)$ time, where $w_{\max} := \max_{ij:A_{ij}\neq-\infty} |A_{ij}|$ and d denotes the diameter of G.

4.1.2 Approach

In contrast to previous combinatorial approaches for MMC, we tackle this discrete problem via *continuous* optimization techniques. At a high level, we follow a standard template for approximation algorithms that consists of two steps: approximately solve a linear programming (LP) relaxation; then round the fractional solution to a vertex without worsening the LP cost by much. While this high-level template is standard, implementing it efficiently for MMC poses several obstacles. In particular, both steps require new specialized algorithms since out-of-the-box LP solvers and rounding algorithms are too slow for our desired runtime. Moreover, our goal of designing a memory-optimal algorithm restricts memory usage to being sublinear in the graph size, thereby precluding many natural approaches.

Our starting point is the classical LP relaxation of MMC

$$\min_{F\in\mathcal{F}_E} \sum_{e\in E} F(e)w(e), \tag{MMC-P}$$

where above the decision set $\mathcal{F}_E$ is the polytope consisting of circulations on G that are normalized to have unit total flow. Details on this LP are in the preliminaries section §4.2.

Step 1: optimization. This is the main step of the algorithm—both conceptually and technically. In it, we find a near-optimal solution for (MMC-P). We do this by employing *entropic regularization*, a celebrated technique for regularizing optimization problems over probability distributions. This is motivated by viewing

the normalized circulations in $\mathcal{F}_E$ as probability distributions on the edges of G (see Remark 4.2.1). The key insight is that entropically regularizing (MMC-P) results in a convex optimization problem that corresponds to an associated *Matrix Balancing* problem. This effectively reduces approximating (MMC-P) to a problem for which near-linear time algorithms were recently developed [9, 17, 67, 167]. In particular, we employ a randomized[4] version of Osborne's algorithm for Matrix Balancing which is practical and provably runs in near-linear time, as shown in Chapter 3. A further benefit of our reduction is that Matrix Balancing can be performed in a memory-optimal way, yielding a fractional solution for (MMC-P) that is compactly represented using $O(n)$ memory despite having m nonzero entries. See §4.4 for details and for natural dual interpretations of the regularization and algorithm.

Step 2: rounding. Step 1 outputs a near-feasible circulation (since Matrix Balancing can only be performed approximately) with near-optimal objective for (MMC-P). In this step, we compute from this a near-optimal cycle for MMC. We perform this in two sub-steps.

First, we correct feasibility without changing much flow, thereby preserving near-optimality. We do this by re-routing flow from vertices with flow surplus to vertices with flow deficiency via short paths. While a naïve implementation of this requires $O(mn)$ time and $O(nd)$ memory, there is a simple trick that enables implementing this in near-linear time and in a memory-optimal way: route all these paths through an arbitrary vertex. Details in §4.5.1.

Second, we round the resulting near-optimal circulation (a fractional point in $\mathcal{F}_E$) to a cycle (a vertex of $\mathcal{F}_E$) while preserving the objective of (MMC-P). The Cycle-Cancelling algorithm [196] does this by decomposing the circulation into a convex combination of cycles, and then outputting the best cycle. However, it has a prohibitive $O(mn)$ runtime. Since we can tolerate ε error, a Ford-Fulkerson-esque argument enables us to speed up this algorithm to near-linear time by simply running it on a quantization of the circulation. Details in §4.5.2.

4.1.3 Prior work

4.1.3.1 Exact algorithms

There is an extensive literature on MMC algorithms; Table 4.1 summarizes the fastest known runtimes. These runtimes are incomparable in that each is best for a certain parameter regime. The fastest algorithm for very large edge weights is the

[4]This is the only source of randomness in our proposed algorithm.

$O(mn)$ dynamic-programming algorithm of [130].[5] For more moderate weights (e.g., integers of polynomial size in n), the $O(m\sqrt{n}\log(nw_{\max}))$ scaling-based algorithm of [165] is faster. Faster runtimes for certain parameter regimes are implicit from recent algorithmic developments for Single Source Shortest Paths (SSSP). The connection is that SSSP algorithms can detect negative cycles, and MMC on an integer-weighted graph is reducible to detecting negative cycles on $O(\log(nw_{\max}))$ graphs with modified edge weights [138]. This results in an $O(n^\omega w_{\max}\log(nw_{\max}))$ runtime which is faster for dense graphs with small weights [193, 240], and an $m^{11/8+o(1)}\log^2 w_{\max}$ runtime which is faster for sparse graphs with moderate weights [26].

Author	Runtime	Memory
Karp (1978) [130]	$O(mn)$	$O(n^2)$
Orlin and Ahuja (1992) [165]	$\tilde{O}(m\sqrt{n})$	$O(n)$
Sankowski (2005) [193], Yuster and Zwick (2005) [240]	$\tilde{O}(n^\omega)$	$O(n^2)$
Axiotis et al. (2020) [26]	$m^{11/8+o(1)}$	$O(m)$

Table 4.1: Fastest runtimes for exact MMC computation. The memory reported is the additional storage beyond reading the input (see §4.6.1.2). For simplicity, here edge weights are in $\{-1, 0, 1\}$; see the main text for detailed dependence on $w_{\max}$.

■ 4.1.3.2 Approximation algorithms

Table 4.2 lists the fastest approximation algorithms for MMC. The fastest existing approximation algorithm is the $\tilde{O}(n^\omega/\delta)$ algorithm of [59] for approximating MMC to a $(1\pm\delta)$ multiplicative factor, in the special case of nonnegative integer weights. By taking $\delta = O(\varepsilon/w_{\max})$, this can be converted into an $\pm\varepsilon$ additive approximation algorithm with runtime $\tilde{O}(n^\omega w_{\max}/\varepsilon)$. This runtime is only faster than the exact algorithms of [193, 240] by a factor of $\tilde{O}(1/\varepsilon)$, which provides significant runtime gains only when the approximation accuracy ε is quite large.

Author	Runtime	Memory
Chatterjee et al. (2014) [59]	$\tilde{O}(n^\omega/\varepsilon)$	$O(n^2)$
This chapter (Theorem 4.6.2)	$\tilde{O}(md^2/\varepsilon^2)$	$O(n)$

Table 4.2: Fastest runtimes for approximating MMC to ε additive accuracy. The memory reported is the additional storage beyond reading the input (see §4.6.1.2). For simplicity, here edge weights are in $\{-1, 0, 1\}$; see the main text for detailed dependence on $w_{\max}$.

[5] The algorithms of [164, 238] have similar worst-case runtimes but better best-case and empirical runtimes.

We also mention Howard's policy-iteration algorithm [117]. Although the fastest known theoretical runtime for it is slower[6] than other algorithms, it is often used in practice because its empirical performance significantly outperforms its theoretical runtime [66, 75, 76]. On the other hand, the practical runtime of Howard's algorithm is observed to be at least $\Omega(mn)$ rather than near-linear when run on "difficult" inputs [75, 94], see also Figure 4.2.

Remark 4.1.2 (Alternative approach)**.** *An alternative algorithm that uses the same rounding subroutine as* ***AMMC****, but instead uses area-convexity regularization for the optimization subroutine, yields a slightly faster theoretical runtime of* $\tilde{O}(mdw_{\max}/\varepsilon)$*. The tradeoff is that unlike* ***AMMC****, this algorithm is not memory-optimal and performs poorly in practice. For details, see the extended version of the paper upon which this chapter is based [16].*

4.1.4 Simultaneous work

This chapter is based on the paper [18]. After v1 of this paper [18] was posted to arXiv, the paper [222] appeared on arXiv (and has since appeared in FOCS [223]). That paper [222] provides a breakthrough for solving a number of graph problems (including MMC) in near-linear time for graphs that are sufficiently dense $m = \tilde{\Omega}(n^{1.5})$. We mention the tradeoffs between this MMC algorithm and ours. On one hand, their algorithm can compute exact solutions whereas ours can only compute approximations with moderate accuracy. On the other hand, (1) their algorithm relies on Laplacian solvers for which implementations are currently impractical [116]; (2) our algorithm is memory-optimal and uses $O(n)$ memory, compared to the $\Omega(m)$ used by theirs; and (3) our algorithm still has near-linear runtime for sparse graphs $m = o(n^{1.5})$ with small diameter.

4.1.5 Roadmap

§4.2 recalls preliminaries. §4.3 details the two steps in our approach—optimize and round—which we implement efficiently in §4.4 and §4.5, respectively. §4.6 puts these pieces together to conclude our algorithm. §4.7 provides preliminary numerical simulations.

4.2 Preliminaries

Throughout, we assume that G is *strongly connected*, i.e., that there is a directed path from every vertex to every other. This is without loss of generality since

[6] Namely, $O(mn^3 w_{\max}/\varepsilon)$ for approximating MMC to ε additive accuracy if stopped early [75, Theorem 3.5].

we can decompose a general graph G into its strongly connected components in linear time [215], and then solve MMC on G by solving MMC on each component.

For simplicity, we assume each input edge weight is represented using an $\tilde{O}(1)$-bit number. This is essentially without loss of generality since after translating the weights and truncating them to $\pm\,\varepsilon$ additive accuracy—which does not change the problem of additively approximating MMC—all weights are representable using $O(\log(w_{\max}/\,\varepsilon)) = \tilde{O}(1)$-bit numbers.

In the sequel, we make use of a simple folklore algorithm for approximating the unweighted diameter d to within a factor of 2 in $O(m)$ time. This algorithm, called `ADIAM`, runs Breadth First Search to and from some vertex v, and returns the sum of the maximum distance found to and from v. It is straightforward to show that the output $\tilde{d}$ satisfies $d \leqslant \tilde{d} \leqslant 2d$. Efficiently computing better approximations is an active research area, but this suffices for our purposes.

4.2.1 Notation

Throughout, we reserve G for the graph, V for its vertex set, E for its edge set, w for its edge weights, $n = |V|$ for its number of vertices, $m = |E|$ for its number of edges, and d for its *unweighted* diameter (i.e., the maximum over $u, v \in V$ of the shortest unweighted path from u to v). For a positive integer n, we denote the set $\{1, \ldots, n\}$ by $[n]$. Any summation or maximum over indices i or j with unspecified limits ranges over $i, j \in [n]$.

Linear algebraic notation. Although this chapter targets graph theoretic problems, it is often helpful—both for intuition and conciseness—to express things using linear algebraic notation. For a weighted digraph $G = (V, E, w)$, we write W to denote the $n \times n$ matrix with ij-th entry $w(i, j)$ if $(i, j) \in E$, and ∞ otherwise. The support of a matrix A is $\mathrm{supp}(A) := \{(i, j) : A_{ij} \neq 0\}$. We write $\mathbf{0}$ and $\mathbf{1}$ to denote the all-zeros and all-ones vectors, respectively, in an ambient dimension clear from context (typically $\mathbb{R}^n$). For a vector $v \in \mathbb{R}^n$, we denote its ℓ_1 norm by $\|v\|_1 := \sum_i |v_i|$, its ℓ_∞ norm by $\|v\|_\infty = \max_i |v_i|$, its entrywise exponentiation by $\exp[v]$, and its diagonalization by $\mathrm{diag}(v) \in \mathbb{R}^{n \times n}$. For a matrix A, we denote the ℓ_1 norm of its vectorization by $\|A\|_1 := \sum_{ij} |A_{ij}|$, and its *entrywise* exponentiation by $\exp[A]$.

Flows and circulations. A *flow* on a digraph $G = (V, E)$ is a function $f : E \to \mathbb{R}_{\geqslant 0}$. Equivalently, in linear algebraic notation, this is a matrix $F \in \mathbb{R}^{n \times n}_{\geqslant 0}$ with $\mathrm{supp}(F) \subseteq E$. The corresponding *inflow*, *outflow*, and *netflow* for a vertex $i \in V$ are respectively $\sum_{(j,i) \in E} f(j, i)$, $\sum_{(i,j) \in E} f(i, j)$, and $\sum_{(j,i) \in E} f(j, i) - \sum_{(i,j) \in E} f(i, j)$; or in linear algebraic notation $(F^T \mathbf{1})_i$, $(F\mathbf{1})_i$, and $(F^T \mathbf{1} - F\mathbf{1})_i$. A flow is *balanced* at a vertex if that vertex has 0 netflow. A *circulation* is a flow

that is balanced at each vertex. The *total netflow imbalance* of a flow F is denoted $\delta(F) := \|F\mathbf{1} - F^T\mathbf{1}\|_1$. A flow or circulation is *normalized* if $\sum_{(i,j)\in E} f(i,j) = 1$.

Probability distributions. The set of discrete distributions on k atoms is associated with the k-simplex $\Delta_k := \{v \in \mathbb{R}^k_{\geqslant 0} : \sum_i v_i = 1\}$, the set of joint distributions on $V \times V$ with $\Delta_{n\times n} := \{P \in \mathbb{R}^{n\times n}_{\geqslant 0} : \sum_{ij} P_{ij} = 1\}$, and the set of distributions on E with $\Delta_E := \{P \in \Delta_{n\times n} : \mathrm{supp}(P) \subseteq E\}$.

4.2.2 LP relaxations of Min-Mean-Cycle

Here we recall the classical primal/dual pair of LP relaxations of MMC. Consider a weighted digraph $G = (V, E, w)$. Associate to each cycle σ an $n \times n$ matrix F_σ with ij-th entry equal to $1/|\sigma|$ if $(i,j) \in \sigma$, and 0 otherwise. Then MMC can be formulated as $\mu(G) = \min_{\text{cycle } \sigma} \langle F_\sigma, W\rangle$, where the inner product $\langle F_\sigma, W\rangle := \sum_{(i,j)\in E}(F_\sigma)_{ij}W_{ij}$ ranges over the edges of G. The LP relaxation of this discrete problem is

$$\min_{F\in\mathcal{F}_E} \langle F, W\rangle, \tag{MMC-P}$$

where $\mathcal{F}_E$ is the convex hull of $\{F_\sigma : \sigma \text{ cycle}\}$. It is well-known (e.g., [6, Problem 5.47]) that

$$\mathcal{F}_E = \{F \in \Delta_E : F\mathbf{1} = F^T\mathbf{1}\}.$$

Remark 4.2.1 (Interpretations of $\mathcal{F}_E$)**.** *From a graph theoretic perspective, $\mathcal{F}_E$ is the set of normalized circulations on G; and from a probabilistic perspective, $\mathcal{F}_E$ is the set of* joint distributions on the edge set $E \subseteq V \times V$ with identical marginal distributions. *There are also natural interpretations of the ℓ_1 distance $\|F\mathbf{1} - F^T\mathbf{1}\|_1$ of a matrix $F \in \Delta_E$ from $\mathcal{F}_E$: from a graph theoretic perspective, it is the* total netflow imbalance*; and from a probabilistic perspective, it is (two times) the* total variation distance between the marginals.

Throughout, we call (MMC-P) the primal LP relaxation. We refer to the dual of (MMC-P) as the dual LP relaxation. This is the LP $\max_{p\in\mathbb{R}^n,\lambda\in\mathbb{R}\,:\,\lambda\leqslant W_{ij}+p_i-p_j,\,\forall(i,j)\in E}\lambda$, but in the sequel it is helpful to re-write it in the following saddle-point form:

$$\max_{p\in\mathbb{R}^n}\min_{(i,j)\in E} W_{ij} + p_i - p_j. \tag{MMC-D}$$

4.3 Algorithmic framework

Here we detail the algorithmic framework we use for approximating MMC. As overviewed in §4.1.2, the framework consists of two steps: approximately solve

the LP relaxation (MMC-P), and then round this fractional solution to a vertex with nearly as good value for (MMC-P). While the optimization step is sufficient for estimating the *value* $\mu(G)$ of MMC, the rounding step yields a feasible *solution* (i.e., a cycle).

Algorithm 4.1 summarizes the accuracy required of each step. Note that the optimization step produces a near-optimal solution that is not necessarily feasible, but rather *near-feasible* in that we allow a slightly imbalanced netflow $\delta(P) = \|P\mathbf{1} - P^T\mathbf{1}\|_1$ up to some $\delta > 0$; in the sequel, we take $\delta = \Theta(\varepsilon/(w_{\max}d))$. Our rounding step accounts for this near-feasibility.

Input: Weighted digraph $G = (V, E, w)$, accuracy $\varepsilon > 0$
Output: Cycle σ in G satisfying $\bar{w}(\sigma) \leqslant \mu(G) + \varepsilon$

\\ Optimization step: compute near-feasible, near-optimal solution P for (MMC-P)
1: Find matrix $P \in \Delta_E$ satisfying $\delta(P) \leqslant \delta$ and $\langle P, W\rangle \leqslant \mu(G) + \frac{\varepsilon}{2}$

\\ Rounding step: round P to a vertex of $\mathcal{F}_E$ with nearly as good cost for (MMC-P)
2: Find cycle σ satisfying $\bar{w}(\sigma) \leqslant \langle P, W\rangle + \frac{\varepsilon}{4} + \frac{\varepsilon\,\delta(P)}{4\delta}$

Algorithm 4.1: Algorithmic framework for approximating MMC.

Observation 4.3.1 (Approximation guarantee for Algorithm 4.1)**.** *Given any weighted digraph G and any accuracy $\varepsilon > 0$, Algorithm 4.1 outputs a cycle σ in G satisfying $\bar{w}(\sigma) \leqslant \mu(G) + \varepsilon$.*

The proof is immediate by definition of the algorithmic framework. The obstacle is how to efficiently implement the two steps. This is shown in the following two sections.

■ 4.4 Efficient optimization of the LP relaxation

Here, we use Matrix Balancing to efficiently implement the optimization in the framework described in §4.3. Below, §4.4.1 describes the connections between MMC and Matrix Balancing, and §4.4.2 makes this algorithmic.

Some preliminary definitions about Matrix Balancing for this section. These definitions as well as more background are in Chapter 3, but are recalled here for the convenience of the reader. A matrix $A \in \mathbb{R}^{n\times n}_{\geqslant 0}$ is *balanced* if $A\mathbf{1} = A^T\mathbf{1}$. The *Matrix Balancing problem* for input $K \in \mathbb{R}^{n\times n}_{\geqslant 0}$ is to find a positive diagonal matrix D (if one exists) such that $A = DKD^{-1}$ is balanced.[7] K is *balanceable* if

[7]Technically, this is the problem of Matrix Balancing in the ℓ_1 norm, since the goal is to match the ℓ_1 norm of the rows and columns of A. However, we simply call this task "Matrix

	Primal		Dual	
Min-Mean-Cycle	$\min_{F \in \mathcal{F}_E} \langle F, W \rangle$	(MMC-P)	$\max_{p \in \mathbb{R}^n} \min_{ij} W_{ij} + p_i - p_j$	(MMC-D)
Matrix Balancing	$\min_{F \in \mathcal{F}_E} \langle F, W \rangle - \eta^{-1} H(F)$	(MB-P)	$\max_{p \in \mathbb{R}^n} \operatorname{smin}_{\eta, ij} W_{ij} + p_i - p_j$	(MB-D)

Table 4.3: Primal/dual LP relaxations of MMC (top), and our proposed regularizations (bottom). The regularized problems (MB-P) and (MB-D) are dual convex programs, with (essentially) unique solutions corresponding to balancing $K = \exp[-\eta W]$.

such a solution D exists (see Remark 4.4.4). The notion of *approximate Matrix Balancing* is introduced later in §4.4.2.

4.4.1 Connection to Matrix Balancing

The key connection is that appropriately regularizing the LP relaxation of MMC results in a convex optimization problem that is equivalent to an associated Matrix Balancing problem. This regularization can be equivalently performed on either the primal or dual LP (see Table 4.3); we describe both perspectives as they give complementary insights. We note that while these regularized problems are well-known to be connected to Matrix Balancing (e.g., [85, 126]), the relation of MMC to these regularized problems and Matrix Balancing is, to our knowledge, not previously known.

4.4.1.1 Primal regularization

In the primal, we employ *entropic regularization*: we subtract η^{-1} times the Shannon entropy $H(F)$ from the objective in the primal LP relaxation (MMC-P). Recall that the Shannon entropy of a discrete distribution p is $H(p) := -\sum_i p_i \log p_i$, where we adopt the standard convention that $0 \log 0 = 0$. Note that this regularization results in a strictly convex optimization problem by strict concavity of the entropy. This regularization is motivated by the Max-Entropy principle; indeed, recall from Remark 4.2.1 the interpretation of (MMC-P) as an optimization over probability distributions. The choice of the regularization parameter η is discussed in Remark 4.4.6 below, and is based on balancing the fact that (MB-P) is "more convex" and thus easier to solve for small η, while its fidelity to the original problem (MMC-P) improves for large η due to the following basic bound.

Balancing" because every instance of Matrix Balancing in this chapter is in the setting of the ℓ_1 norm.

Lemma 4.4.1 (Entropy bound)**.** *For any probability distribution $p \in \Delta_K$ with support size $k := |\{i \in [K] : p_i \neq 0\}| \leqslant K$, we have $0 \leqslant H(p) \leqslant \log k$.*

■ 4.4.1.2 Dual regularization

In the dual, we employ *softmin smoothing*: we re-write the dual LP relaxation as the max-min saddle-point problem (MMC-D), and then replace the inner min by a smooth approximation smin_η, which is defined for a parameter $\eta > 0$ by

$$\operatorname*{smin}_{i\in[k]\,\eta} a_i := -\frac{1}{\eta} \log \left(\sum_{i=1}^{k} e^{-\eta a_i} \right),$$

where we adopt the standard convention $e^{-\infty} = 0$ to extend this notation to $a_i \in \mathbb{R} \cup \{+\infty\}$. Note that this regularization results in a concave optimization problem by concavity of the softmin function—in fact, strictly concave on the orthogonal complement of the subspace spanned by $\mathbf{1}$. A similar discussion as for the primal regularization applies about the choice of regularization parameter η, except that here the fidelity of the regularized problem to the original unregularized problem is based on the following basic bound.

Lemma 4.4.2 (Softmin approximation bound)**.** *For any $a_1, \ldots, a_k \in \mathbb{R} \cup \{+\infty\}$ and $\eta > 0$,*

$$0 \leqslant \min_{i\in[k]} a_i - \operatorname*{smin}_{i\in[k]\,\eta} a_i \leqslant \frac{\log k}{\eta}$$

The regularized optimization problem (MB-D) is given in Table 4.3. Expanding the softmin and re-parameterizing $x := -\eta p$ gives the more convenient equivalent form:

$$-\frac{1}{\eta} \min_{x\in\mathbb{R}^n} \log \left(\sum_{ij} e^{x_i - x_j} K_{ij} \right), \tag{MB-D'}$$

where $K := \exp[-\eta W]$ denotes the *entrywise* exponentiated matrix with entries $K_{ij} = e^{-\eta W_{ij}}$.

■ 4.4.1.3 Connections and remarks

Not only are (MB-P) and (MB-D) both convex optimization problems, but also they are convex duals[8] satisfying strong duality. The optimality conditions clarify the connection between these problems and Matrix Balancing: the (unique) solution of (MB-P) corresponds to the (unique) balancing of K modulo normalization,

[8] Formally, this requires equivalently re-writing (MB-D) in constrained form.

and the solutions of (MB-D') (unique up to translation by $\mathbf{1}$) correspond to the diagonal balancing matrices (unique up to a constant factor). This is formally stated as follows.

Lemma 4.4.3 (Optimality conditions for (MB-P) and (MB-D'))**.** *Let $G = (V, E, w)$ be strongly connected and $\eta > 0$. Then:*

(1) $F \in \mathcal{F}_E$ and $x \in \mathbb{R}^n$ are optimal solutions for (MB-P) *and* (MB-D')*, respectively, if and only if $F = A/\sum_{ij} A_{ij}$, where $A = \operatorname{diag}(e^x) K \operatorname{diag}(e^{-x})$.*

(2) (MB-P) *has a unique solution. The solutions to* (MB-D') *are unique up to translation by $\mathbf{1}$.*

A similar result can be found in [126, Theorem 1], although the focus there is on the dual regularized problem. For completeness, we provide a short proof here that highlights the primal regularized problem and the convex duality.

Proof. Dualize the affine constraint $F\mathbf{1} = F^T\mathbf{1}$ in (MB-P) via the penalty $p^T(F\mathbf{1} - F^T\mathbf{1}) = \sum_{(i,j)\in E} F_{ij}(p_i - p_j)$, where $p \in \mathbb{R}^n$ is the associated Lagrange multiplier. This results in the minimax problem

$$\min_{F\in\Delta_E} \max_{p\in\mathbb{R}^n} \sum_{(i,j)\in E} F_{ij}(W_{ij} + p_i - p_j + \eta^{-1} \log F_{ij}) \tag{4.1}$$

By Sion's Minimax Theorem [204], this equals the maximin problem

$$\max_{p\in\mathbb{R}^n} \min_{F\in\Delta_E} \sum_{(i,j)\in E} F_{ij}(W_{ij} + p_i - p_j + \eta^{-1} \log F_{ij}) \tag{4.2}$$

The inner minimization problem can now be solved explicitly. A standard Lagrange multiplier calculation shows that at optimality, F is the matrix with ij-th entry equal to

$$F_{ij} = ce^{-\eta(W_{ij}+p_i-p_j)} \tag{BAL-OPT}$$

where $c = 1/(\sum_{(i,j)\in E} e^{-\eta(W_{ij}+p_i-p_j)})$ is the normalizing constant. (Note that if $(i,j) \notin E$, then $W_{ij} = \infty$, so $F_{ij} = e^{-\eta W_{ij}} = 0$.) Plugging (BAL-OPT) into (4.2) and simplifying yields

$$\max_{p\in\mathbb{R}^n} -\eta^{-1} \log \left(\sum_{(i,j)\in E} e^{-\eta(W_{ij}+p_i-p_j)} \right), \tag{4.3}$$

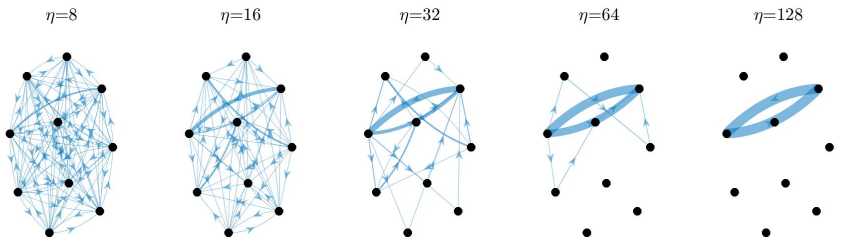


Figure 4.1: Effect of entropic regularization on the sparsity of the optimal solution F^η of (MB-P). Plotted here is F^η for varying regularization parameter η, where edge (i,j) is drawn with width proportional to F^η_{ij}, and dropped if it has sufficiently small mass.

which is precisely (MB-D). This establishes strong duality. Item (1) then follows from the optimality condition established above in (BAL-OPT).

For item (2), strict concavity of entropy implies that (MB-P) has a unique optimal solution. This combined with the optimality condition in item (1) implies that $\mathrm{diag}(e^x)K\,\mathrm{diag}(e^{-x})$ is invariant among optimal solutions x of (MB-D'). Thus if x and y are both solutions, then $x_i - x_j = y_i - y_j$ for all edges $(i,j) \in E$. It follows that in each strongly connected component of G, the difference $x_i - y_i$ is constant over all vertices i. Since G is assumed strongly connected, x and y are equal up to an additive shift of $\mathbf{1}$. □

The strongly connected assumption in Lemma 4.4.3 is important for balanceability:

Remark 4.4.4 (Balanceability for MMC)**.** $K \in \mathbb{R}^{n\times n}_{\geqslant 0}$ *is* balanceable *if and only if* K *is* irreducible*—i.e., the graph* $G_K = ([n], \mathit{supp}(K))$ *is strongly connected [166]. Thus, in our MMC application,* $K = \exp[-\eta W]$ *is balanceable since* $G = G_K$ *is strongly connnected (see §4.2). Furthermore, balanceability is necessary and sufficient for uniqueness (modulo translation) of the solutions to the dual regularized problem* (MB-D')*, essentially because balanceability can be shown to be equivalent to strict concavity of the dual regularized problem* (MB-D') *on the orthogonal complement of the subspace spanned by* $\mathbf{1}$*.*

We conclude this discussion with two remarks about the regularization parameter η.

Remark 4.4.5 (Effect of regularizing MMC)**.** *The solution* F^η *of* (MB-P) *is readily characterized in the limit as the regularization dominates* ($\eta \to 0$) *or vanishes* ($\eta \to \infty$)*:* $\lim_{\eta\to 0} F^\eta$ *is the max-entropy element of* $\mathcal{F}_E$*, and* $\lim_{\eta\to\infty} F^\eta$

is the max-entropy solution among optimal solutions for (MMC-P).[9] *For every finite η, the solution F^η is dense in that $F^\eta_{ij} > 0$ for every edge (i, j). However, as η increases (i.e., the regularization decreases), F^η concentrates on edges belonging to Min-Mean-Cycle(s); see Figure 4.1 for an illustration.*

Remark 4.4.6 (Tradeoff for regularizing MMC)**.** *There is a natural algorithmic tradeoff for choosing η: roughly, more regularization makes $K = \exp[-\eta W]$ easier to balance, while less regularization ensures fidelity of the regularized problems to the original LPs. Therefore, we take η as small as possible such that solving the regularized problems yields an $O(\varepsilon)$ optimal solution for the original LPs (and thus MMC). A simple argument—either bounding the primal entropy regularization by $\eta^{-1} \log m$ using Lemma 4.4.1, or bounding the dual softmin approximation error by $\eta^{-1} \log m$ using Lemma 4.4.2—shows that $\eta = O(\varepsilon^{-1} \log m)$ suffices.*

■ 4.4.2 Optimization via Matrix Balancing

We now make the connections in §4.4.1 algorithmic by reducing the optimization step in the algorithmic framework described in §4.3, to Matrix Balancing. Although Matrix Balancing is difficult to perform *exactly*, we show that performing it *approximately* suffices.

Definition 4.4.7 (Approximate Matrix Balancing)**.** *A nonnegative matrix A is* δ-balanced *if*

$$\frac{\|A\mathbf{1} - A^T\mathbf{1}\|_1}{\sum_{ij} A_{ij}} \leqslant \delta. \tag{4.4}$$

The approximate Matrix Balancing problem *for $K \in \mathbb{R}^{n\times n}_{\geqslant 0}$ and $\delta > 0$ is to find a positive diagonal matrix D such that $A := DKD^{-1}$ is δ-balanced and satisfies $\sum_{ij} A_{ij} \leqslant \sum_{ij} K_{ij}$.*[10]

We now state the main result of this section: a reduction from the optimization step in the algorithmic framework described in §4.3, to approximately balancing the matrix $K = \exp[-\eta W]$ to accuracy $\delta = \Theta(\varepsilon / (w_{\max} d))$, where $\eta = \Theta((\log m)/\varepsilon)$. The upshot is that this allows us to leverage known near-linear time algorithms for approximate Matrix Balancing.

[9] This is in analog to entropic Optimal Transport [178, Proposition 4.1], and can be proved similarly.

[10] The second condition $\sum_{ij} A_{ij} \leqslant \sum_{ij} K_{ij}$ is only for technical purposes (it ensures conditioning bounds, see Lemma 4.4.11) and is a mild requirement since all natural balancing algorithms satisfy it. Indeed, balancing K is equivalent to minimizing $\sum_{ij} A_{ij}$ (Lemma 4.4.3), and $\sum_{ij} K_{ij}$ is the value of $\sum_{ij} A_{ij}$ without any balancing.

Theorem 4.4.8 (Efficient optimization via Matrix Balancing)**.** *Let $G = (V, E, w)$ be strongly connected, $\eta = (2.5 \log m)/\varepsilon$, and $\delta \leqslant \varepsilon/(16 w_{\max} d)$. Let $x \in \mathbb{R}^n$ be such that $\operatorname{diag}(e^x)$ solves the δ-approximate Matrix Balancing problem on $K = \exp[-\eta W]$, and denote $A := \operatorname{diag}(e^x) K \operatorname{diag}(e^{-x})$. Then $P = A/(\sum_{ij} A_{ij})$ satisfies $P \in \Delta_E$, $\delta(P) \leqslant \delta$, and $\langle P, W \rangle \leqslant \mu(G) + \varepsilon/2$.*

It is clear by construction that $P \in \Delta_E$ and $\delta(P) \leqslant \delta$; the near-optimality $\langle P, W \rangle \leqslant \mu(G) + \varepsilon/2$ is what requires proof. The intuition is as follows. Since P is approximately balanced, the (nearly feasible) pair of primal-dual solutions (P, x) nearly satisfies the optimality conditions in Lemma 4.4.3, and thus P is nearly optimal for (MB-P). Since (MB-P) is pointwise close to the primal LP relaxation (MMC-P) (since the regularization is small by Lemma 4.4.1), therefore P is also nearly optimal for the original optimization problem (MMC-P).

To formalize this intuition we require three lemmas. First, we compute the gap between objectives for a certain family of primal-dual "solution" pairs for (MB-P) and (MB-D') inspired by the optimality conditions in Lemma 4.4.3. Note that the primal solution may not be feasible since it may not be balanced—in fact, Lemma 4.4.9 shows that this imbalance controls this gap.

Lemma 4.4.9 (Duality gap)**.** *Let $\eta > 0$ and $x \in \mathbb{R}^n$. Define $K = \exp[-\eta W]$, $A = \operatorname{diag}(e^x) K \operatorname{diag}(e^{-x})$, and $P = A/(\sum_{ij} A_{ij})$. Then $(\langle P, W \rangle - \eta^{-1} H(P)) - (-\eta^{-1} \log \sum_{ij} A_{ij}) = \eta^{-1} x^T (P\mathbf{1} - P^T \mathbf{1})$.*

Proof. The proof is a straightforward calculation. Denote $s = \sum_{ij} A_{ij}$. Note that $-\eta^{-1} H(P) = \eta^{-1} \sum_{ij} P_{ij} \log P_{ij} = \eta^{-1} \sum_{ij} P_{ij} \log(A_{ij}/s) = \eta^{-1} \sum_{ij} P_{ij} \log A_{ij} - \eta^{-1} \log s$. Thus

$$\begin{aligned}
\langle P, W \rangle - \eta^{-1} H(P) + \eta^{-1} \log s &= \langle P, W \rangle + \eta^{-1} \sum_{ij} P_{ij} \log A_{ij} \\
&= \langle P, W \rangle + \eta^{-1} \sum_{ij} P_{ij} (x_i - x_j + \log K_{ij}) \\
&= \eta^{-1} x^T (P\mathbf{1} - P^T \mathbf{1}).
\end{aligned}$$

Above, the second equality is by definition of A, and the third equality is by definition of K and simplifying. □

The second lemma shows that the dual balancing objective gives a lower bound on MMC. This amounts to the pointwise nonnegativity of our regularizations of the LP relaxations.

Lemma 4.4.10 (Lower bound on MMC via balancing)**.** *Consider any $\eta > 0$ and $x \in \mathbb{R}^n$. Let $K = \exp[-\eta W]$ and $A = \operatorname{diag}(e^x) K \operatorname{diag}(e^{-x})$. Then $-\eta^{-1} \log \sum_{ij} A_{ij} \leqslant \mu(G)$.*

Proof. Let $p = \eta x$. By Lemma 4.4.2, $-\eta^{-1} \log \sum_{ij} A_{ij} = \operatorname{smin}_{\eta,(i,j)\in E} W_{ij} + p_i - p_j \leqslant \min_{(i,j)\in E} W_{ij} + p_i - p_j$. By feasibility of p for the dual LP relaxation (MMC-D), this is at most $\mu(G)$. □

The third lemma is a standard conditioning bound (e.g., [17, Lemma 3.5]) for nontrivial balancings, i.e., $x \in \mathbb{R}^n$ with objective for (MB-D') no worse than $\mathbf{0}$. Below, let $\kappa := \frac{\sum_{ij} K_{ij}}{\min_{ij\in \operatorname{supp}(K)} K_{ij}}$.

Lemma 4.4.11 (Conditioning of nontrivial balancings)**.** *Let $K \in \mathbb{R}^{n\times n}_{\geqslant 0}$ be balanceable and $G = ([n], supp(K))$. If $x \in \mathbb{R}^n$ satisfies $\sum_{ij} e^{x_i - x_j} K_{ij} \leqslant \sum_{ij} K_{ij}$, then $\max_i x_i - \min_i x_i \leqslant d \log \kappa$.*

Note that in `AMMC`, we have $K = \exp[-\eta W]$ and $\eta = O((\log m)/\varepsilon)$, and thus

$$\log \kappa \leqslant \log \frac{m \exp(\eta w_{\max})}{\exp(-\eta w_{\max})} = \log m + 2\eta w_{\max} \tag{4.5}$$

which is of size $O((w_{\max}/\varepsilon) \log m)$. We are now ready to prove Theorem 4.4.8.

Proof of Theorem 4.4.8. Rearranging the inequality in Lemma 4.4.9 yields

$$\langle P, W\rangle = -\eta^{-1} \log \sum_{ij} A_{ij} + \eta^{-1} H(P) + \eta^{-1} x^T (P\mathbf{1} - P^T\mathbf{1}).$$

We show the right hand side is at most $\mu(G) + \varepsilon/2$. The first term is at most $-\eta^{-1} \log \sum_{ij} A_{ij} \leqslant \mu(G)$ by Lemma 4.4.10. The second term is at most $\eta^{-1} H(P) \leqslant \eta^{-1} \log m = 2\varepsilon/5$ by Lemma 4.4.1 and the choice of η. Finally, the third term is at most

$$\frac{1}{\eta} x^T (P\mathbf{1} - P^T\mathbf{1}) \leqslant \frac{1}{2\eta} (\max_i x_i - \min_i x_i) \, \|P\mathbf{1} - P^T\mathbf{1}\|_1 \leqslant \frac{\delta d}{2\eta} \log \kappa \leqslant \frac{\varepsilon}{10},$$

where above the first inequality is by applying Hölder's inequality after possibly re-centering x (since $x \mapsto x^T(P\mathbf{1} - P^T\mathbf{1})$ is invariant under adding multiples of the all-ones vector $\mathbf{1}$ to x); the second inequality is by Lemma 4.4.11 and the construction of P by re-normalizing a δ-balanced matrix; and the final inequality is by the conditioning bound (4.5), the choice of η, and the bound $\varepsilon \leqslant 2w_{\max}$ (which may be assumed otherwise every cycle is ε-suboptimal). □

■ 4.5 Efficient rounding of the LP relaxation

Here we present an efficient implementation of the rounding step in the algorithmic framework described in §4.3.

Theorem 4.5.1 (Efficient rounding)**.** *There is an algorithm (namely, **ROUNDQCIRC** in §4.5.1 followed by **ROUNDCYCLE** in §4.5.2) that, given $G = (V, E, w)$, a normalized flow $P \in \Delta_E$ with netflow imbalance $\delta(P) \leqslant 1/d$, and an accuracy $\varepsilon > 0$, takes $O(mdw_{\max}/\varepsilon)$ time to output a cycle σ in G satisfying*

$$\bar{w}(\sigma) \leqslant \langle P, W \rangle + \frac{\varepsilon}{4} + 4w_{\max} d\delta(P).$$

In particular, if $\delta(P) \leqslant \varepsilon/(16w_{\max}d)$, then $\bar{w}(\sigma) \leqslant \langle P, W \rangle + \varepsilon/2$.

Furthermore, this algorithm can be implemented using only $O(n)$ additional memory. But since this modification is a minor extension, we defer it to §4.8.1 for ease of exposition.

We perform the rounding in two steps. First, `ROUNDQCIRC` rounds the near-circulation P to a circulation $F \in \mathcal{F}_E$ such that (i) little flow is adjusted, and (ii) F is γ-quantized[11] for an appropriately chosen scalar γ. Property (i) ensures that the cost is approximately preserved, and property (ii) enables the efficient implementation of the second step. Second, `ROUNDCYCLE` rounds $F \in \mathcal{F}_E$ to a vertex while preserving the cost. The formal guarantees are as follows.

Lemma 4.5.2 (Guarantee for `ROUNDQCIRC`)**.** *Given $G = (V, E, w)$, $P \in \Delta_E$ satisfying $\delta(P) \leqslant 1/d$, and $\varepsilon > 0$, **ROUNDQCIRC** takes $O(m + nd)$ time to output $F \in \mathcal{F}_E$ such that F is γ-quantized for $\gamma = \Omega(\varepsilon/(mdw_{\max}))$, and*

$$\|F - P\|_1 \leqslant 4d\delta(P) + \frac{\varepsilon}{4w_{\max}}. \tag{4.6}$$

Lemma 4.5.3 (Guarantee for `ROUNDCYCLE`)**.** *Given $G = (V, E, w)$ and a γ-quantized $F \in \mathcal{F}_E$, **ROUNDCYCLE** takes $O(m + \gamma^{-1})$ time to output a cycle σ satisfying $\bar{w}(\sigma) \leqslant \langle W, F \rangle$.*

The proof of Theorem 4.5.1 is immediate from these two lemmas.

Proof of Theorem 4.5.1. The runtime follows from Lemmas 4.5.2 and 4.5.3. Let F be the output of `ROUNDQCIRC`. By Lemma 4.5.3, $\bar{w}(\sigma) \leqslant \langle F, W \rangle = \langle P, W \rangle + \langle F - P, W \rangle$. By Hölder's inequality and Lemma 4.5.2, $\langle F - P, W \rangle \leqslant w_{\max}\|F - P\|_1 \leqslant 4w_{\max}d\delta(P) + \varepsilon/4$. □

§4.5.1 and §4.5.2 respectively detail these subroutines `ROUNDQCIRC` and `ROUNDCYCLE`, and prove their respective guarantees Lemmas 4.5.2 and 4.5.3.

[11] We say a matrix is γ-quantized if each entry is an integer multiple of γ.

4.5.1 Rounding to the circulation polytope

Here we describe the algorithm `ROUNDQCIRC` and prove Lemma 4.5.2. Let us first ignore quantization: given G and a normalized flow $P \in \Delta_E$, how to efficiently compute a normalized circulation $F \in \mathcal{F}_E$ such that the adjusted flow $\|F - P\|_1$ is small compared to the total netflow imbalance $\delta(P) = \|P\mathbf{1} - P^T\mathbf{1}\|_1$? Since this does not require edge weights, we may presently think of G as unweighted.

A simple approach is: until all vertices have balanced flow, push flow from any vertex i with negative netflow to any vertex j with positive netflow along the shortest path in G until i or j is balanced. After a normalization at the end, this produces an $F \in \mathcal{F}_E$ satisfying[12]

$$\|F - P\|_1 = O(d\delta(P)). \tag{4.7}$$

While this ratio $\|F - P\|_1/\delta(P)$ is optimally small in the worst-case, the runtime is a prohibitive $\Theta(mn)$. The bottleneck is $\Theta(n)$ shortest path computations, each taking $\Theta(m)$ time.

A simple trick for speeding this up while maintaining (4.7) is to use cheap estimates of the shortest paths that are of length at most $2d$. Specifically, choose any vertex $v \in V$, and route all paths used in the flow-rebalancing through v using the shortest path to/from v. See Algorithm 4.2 for pseudocode. Note that computing all shortest paths to/from v (line 1 of `ROUNDCIRC`) takes $O(m)$ time by running two Breadth First Searches [196, §6.2]. Note also that by maintaining two list of vertices, one for vertices i with positive flow imbalance $\delta_i(Q) > 0$ and the other for vertices i with negative flow imbalance $\delta_i(Q) < 0$, we can implement both lines 3 and 4 of `ROUNDCIRC` in constant time.

Input: Digraph $G = (V, E)$, normalized flow $P \in \Delta_E$
Output: Normalized circulation $F \in \mathcal{F}_E$ satisfying (4.8)

1: Choose $v \in V$, compute shortest paths to and from v
2: $Q \leftarrow P$, $\delta(Q) \leftarrow Q^T\mathbf{1} - Q\mathbf{1}$ ▷ Initial imbalance
3: **while** $\delta(Q) \neq 0$ **do**
4: Choose any vertices i and j with $\delta_i(Q) > 0$ and $\delta_j(Q) < 0$
5: $\delta_{ij} \leftarrow \min(\delta_i(Q), -\delta_j(Q))$
6: Add δ_{ij} in Q to each edge on paths $i \to v \to j$ found in line 1 ▷ Push flow
7: $\delta_i(Q) \leftarrow \delta_i(Q) - \delta_{ij}$, $\delta_j(Q) \leftarrow \delta_j(Q) + \delta_{ij}$ ▷ Update imbalance
8: **return** $F \leftarrow Q/\sum_{ij} Q_{ij}$

Algorithm 4.2: `ROUNDCIRC`: efficiently rounds to $\mathcal{F}_E$ without adjusting much flow.

[12]This follows from essentially the same argument as in the proof of Lemma 4.5.4.

Lemma 4.5.4 (Guarantee for `ROUNDCIRC`)**.** *Given a strongly connected digraph $G = (V, E)$ and a matrix $P \in \Delta_E$, `ROUNDCIRC` takes $O(m + nd)$ time to output $F \in \mathcal{F}_E$ satisfying*

$$\|F - P\|_1 \leqslant 2d\delta(P). \tag{4.8}$$

Proof. All steps besides the while loop take $O(m)$ time. For this loop: each iteration takes $O(d)$ time since flow is pushed along at most $2d$ edges. Also, there are at most n iterations, since each path saturates at least one vertex. Thus the while loop takes $O(nd)$ time.

For correctness, clearly $F \in \mathcal{F}_E$; it remains to show the guarantee (4.8). Consider the path from i to v to j along which we add flow in line 6. Since the paths from i to v and from v to j are both shortest paths, each is of length at most d. Thus the total flow added to the path $i \to v \to j$ is at most $2d\delta_{ij}$. Summing over all paths yields

$$\|Q - P\|_1 \leqslant d\delta(P). \tag{4.9}$$

Now since Q is entrywise bigger than F and P, and since $\|F\|_1 = 1 = \|P\|_1$, we have $\|Q-F\|_1 = \|Q-P\|_1 \leqslant d\delta(P)$. Therefore $\|F-P\|_1 \leqslant \|F-Q\|_1+\|Q-P\|_1 \leqslant 2d\delta(P)$. □

■ 4.5.1.1 Rounding to a quantized circulation

We now address the quantization required in Lemma 4.5.2: simply quantize and re-normalize P before `ROUNDCIRC`. Pseudocode is in Algorithm 4.3. Note this quantization must be performed before `ROUNDCIRC` since quantizing afterwards can unbalance the circulation. Note also that we need an estimate of d for the quantization size; this is computed using the simple algorithm `ADIAM` (see §4.2). The proof of Lemma 4.5.2 (i.e., the guarantee of `ROUNDQCIRC`) is straightforward from Lemma 4.5.4 (i.e., the guarantee of `ROUNDCIRC`), and is deferred to §4.8.2.

Input: Weighted digraph $G = (V, E, w)$, normalized flow $P \in \Delta_E$, accuracy ε
Output: Quantized, normalized circulation $F \in \mathcal{F}_E$ satisfying (4.6)

1: $\tilde{d} \leftarrow$ `ADIAM`(G), $\alpha \leftarrow \varepsilon / (40m\tilde{d}w_{\max})$
2: $R \leftarrow \alpha\lfloor P/\alpha \rfloor$ ▷ Round down P_{ij} to integer multiple of α
3: $\tilde{P} \leftarrow R / \sum_{ij} R_{ij}$ ▷ Renormalize to have unit total flow
4: **return** $F \leftarrow$ `ROUNDCIRC`$(G, \tilde{P})$

Algorithm 4.3: `ROUNDQCIRC`: efficiently rounds to quantized circulation in $\mathcal{F}_E$ without adjusting much flow.

4.5.2 Rounding a circulation to a cycle

Here we describe the algorithm ROUNDCYCLE and prove Lemma 4.5.3. A simple approach for rounding a normalized circulation $F \in \mathcal{F}_E$ to a cycle σ satisfying $\bar{w}(\sigma) \leqslant \langle W, F\rangle$ is to decompose F into a convex decomposition of cycles using the Cycle-Cancelling algorithm [196], and then output the cycle with best objective value. However, the runtime is a prohibitive $\Theta(mn)$. The bottleneck is $\Theta(m)$ cycle cancellations, each taking up to $\Theta(n)$ time. Intuitively, this factor of n arises since cancelling a long cycle of length up to n takes a long time yet does not give more "benefit" than a short cycle. We speed up this algorithm by exploiting the quantization of F to ensure that cancelling long cycles gives a proportionally larger benefit than short cycles.

Specifically, let ROUNDCYCLE be the following minor modification of the Cycle-Cancelling algorithm. Initialize $\tilde{F} = F$. While $\tilde{F} \neq 0$, choose any vertex i that has an outgoing edge (i, j) with nonzero flow $\tilde{F}_{ij} \neq 0$. Run Depth First Search (DFS) from i until some cycle σ is created. If $\bar{w}(\sigma) \leqslant \langle F, w\rangle$, then terminate. Otherwise, cancel the cycle σ by subtracting $\tilde{f}_\sigma := \min_{e\in\sigma} \tilde{F}_e$ from the flow $\tilde{F}_e$ on each edge $e \in \sigma$. Then continue the DFS in a way that re-uses previous work—this is crucial for near-linear runtime. Specifically, if the previous DFS created a cycle by returning to an intermediate vertex $j \neq i$, then continue the DFS from j, keeping the work done by the DFS from i to j. Otherwise, if the previous DFS created a cycle by returning to the initial vertex i, then restart the DFS at any vertex which has an outgoing edge with nonzero flow. Note that ROUNDCYCLE leverages the quantization only in its runtime analysis.

Proof of Lemma 4.5.3. Correctness is immediate by linearity. For the runtime, the key is the invariant that $\tilde{F}$ remains a γ-quantized circulation. That $\tilde{F}$ is a circulation ensures that the DFS always finds an outgoing edge and thus always finds a cycle since some vertex is eventually repeated. When such a cycle σ is found, its cancellation lowers the total flow $\sum_{ij} \tilde{F}_{ij}$ by $\tilde{f}_\sigma|\sigma|$, which is at least $\gamma|\sigma|$ by the invariant. Since the total flow is initially $\sum_{ij} F_{ij} = 1$, ROUNDCYCLE therefore terminates after cancelling cycles with at most γ^{-1} total edges, counting multiplicity if an edge appears in multiple cancelled cycles. Since processing an edge takes $O(1)$ amortized time (again counting multiplicity), we conclude the desired $O(m + \gamma^{-1})$ runtime bound. □

4.6 Concluding the approximation algorithm

Algorithm 4.4 provides pseudocode for our proposed approximation algorithm AMMC. It instantiates the framework in §4.3 using the approximate Matrix Balanc-

ing reduction in Theorem 4.4.8 for the optimization, and using the algorithm in Theorem 4.5.1 for the rounding. By Theorem 4.4.8, AMMC succesfully approximates MMC regardless of how the balancing is performed. Since balancing is an active area of research (e.g., [9, 17, 67, 167]), we abstract this computation into a subroutine ABAL: given a balanceable $K \in \mathbb{R}^{n\times n}_{\geqslant 0}$ and an accuracy $\delta > 0$, ABAL outputs a vector $x \in \mathbb{R}^n$ such that $\operatorname{diag}(e^x)$ solves approximate Matrix Balancing on K to δ accuracy. Let $\mathcal{T}_{\texttt{ABAL}}(K, \delta)$ and $\mathcal{M}_{\texttt{ABAL}}(K, \delta)$ respectively denote the runtime and memory of ABAL.

Input: Weighted digraph $G = ([n], E, w)$, accuracy $\varepsilon > 0$
Output: Cycle σ in G satisfying $\bar{w}(\sigma) \leqslant \mu(G) + \varepsilon$

\\ Optimization step: compute near-feasible, near-optimal solution P for (MMC-P)
1: $\tilde{d} \leftarrow \texttt{ADIAM}(G)$, $\delta \leftarrow \varepsilon / (16 w_{\max} \tilde{d})$ ▷ Precision to balance
2: $\eta \leftarrow 2.5(\log m)/\varepsilon$, $K \leftarrow \exp[-\eta W]$ ▷ Matrix to balance
3: $x \leftarrow \texttt{ABAL}(K, \delta)$, $A \leftarrow \operatorname{diag}(e^x) K \operatorname{diag}(e^{-x})$, $P \leftarrow A/(\sum_{ij} A_{ij})$ ▷ Balance K

\\ Rounding step: round P to a vertex of $\mathcal{F}_E$ with nearly as good cost for (MMC-P)
4: $F \leftarrow \texttt{ROUNDQCIRC}(G, P, \varepsilon)$ ▷ Correct feasibility and quantize
5: $\sigma \leftarrow \texttt{ROUNDCYCLE}(G, F)$ ▷ Round to vertex
return σ

Algorithm 4.4: AMMC: Matrix Balancing approach for approximating MMC.

Below, §4.6.1 establishes guarantees for AMMC in terms of a general subroutine ABAL, thereby reducing approximating MMC to approximate Matrix Balancing. In §4.6.2, we implement ABAL with concrete, state-of-the-art balancing algorithms to conclude our proposed MMC algorithm.

4.6.1 Reducing MMC to matrix balancing

4.6.1.1 Accuracy and runtime

Theorem 4.6.1.A (Accuracy and runtime of AMMC)**.** *Given a weighted digraph $G = (V, E, w)$ and an accuracy $\varepsilon > 0$, AMMC computes a cycle σ in G satisfying $\bar{w}(\sigma) \leqslant \mu(G) + \varepsilon$ in time $\mathcal{T}_{ABAL}(K, \delta) + O(mdw_{\max}/\varepsilon)$.*

Proof. By the guarantee of ADIAM (see §4.2), $d \leqslant \tilde{d} \leqslant 2d$. The runtime of AMMC follows from the runtimes of its constituent subroutines: $O(m)$ for ADIAM, and $O(mdw_{\max}/\varepsilon)$ for rounding (Theorem 4.5.1). Correctness follows from Observation 4.3.1 since AMMC implements both the optimization step (Theorem 4.4.8) and the rounding step (Theorem 4.5.1) to the accuracies prescribed in the algorithmic framework described in §4.3 for $\delta = \varepsilon/(16 w_{\max}\tilde{d}) \leqslant \varepsilon/(16 w_{\max} d)$. □

■ 4.6.1.2 Memory-optimality

We now describe how to implement AMMC using only $O(n)$ additional memory. For ease of exposition, the memory usage counts the total numbers stored. (In §4.6.1.3, we show AMMC is implementable using $\tilde{O}(1)$-bit numbers.) Since storing G requires $\Theta(m)$ memory, we assume $G = (V, E, w)$ is input to AMMC through two oracles:

- Edge oracle: given $i \in V$ and $k \in [n]$, it returns the k-th incoming and outgoing edges from i (in any arbitrary but fixed orders). If k is larger than the indegree or outdegree of i, the respective query returns null.
- Weight oracle: given $i, j \in V$, it returns $w(i, j)$ if $(i, j) \in E$, and ∞ otherwise.

For simplicity, we assume that queries to these oracles take $O(1)$ time. In practice, the edge oracle can be implemented with simple, standard adjacency lists; and the weight oracle by e.g., hashing or re-computing weights on the fly if $w(\cdot, \cdot)$ is an efficiently computable function.

Critically, in AMMC we do *not* explicitly compute the intermediate matrices K, A, P, and F; instead, we form *implicit* representations for them. To formalize this, it is helpful to define the notion of an (T, M) *matrix oracle* for a matrix: this is a data structure that uses M storage, and can return a queried entry of the matrix in T time and $O(1)$ additional memory.

Theorem 4.6.1.B (Memory-optimality of AMMC)**.** *There is an implementation of AMMC that, given G through its edge and weight oracles, achieves the accuracy guarantee in Theorem 4.6.1.A and uses $\mathcal{T}_{ABAL}(K, \delta) + O(mdw_{\max}/\varepsilon + m \log n)$ time and $\mathcal{M}_{ABAL}(K, \delta) + O(n)$ memory.*

Proof. We form an $(O(1), O(1))$ matrix oracle for K by storing η—a query for entry K_{ij} is performed by querying $w(i, j)$ and computing $e^{-\eta w(i,j)}$. We form an $(O(1), O(n))$ matrix oracle for P by storing x and $s_A := \sum_{ij} e^{x_i - x_j} K_{ij}$—a query for entry P_{ij} is performed by querying K_{ij} and computing $e^{x_i - x_j} K_{ij}/s_A$. This matrix oracle for P is passed as input to the rounding algorithms, which are implemented in the memory-efficient manner in Theorem 4.8.1. □

■ 4.6.1.3 Bit-complexity

Above, our analysis assumes exact arithmetic for ease of exposition; however, numerical precision is an important issue since naïvely implementing AMMC can require large bit-complexity—indeed, since $\max_i x_i - \min_j x_j$ can be $\Omega(d)$ [126, §3], naïvely operating on $A = \operatorname{diag}(e^x) K \operatorname{diag}(e^{-x})$ can require $\Omega(d)$-bit numbers. Here, we establish that AMMC can be implemented on $\tilde{O}(1)$-bit numbers. (This

analysis excludes the ABAL subroutine since we have not yet instantiated it, but the concrete implementation used below also has logarithmic bit complexity; details in §4.6.2.)

Theorem 4.6.1.C (Bit-complexity of AMMC)**.** *There is an implementation of AMMC that, aside from possibly ABAL, performs all arithmetic operations over* $O(\log \frac{nw_{\max}}{\varepsilon}) = \tilde{O}(1)$*-bit numbers and achieves the same runtime bounds (in terms of arithmetic operations), memory bounds (in terms of total numbers stored), and accuracy guarantees as in Theorem 4.6.1.B.*

This implementation essentially only modifies how AMMC computes entries of K, A, and P on the exponential scale by using the log-sum-exp trick. Details are deferred to §4.8.3. Briefly, this modification relies on the observation that AMMC is robust in the sense that it outputs an $O(\varepsilon)$-suboptimal cycle even if these entries are computed to low precision.

■ 4.6.2 Concrete implementation

By Theorem 4.6.1, AMMC approximates MMC using any approximate balancing subroutine ABAL. The fastest practical instantiations of ABAL are variants of Osborne's algorithm [166]. In particular, combining Theorem 4.6.1 with the recent analysis of the Random Osborne algorithm in [17] yields the following near-linear runtime for approximating MMC on graphs with polylogarithmic diameter, both in expectation and with high probability. To emphasize the algorithm's practicality, below we write the single logarithmic factor in the runtime rather than hiding it with the $\tilde{O}$ notation.

Theorem 4.6.2 (Main result: AMMC with Random Osborne)**.** *Consider implementing ABAL using the Random Osborne algorithm in [17]. Then given a weighted digraph G through its edge and weight oracles, and an accuracy $\varepsilon > 0$, AMMC computes a cycle σ in G satisfying $\bar{w}(\sigma) \leqslant \mu(G) + \varepsilon$ using $O(n)$ memory and T arithmetic operations on $O(\log(\frac{nw_{\max}}{\varepsilon})) = \tilde{O}(1)$-bit numbers, where T satisfies*

- *(Expectation guarantee.)* $\mathbb{E}[T] = O(md^2(\frac{w_{\max}}{\varepsilon})^2 \log n)$.
- *(High probability guarantee.) For all* $\alpha \in (0,1)$, $\mathbb{P}\left(T \leqslant md^2(\frac{w_{\max}}{\varepsilon})^2 \log n \log \frac{1}{\alpha}\right) \geqslant 1 - \alpha$.

Proof. The runtime and bit-complexity of Random Osborne follow from [17, Theorem 5.1 and 8.1] combined with the conditioning bound (4.5). Random Osborne requires only $O(n)$ memory since K is given through its query oracle. For the rest of AMMC, apply Theorem 4.6.1. □

Remark 4.6.3 (Numerical implementation)**.** *As described in §4.6.1.3, computing* $K = \exp[-\eta W]$ *runs into numerical precision issues for large* η*. This is circumvented by* not *explicitly computing* K*: numerical implementations of Osborne's algorithm operate on* K_{ij} *only through* $\log K_{ij} = -\eta W_{ij}$*, and compute all intermediate quantities via the log-sum-exp trick [17].*

Remark 4.6.4 (Alternative implementation)**.** *`ABAL` can also be implemented using the algorithm of [67]. This achieves comparable theoretical guarantees*[13]*, but relies on Laplacian solvers which (currently) have no practical implementation [116].*

■ 4.7 Preliminary numerical simulations

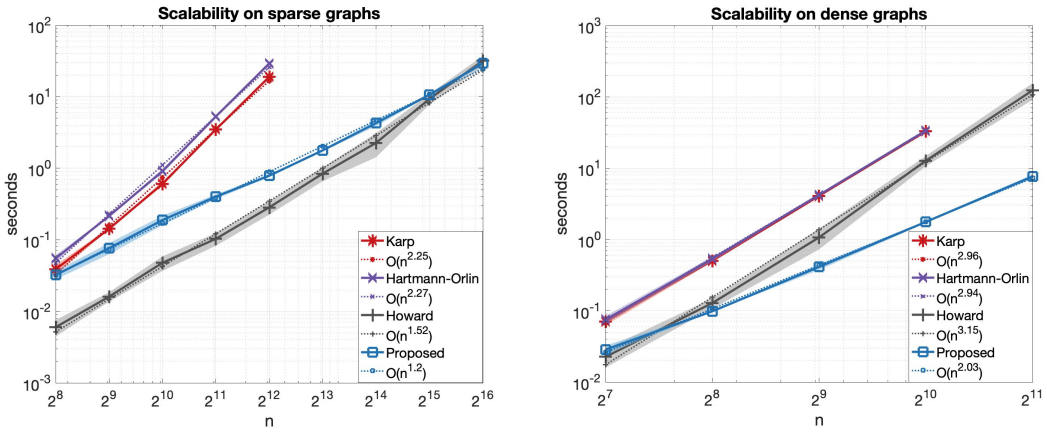
Although the focus of this chapter is theoretical, here we provide preliminary numerical simulations that investigate the practical aspects of our proposed algorithm `AMMC` and validate our theoretical findings.

Experimental setup. We compared `AMMC` with state-of-the-art MMC algorithms on a number of different input graphs (e.g., sparse, dense, random, etc.). In all cases, we empirically observed that `AMMC` had close to linear runtime. Because many problem instances (e.g., random graphs) are "easy" for most MMC algorithms [94], some competitor algorithms ran faster than expected on some of these inputs. Hence, in order to appreciate the differences between `AMMC` and the competitor algorithms, below we benchmark on the "hardest" families of problem instances from the comprehensive experimental survey [94]. These "hard" instances are formed by taking a random graph (either sparse or dense), planting a Hamiltonian Cycle and setting its weights so that it is the Minimum-Mean-Cycle, and then hiding this optimal cycle by randomly permuting the vertices and performing "potential perturbations"; full reproduciblity details are provided in §4.8.4. The resulting graphs are either sparse (with $m \approx 7n$ edges) or dense (with $m \approx n^2/2$ edges), and have a unique Minimum-Mean-Cycle that is maximally long. All experiments are run on a standard 2018 MacBook Pro laptop.

■ 4.7.1 Scalability

Figure 4.2 demonstrates that `AMMC` enjoys (close to) linear runtime in practice and is competitive with the three state-of-the-art algorithms implemented in the popular, heavily-optimized C++ LEMON library [79]. These competitors are the algorithm of Karp [130], the algorithm of Hartmann and Orlin [112],

[13] Namely, $\tilde{O}(md(w_{\max}/\varepsilon)^3)$ arithmetic operations over $\tilde{O}(\text{poly}(w_{\max}/\varepsilon))$-bit numbers (by combining Theorem 4.18 and Lemma 4.24 of [67] with the bound (4.5)).



(a) For sparse graphs with $m = \Theta(n)$ edges, a linear runtime is $O(m) = O(n)$.

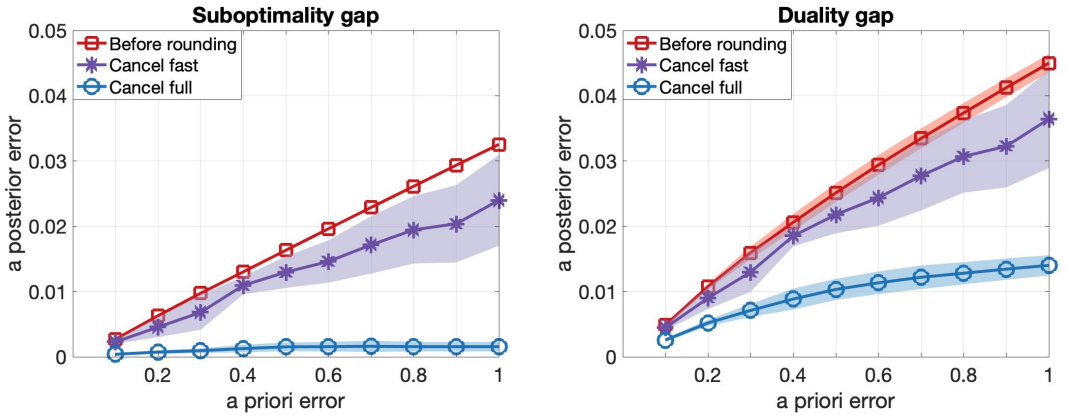
(b) For dense graphs with $m = \Theta(n^2)$ edges, a linear runtime is $O(m) = O(n^2)$.

Figure 4.2: Scalability of our proposed algorithm `AMMC` vs state-of-the-art algorithms implemented in the popular LEMON library [79]. `AMMC` computes an approximate solution (here to roughly 3 digits of precision) whereas the others compute exact solutions. The input instances are described in the main text. We report the average runtime (solid line) over 10 runs, with 1 standard deviation indicated by the shading. We estimate each algorithm's asymptotic runtime using linear regression (dashed line). The asymptotic runtime of `AMMC` on both sparse graphs (left) and dense graphs (right) is close to linear and outperforms all competitors.

and the Howard iteration algorithm [66, 75, 76, 117]. Note that `AMMC` computes approximate solutions whereas these competitors obtain exact solutions. In this experiment, the accuracy parameter of `AMMC` is set so that the suboptimality is $\sim 10^{-3}$ (edge weights are normalized to $[0, 1]$). Smaller ε leads to qualitatively similar results of near-linear runtime, although the constants of course degrade.

In Figure 4.2, we estimate the asymptotic runtime of each algorithm using linear regression; these fits are quite accurate. Observe that `AMMC` has the fastest asymptotic runtime among all competitor algorithms. Moreover, the asymptotic runtime of `AMMC` on both the sparse graph inputs (Figure 4.2a) and dense graph inputs (Figure 4.2b) is close to linear. In contrast, none of the competitor algorithms exhibit near-linear runtime scalings on either input. This enables `AMMC` to scale to larger instances than the competitor algorithms.

We remark that while the LEMON library is heavily-optimized, our implementation of `AMMC` is not. An optimized implementation of `AMMC` may lead to better constants and runtimes. Indeed, as written on page 1 of the empirical survey [94], "efficient implementations of MMC algorithms require nontrivial engineering, including data structures, efficient incremental restart, early termination detection,



(a) Error from the true value of MMC.

(b) Error bound that AMMC can certify.

Figure 4.3: AMMC often finds significantly better approximations than our worst-case theoretical bounds guarantee. This is demonstrated by plotting the *a posteriori* error versus the *a priori* error estimate ε. The *a posteriori* error is measured via the suboptimality (left) and the duality gap (right). The input is the sparse graphs described in the main text, with $n = 2^{12}$ vertices. See the main text for a detailed description of the three plotted lines. We report the average performance over 50 runs, with 1 standard deviation indicated by the shading.

and hybrid algorithms." These are interesting directions for future research, but out of the scope of this chapter.

We also remark that we implement AMMC with a slightly different variant of Osborne's algorithm than in our theoretical results: Random-Reshuffle Cyclic Osborne (see [17] for a description). Random Osborne is used in our theoretical analysis and provably yields near-linear runtimes (Theorem 4.6.2). Random-Reshuffle Cyclic Osborne often enjoys slightly faster empirical convergence, but comparable theoretical guarantees are not known.

■ 4.7.2 Outperforming worst-case theoretical guarantees

Here we mention that AMMC often finds significantly better approximations than our worst-case theoretical guarantees. A constant factor improvement is of course explained by the fact that we have not optimized the constants in this chapter. However, even better performance appears to occur if the Cycle-Cancelling subroutine ROUNDCYCLE described in §4.5.2 is not terminated early; that is, if the fractional Matrix Balancing circulation is fully decomposed into cycles and the best one is output. The point is that often, at least one of these cycles is significantly better than the average—which is all that can be guaranteed in the

worst-case by a linearity argument (c.f. §4.5.2). Note also that our near-linear runtime bound still applies to this modified algorithm (since this is simply the worst-case of our proved runtime bound, c.f. the proof of Lemma 4.5.3).

To investigate the practical improvement from different versions of `ROUNDCYCLE`, we plot in Figure 4.3 the error of three increasingly finer estimates of $\mu(G)$ that `AMMC` (implicitly) makes:

- "Before rounding" refers to the value $\langle W, F\rangle$ of the normalized circulation F computed by `AMMC` before `ROUNDCYCLE` (i.e., the output of `ROUNDQCIRC`).
- "Cancel fast" refers to the value $\bar{w}(\sigma_{\text{fast}})$ of the cycle σ_{fast} computed by the version of `ROUNDCYCLE` that terminates early.
- "Cancel full" refers to the value $\bar{w}(\sigma_{\text{full}})$ of the cycle σ_{full} computed by the version of `ROUNDCYCLE` that does not terminate early.

Clearly, $\langle W, F\rangle \geqslant \bar{w}(\sigma_{\text{fast}}) \geqslant \bar{w}(\sigma_{\text{full}}) \geqslant \mu(G)$. Indeed, each of these three estimates is an upper bound on $\mu(G)$ by feasibility for the primal LP (MMC-P). In Figure 4.3a, we plot this primal suboptimality, a.k.a., the difference between the estimate and $\mu(G)$. Note that this suboptimality is not computable with `AMMC` since it requires the exact value of MMC. In Figure 4.3b, we plot an upper bound on this suboptimality that `AMMC` can provably certify: the duality gap between these primal estimates and the estimate of the dual LP (MMC-D) obtained by using the approximate Matrix Balancing solution computed in step 1 of `AMMC`.

As Figure 4.3 shows, in practice the error of `AMMC`—measured either via the true suboptimality or the certifiable duality gap—is much better than the worst-case bounds when `ROUNDCYCLE` is terminated early, and moreover is even better when `ROUNDCYCLE` is run to completion.

4.8 Deferred details

4.8.1 Memory optimality of the rounding algorithm

Here we describe a memory-efficient implementation of the rounding algorithm in Theorem 4.5.1. See §4.6.1.2 for the definitions of a matrix oracle and the edge and weight oracles of a graph. Note that in what follows, $T = O(1)$ and $M = O(n)$ for `AMMC`; see Theorem 4.6.1.B.

Theorem 4.8.1 (Memory-efficient rounding)**.** *If G is given through its edge oracle and weight oracle, and P is given through an (T, M) matrix oracle, then the algorithm in Theorem 4.5.1 can be run in $O(m(T + \log n + dw_{\max}/\varepsilon))$ time and $O(M + n)$ memory.*

Proof. We describe how to implement the algorithms in Theorem 4.5.1 in a memory-efficient way that does not change the outputted cycle. The subroutine `ADIAM` can be implemented using $O(n)$ memory since Breadth First Search can be implemented using the edge oracle for G and $O(n)$ memory. To perform lines 2 and 3, `ROUNDQCIRC` forms an $(T + O(1), M + O(1))$ matrix oracle for $\tilde{P}$ by using $O(1)$ additional memory to compute and store $s_R := \sum_{ij} R_{ij}$—then an entry $\tilde{P}_{ij}$ can be queried by querying P_{ij} and computing $\alpha\lfloor P_{ij}/\alpha \rfloor / s_R$.

`ROUNDCIRC` takes this matrix oracle for $\tilde{P}$ as input and forms an $(T+O(\log n), M+O(n))$ matrix oracle for F. Specifically, it implicitly performs line 6 by storing in a Balanced Binary Search Tree the amount of flow, totalled over these saturating paths, pushed along each edge. This takes $O(n)$ additional storage since all edges lie on the Shortest Paths trees in or out of v, which collectively contain at most $2(n-1)$ edges. The matrix oracle for F also stores $s_Q := \sum_{ij} Q_{ij}$—then an entry F_{ij} can be queried by querying $\tilde{P}_{ij}$, querying the amount of adjusted flow on edge (i, j) in the Balanced Binary Search Tree, and re-normalizing by s_Q.

In `ROUNDCYCLE`, we maintain for each vertex i a counter j_i. This is the lowest index with respect to the (outgoing) edge oracle of G, that corresponds to an outgoing edge from i with nonzero flow. The DFS always takes these edges. We query each F_{ij} at most once: the first time we cancel a cycle with that edge. If the edge is partially cancelled, then we store the remaining flow. (If the edge is fully saturated, then we do not need to store anything since we will never come back to it). By the bias of the DFS, there are always at most n partially cancelled edges (one for each vertex), so this requires $O(n)$ additional memory. □

■ 4.8.2 Proof of Lemma 4.5.2

Lemma 4.8.2 (Helper lemma for `ROUNDQCIRC`)**.** *Consider P, R, $\tilde{P}$, and α in `ROUNDQCIRC`. Then (i) $\|\tilde{P} - P\|_1 \leqslant 2\alpha m$, and (ii) $\delta(\tilde{P}) \leqslant 2\delta(P) + 4\alpha m$.*

Proof. Proof of item (i). First note that since rounding P to R changes every entry by at most α, thus $\|R - P\|_1 \leqslant \alpha m$, and so also $\sum_{ij} R_{ij} \geqslant 1 - \alpha m$. By definition of $\tilde{P}$, $\|\tilde{P} - R\|_1 = 1 - \|R\|_1 \leqslant \alpha m$. Thus by the triangle inequality, $\|\tilde{P} - P\|_1 \leqslant \|\tilde{P} - R\|_1 + \|R - P\|_1 \leqslant 2\alpha m$.

Proof of item (ii). Note that rounding on an edge to an integer multiple of α increases the flow imbalance at each adjacent vertex by at most α, thereby increasing the total imbalance by at most 2α. Thus R has imbalance at most $\delta(R) \leqslant \delta(P) + 2\alpha m$. By definition of $\tilde{P}$, we have $\delta(\tilde{P}) = \delta(R)/(\sum_{ij} R_{ij}) \leqslant (\delta(P)+2\alpha m)/(\sum_{ij} R_{ij})$. We therefore conclude by observing that $1/(\sum_{ij} R_{ij}) \leqslant 2$, which follows from $\sum_{ij} R_{ij} \geqslant 1-\alpha m$ combined with the fact that $\alpha \leqslant 1/(2m)$. □

Proof of Lemma 4.5.2. The runtime bound follows from the runtimes of `ADIAM` (see §4.2) and `ROUNDCIRC` (Lemma 4.5.4). The guarantee $F \in \mathcal{F}_E$ is immediate from Lemma 4.5.4.

Next, we establish (4.6). By item (i) of Lemma 4.8.2, $\|\tilde{P} - P\|_1 \leqslant 2\alpha m$. Moreover, by Lemma 4.5.4 and then item (ii) of Lemma 4.8.2, $\|F - \tilde{P}\|_1 \leqslant 2d\delta(\tilde{P}) \leqslant 4d\delta(P) + 8\alpha md$. Thus $\|F - P\|_1 \leqslant \|F - \tilde{P}\|_1 + \|\tilde{P} - P\|_1 \leqslant 4d\delta(P) + 10\alpha md$. By our choice of α and the bound $\tilde{d} \geqslant d$ (see §4.2)), the latter summand is at most $\varepsilon/(4w_{\max})$.

Finally, we establish the quantization guarantee. By construction, R is α-quantized, and so $\tilde{P}$ is β-quantized for $\beta := \alpha/(\sum_{ij} R_{ij}) \geqslant \alpha$. Since $\tilde{P}$ is the input to `ROUNDCIRC` in `ROUNDQCIRC`, in `ROUNDCIRC` Q will be β-quantized since $\tilde{P}$ is. Thus F is γ-quantized for $\gamma := \beta/\sum_{ij} Q_{ij}$. Now $\sum_{ij} Q_{ij} = \sum_{ij} \tilde{P}_{ij} + \sum_{ij}(Q_{ij} - \tilde{P}_{ij}) \leqslant 1 + d\delta(\tilde{P})$ by (4.9), and this is $O(1)$ by item (ii) of Lemma 4.8.2 and the assumption that $\delta(P) \leqslant 1/d$. Therefore $\gamma = \Omega(\beta) = \Omega(\alpha)$. We conclude by our choice of α and the fact that $\tilde{d} \leqslant 2d$ (see §4.2). □

■ 4.8.3 Bit complexity

Here we prove Theorem 4.6.1.C. For simplicity of exposition, we omit constants and show how to ensure `AMMC` outputs an $O(\varepsilon)$-suboptimal cycle; the claim then follows by re-normalizing ε.

Proof of Theorem 4.6.1.C. Modification of `AMMC`. The computation of A and P is modified slightly as follows. Let $\alpha = c\varepsilon/(w_{\max} md)$ for a sufficiently small constant c. (i) Read and store the input weights W_{ij} and the output x of `ABAL` to $\pm\alpha$ precision. (ii) Compute and store $Y_{ij} := x_i - x_j + \eta W_{ij}$ to $\pm\alpha$ precision for each $(i,j) \in E$. (iii) Translate $Z_{ij} := Y_{ij} - y$, where $y := \max_{ij} Y_{ij}$. (iv) Compute $A_{ij} = e^{Z_{ij}}$ to $\pm\alpha^2$ precision if $Z_{ij} \geqslant \log\alpha$, and set $A_{ij} = 0$ otherwise. (v) Compute entries of $P = A/\sum_{ij} A_{ij}$ to $\pm\alpha$ precision.

Bit-complexity analysis. By definition of α, $\log\frac{1}{\alpha} = O(\log\frac{nw_{\max}}{\varepsilon}) = \tilde{O}(1)$. (i) The bit complexity of the stored weights is thus $O(\log\frac{w_{\max}}{\alpha}) = \tilde{O}(1)$. The bit complexity of the stored x is $O(\log\frac{\max_i x_i - \min_i x_i}{\alpha}) = \tilde{O}(1)$, since $\log(\max_i x_i - \min_i x_i) = O(\log\frac{nw_{\max}}{\varepsilon}) = \tilde{O}(1)$ by Lemma 4.4.11 and (4.5). (ii), (iii) The bit complexity of Y, y, Z is similarly $\tilde{O}(1)$. (iv) The bit complexity of A_{ij} is $O(\log\frac{1}{\alpha}) = \tilde{O}(1)$. (v) The bit complexity of P_{ij} is $O(\log\frac{1}{\alpha^2}) = \tilde{O}(1)$. Since P has low bit-complexity, the rest of `AMMC` does by construction of the rounding algorithms.

Proof of correctness. We make use of the following lemma.

Lemma 4.8.3 (Robustness of `AMMC`)**.** *The following changes to `AMMC` affect the mean-weight $\bar{w}(\sigma)$ of the returned cycle σ by at most $\pm O(\varepsilon)$:*

(1) The entries of P are approximated to $\pm\alpha$ additive error and remain nonnegative.

(2) The nonzero entries of P are approximated to $[1\pm\alpha]$ multiplicative error.

(3) The nonzero entries of A are approximated to $[1\pm\alpha]$ multiplicative error.

Proof. The proof of item (1) is identical to the truncation in `ROUNDQCIRC` in Lemma 4.5.2. Item (2) then follows since $P_{ij} \leqslant 1$. Item (3) then follows since $P = A/(\sum_{ij} A_{ij})$. □

By the guarantee for `AMMC` in exact arithmetic (Theorem 4.6.1.A), it suffices to show that these modifications (i)-(v) affect $\bar{w}(\sigma)$ by at most $\pm O(\varepsilon)$. (i) and (ii) change A_{ij} by $[1\pm O(\alpha)]$ multiplicative error, which is acceptable by item (3) of Lemma 4.8.3. (iii) rescales A, which does not alter P. (iv) First we argue the effect of dropping all $A_{ij} < \alpha$ to 0. The only affected entries of P_{ij} are those dropped to 0; and since (iii) ensures $\sum_{i'j'} A_{i'j'} \geqslant \max_{i'j'} A_{i'j'} = 1$, thus $P_{ij} = A_{ij}/\sum_{i'j'} A_{i'j'}$ must have been at most α, so setting P_{ij} to 0 is acceptable by item (1) of Lemma 4.8.3. Next, we argue the truncation of A_{ij}. The $\pm\alpha^2$ additive precision of A_{ij} implies $[1\pm\alpha]$ multiplicative error for the nonzero entries of A (since they are at least α), which is acceptable by item (3) of Lemma 4.8.3. Finally, (v) is acceptable by item (1) of Lemma 4.8.3. □

■ 4.8.4 Reproducibility details for the experiments

Both the sparse and dense inputs used in §4.4 are generated in a three-step process á la the experimental survey [94]. First, the underlying graph is generated. For the dense graphs, this is an Erdös-Renyi random graph where each edge is included with probability 1/2 and has uniform random weights in $\{1, \ldots, 100\}$. For the sparse graphs, this is a random graph with $5n$ random edges and a random Hamiltonian cycle, again all with uniform random weights in $\{1, \ldots, 100\}$. Second, we plant a Hamiltonian cycle that has weight -1 on one edge, and weight 0 on the rest. This is the "subfamily 05" perturbation of [94]. It ensures that graph has a unique Minimum-Mean-Cycle and moreover that this optimal cycle is maximally long. Third, the planted Hamiltonian Cycle is hidden by randomly permuting the vertices and performing a "potential perturbation"; that is, adjusting $w(i,j) \mapsto w(i,j) + p_i - p_j$ where $p \in \mathbb{R}^n$ is a random vector with entries drawn uniformly from $\{1, \ldots, 200\}$. This potential perturbation does not affect the Minimum Mean Cycle. Finally, all edge weights are normalized to $[0, 1]$ via a simple shift and scaling.

Part II

Multimarginal OT and Tensor-Like Problems

Chapter 5

Hardness results for Multimarginal Optimal Transport

Whereas Part I of this thesis studies optimization problems over joint probability distributions with two constrained marginals, in Part II of this thesis we turn to the case of many constrained marginals. In the common setting of linear optimization, these problems are called Multimarginal Optimal Transport (MOT).

A key issue in applications of MOT is the complexity: the linear program has exponential size in the number of marginals k and their support sizes n. A recent line of work (including the results in the subsequent chapters) has shown that MOT is $\mathrm{poly}(n, k)$-time solvable for certain families of costs that have $\mathrm{poly}(n, k)$-size implicit representations. However, it is unclear what further families of costs this line of algorithmic research can encompass.

The purpose of this chapter is to understand these fundamental limitations by establishing the intractability of a number of MOT problems studied in the literature that have resisted previous algorithmic efforts. For instance, we demonstrate that in the absence of further structure, MOT is intractable even if the objective function decomposes into pairwise interactions between the variables, or if it decomposes in a low-rank manner with super-constant rank. As another example, we provide evidence that repulsive costs make MOT intractable by showing that several such problems of interest are NP-hard to solve. All of our hardness results hold even for approximate computation—and to the best of our knowledge, these are the first inapproximability results for any MOT problems. Together, these intractability results help guide the search for efficient MOT algorithms since they establish that these commonly occuring structural properties of MOT problems are, by themselves, insufficient for developing efficient algorithms.

5.1 Introduction

Multimarginal Optimal Transport (MOT) is the problem of linear programming over joint probability distributions with fixed marginal distributions. In this way, MOT generalizes the classical Kantorovich formulation of Optimal Transport from 2 marginal distributions to an arbitrary number $k \geqslant 2$ of them.

More precisely, an MOT problem is specified by a cost tensor C in the k-fold tensor product space $(\mathbb{R}^n)^{\otimes k} = \mathbb{R}^n \otimes \cdots \otimes \mathbb{R}^n$, and k marginal distributions $\mu_1, \ldots, \mu_k$ in the simplex $\Delta_n = \{v \in \mathbb{R}^n_{\geqslant 0} : \sum_{i=1}^n v_i = 1\}$.[1] The MOT problem is to compute

$$\min_{P \in \mathcal{M}(\mu_1, \ldots, \mu_k)} \langle P, C \rangle \tag{MOT}$$

where $\mathcal{M}(\mu_1, \ldots, \mu_k)$ is the "transportation polytope" consisting of all entrywise non-negative tensors $P \in (\mathbb{R}^n)^{\otimes k}$ satisfying the marginal constraints

$$\sum_{j_1, \ldots, j_{i-1}, j_{i+1}, \ldots, j_k} P_{j_1, \ldots, j_{i-1}, j, j_{i+1}, \ldots, j_k} = [\mu_i]_j$$

for all $i \in \{1, \ldots, k\}$ and $j \in \{1, \ldots, n\}$.

This MOT problem has recently attracted significant interest due to its many applications in data science, applied mathematics, and the natural sciences; see for instance [13, 32, 176, 178] and the many references within. However, a key issue that dictates the usefulness of MOT in applications is its complexity. Indeed, while MOT can be easily solved in $n^{O(k)}$ time since it is a linear program in n^k variables and $n^k + nk$ constraints, this is far from scalable.

In the literature as well as in this thesis, we are interested in "polynomial time" algorithms, where polynomial means in the number of marginals k and their support sizes n (as well as the scale-invariant quantity $C_{\max}/\varepsilon$ if we are considering ε additive approximations, where $C_{\max}$ denotes the maximum magnitude of the entries of C). An obvious obstacle is that in general, one cannot even read the input to MOT—let alone solve MOT—in $\mathrm{poly}(n, k)$ time since the cost tensor C has n^k entries.

Nevertheless, in nearly all applications of practical interest, the cost tensor C has a simple structure that enables it to be input implicitly via a $\mathrm{poly}(n, k)$-sized representation. Moreover, a recent line of work has shown that in many applications where C has such a polynomial-size implicit representation, the MOT

[1] For simplicity, all μ_i are assumed to have the same support size n. Everything in Part II of this thesis extends in a straightforward way to non-uniform sizes n_i where n^k is replaced by $\prod_{i=1}^k n_i$, and $\mathrm{poly}(n, k)$ is replaced by $\mathrm{poly}(\max_i n_i, k)$.

problem can also be solved in polynomial time. A simple-to-describe illustrative example is cost tensors C which have constant rank and are given as input in factored form [13]. Other examples include computing generalized Euler flows [13, 32], computing low-dimensional Wasserstein barycenters [11, 55], random combinatorial optimization [2, 111, 152, 159, 170, 229, 241], Distributionally Robust Optimization [61, 153, 160], solving MOT problems with tree-structured costs [108], and solving MOT problems with costs that have dynamic programming structure or certain combinatorial structure [13]. See Chapter 6 for further details on this algorithmic line of research.

A fundamental question is: What further families of succinctly representable costs lead to tractable MOT problems? Previously, hardness results were only known for exact computation and for MOT problems that either 1) have binary variables ($n = 2$) and supermodular costs [2], or 2) have costs which are related to combinatorial optimization problems, either of the form $C(x) = \min_{v \in V} \langle x, v \rangle$ or $C(x) = \mathbb{1}[\min_{v \in V} \langle x, v \rangle \leqslant t]$ for a polytope V [61, 135, 170, 241].[2]

But what about other problems? There are a number of other MOT problems studied in the literature for which C has a $\text{poly}(n, k)$-sized representation but developing $\text{poly}(n, k)$-time algorithms has resisted previous efforts. As an illustrative example, can MOT still be solved in $\text{poly}(n, k)$ time if the cost C has rank that is low but not constant, say of size $\text{poly}(n, k)$? The purpose of this chapter is to understand the fundamental limitations of this rapidly growing line of research centered around efficient MOT algorithms.

5.1.1 Contributions

In this chapter we establish the intractability of a number of MOT problems studied in the literature that have resisted previous algorithmic efforts, as described below. These intractability results help guide the search for efficient MOT algorithms since they establish that these common "structures" of MOT problems are, by themselves, insufficient for developing efficient algorithms. Moreover, there were no previous results for hardness of approximate computation for MOT; we provide the first such results.

Low-rank costs. In §5.4, we consider MOT problems with low-rank cost tensors given in factored form. Recent algorithmic work has shown that such MOT problems can be solved to arbitrary precision $\varepsilon > 0$ in $\text{poly}(n, k, C_{\max}/\varepsilon)$ time for any *fixed* rank r [13]. However, this algorithm's dependence on r is exponential, and it is a natural question whether such MOT problems can be solved in time

[2] These results are from the random combinatorial optimization literature, which has studied MOT under a different name and in a different community.

that is also polynomial in r. We show that unless $\mathsf{P} = \mathsf{NP}$, the answer is no. Moreover, our hardness result extends even to approximate computation. This provides a converse to the aforementioned algorithmic result.

Pairwise-interaction costs. In §5.5, we consider MOT problems with costs C that decompose into sums of pairwise interactions

$$C_{j_1,\dots,j_k} = \sum_{1 \leqslant i < i' \leqslant k} g_{i,i'}(j_i, j_{i'}) \tag{5.1}$$

for some functions $g_{i,i'} : [n] \times [n] \to \mathbb{R}$. This cost structure appears in many applications; for instance in Wasserstein barycenters [22] and the MOT relaxation of Density Functional Theory [52, 70]. Although these costs have $\mathrm{poly}(n, k)$-size implicit representations, we show that this cost structure alone is not sufficient for solving MOT in polynomial time.

One implication of this NP-hardness result is a converse to the algorithmic result of [13] which shows that MOT problems can be solved in $\mathrm{poly}(n, k)$ time for costs C which are decomposable into local interactions of low treewidth. This is a converse because the pairwise-interactions structure (5.1) also falls under the framework of MOT costs that decompose into local interactions, but has high treewidth.

Repulsive costs. In §5.6, we consider MOT problems with "repulsive costs". Informally, these are costs $C_{j_1,\dots,j_k}$ which encourage diversity between the indices $j_1, \dots j_k$; we refer the reader to the nice survey [80] for a detailed discussion of such MOT problems and their many applications. We provide evidence that repulsive costs lead to intractable MOT problems by proving that several such MOT problems of interest are NP-hard to solve. Specifically, in §5.6.1 we show this for MOT problems with the determinantal cost studied in [54, 80], and in §5.6.2 we show this for the popular MOT formulation of Density Functional Theory [33, 52, 70] with the Coulomb-Buckingham potential. Again, our hardness results extend even to approximate computation.

■ 5.1.2 Related work

Algorithms for MOT. The many applications of MOT throughout data science, mathematics, and the sciences at large have motivated a rapidly growing literature around developing efficient algorithms for MOT. Such algorithms are described in detail in Chapter 6. The purpose of this chapter is to understand fundamental limitations of this line of work.

Complexity of MOT value vs sparse solutions. Deciding whether MOT has a solution of sparsity exactly n is well-known to be NP-hard due to a connection with the NP-hard problem of hypergraph matching [91, 129]. This NP-hardness holds even in special cases such as (i) when the number of marginals $k = 3$ and the cost tensor C has $\{0, 1\}$-valued entries; and (ii) the Wasserstein barycenter problem in dimension $d = 2$ [46].

Nevertheless, in practice the goal is typically to find an MOT solution which is polynomially sparse (albeit perhaps not n-sparse), and this task is *not* necessarily NP-hard. Indeed, for the aforementioned MOT problems of type (i) and (ii), it is possible to compute $O(nk)$-sparse solutions in polynomial time [11, 13], so the existing NP-hardness results about computing n-sparse solutions do not give any indication about whether it is possible to find polynomially sparse solutions. In contrast, our results show that it is NP-hard to even compute the *value* of certain structured MOT problems—which also implies NP-hardness of computing solutions with any polynomial sparsity.

Other related work. We mention two tangentially related bodies of work in passing. First, the transportation polytope (a.k.a., the constraint set in the MOT problem) is an object of significant interest in discrete geometry and combinatorics, see e.g., [77, 236] and the references within. Second, linear programming problems over exponentially-sized joint probability distributions appear in various fields such as game theory [174] and variational inference [226]. However, it is important to note that the complexity of these linear programming problems is heavily affected by the specific linear constraints, which often differ between problems in different fields.

5.2 Preliminaries

Notation. For the convenience of the reader, here we collect notation used commonly throughout the chapter. The k-fold tensor product space $\mathbb{R}^n \otimes \cdots \otimes \mathbb{R}^n$ is denoted by $(\mathbb{R}^n)^{\otimes k}$, and similarly for $(\mathbb{R}^n_{\geqslant 0})^{\otimes k}$. The set $\{1, \ldots, n\}$ is denoted by $[n]$. The i-th marginal of a tensor $P \in (\mathbb{R}^n)^{\otimes k}$ is the vector $m_i(P) \in \mathbb{R}^n$ with j-th entry $\sum_{j_1,\ldots,j_{i-1},j,j_{i+1},\ldots,j_k} P_{j_1,\ldots,j_{i-1},j,j_{i+1},\ldots,j_k}$, for $i \in [k]$ and $j \in [n]$. In this notation, the transportation polytope in (MOT) is $\mathcal{M}(\mu_1, \ldots, \mu_k) = \{P \in (\mathbb{R}^n_{\geqslant 0})^{\otimes k} : m_i(P) = \mu_i,\ \forall i \in [k]\}$. For shorthand, we often denote an index $(j_1, \ldots, j_k)$ by $\vec{j}$. For a tensor $C \in (\mathbb{R}^n)^{\otimes k}$, we denote the maximum absolute value of its entries by $C_{\max} = \max_{\vec{j}} |C_{\vec{j}}|$; similarly for a matrix $p \in \mathbb{R}^{n \times k}$, we denote the maximum absolute value of its entries by $p_{\max}$.

MOT dual. Since MOT is an LP in standard form, the dual LP to (MOT) is

$$\max_{q_1,\dots,q_k\in\mathbb{R}^n} \sum_{i=1}^{k}\langle q_i,\mu_i\rangle \quad \text{subject to} \quad C_{\vec{j}} - \sum_{i=1}^{k}[q_i]_{j_i} \geqslant 0,\ \ \forall \vec{j}\in[n]^k. \quad \text{(MOT-D)}$$

Bit complexity. Throughout, we assume for simplicity of exposition that all entries of the cost C and the weights $p \in \mathbb{R}^{nk}$ inputted in the MIN_C problem have bit complexity at most $\text{poly}(n,k)$. This implies that the distributions μ on which the MOT_C oracle is queried in Theorems 5.3.2 and 5.3.3 also have polynomial bit complexity. The general case is a straightforward extension.

Computational complexity. BPP is the class of problems solvable by polynomial-time randomized algorithms with error probability that is $< 1/3$ (or equivalently, any constant less than $1/2$). The statement "NP $\not\subset$ BPP" is a standard assumption in computational complexity and is the *randomized* version of P $\neq$ NP, i.e., that NP-hard problems do not have polynomial-time randomized algorithms.

■ 5.3 Reducing MIN to MOT

Here we present a basic toolkit that our intractability results build upon. This toolkit enables proving intractability results for MOT by instead proving intractability results for MIN_C, defined below. This is helpful since MIN_C is more directly amenable to NP-hardness and inapproximability reductions because it is phrased as a more conventional combinatorial optimization problem; examples in §5.4, §5.5, and §5.6.

Definition 5.3.1. *For $C \in (\mathbb{R}^n)^{\otimes k}$ and $p = (p_1,\dots,p_k) \in \mathbb{R}^{n\times k}$, the problem $\mathsf{MIN}_C(p)$ is to compute*

$$\min_{(j_1,\dots,j_k)\in\{1,\dots,n\}^k} C_{j_1,\dots,j_k} - \sum_{i=1}^{k}[p_i]_{j_i}. \tag{5.2}$$

In words, MIN_C is the feasibility oracle for the dual to the MOT LP because $p = (p_1,\dots,p_k) \in \mathbb{R}^{n\times k}$ is feasible for (MOT-D) if and only if the problem $\mathsf{MIN}_C(p)$ has non-negative value.

In what follows, let $\mathsf{MOT}_C(\mu)$ denote the problem of computing the optimal value of MOT for a cost tensor $C \in (\mathbb{R}^n)^{\otimes k}$ and marginals $\mu = (\mu_1,\dots,\mu_k) \in (\Delta_n)^k$. In words, the following two theorems establish that for any fixed cost C, the following discrete optimization problem MIN_C can be (approximately) solved in polynomial time if MOT_C can be (approximately) solved in polynomial time. The first reduction is used for proving NP-hardness of exactly solving MOT_C, and the

second reduction is used for proving inapproximability. These two reductions are incomparable.

Theorem 5.3.2 (Exact reduction)**.** *There is a deterministic algorithm that, given access to an oracle solving MOT_C and weights $p \in \mathbb{R}^{n\times k}$, solves $\mathsf{MIN}_C(p)$ in* $\mathrm{poly}(n,k)$ *time and oracle queries.*

Theorem 5.3.3 (Approximate reduction)**.** *There is a randomized algorithm that, given $\varepsilon > 0$, access to an oracle solving MOT_C to additive accuracy ε, and weights $p \in \mathbb{R}^{n\times k}$, solves $\mathsf{MIN}_C(p)$ up to $\varepsilon\cdot\mathrm{poly}(n,k)$ additive accuracy with probability* $2/3$ *in* $\mathrm{poly}(n,k,\frac{C_{\max}+p_{\max}}{\varepsilon})$ *time and oracle queries.*

The proofs of these theorems can be found in the paper [12] upon which this chapter is based, or alternatively can be seen as special cases of the more general results in [141][3].

We conclude this section with several remarks about these two theorems.

Remark 5.3.4 (Converse)**.** *There is no loss of generality when using Theorems 5.3.2 and 5.3.3 to reduce proving the intractability of MOT_C to proving the intractability of MIN_C. This is because the MOT_C and MIN_C problems are* polynomial-time equivalent*—for any cost C, and for both exact and approximate solving—because the converse of this reduction also holds (see §6.3).*

Remark 5.3.5 (Value vs solution for MOT)**.** *A desirable feature of using Theorems 5.3.2 and 5.3.3 to prove intractability of MOT is that the resulting intractability applies regardless of how an MOT* solution *is computed and (compactly) represented[4]. This is because such an argument shows hardness for (approximately) computing the optimal* value *of MOT.*

Remark 5.3.6 (Value vs solution for MIN)**.** *Theorems 5.3.2 and 5.3.3 hold unchanged if the MIN_C problem is modified to require computing the minimizing tuple rather than the minimum value in* (5.2)*. This is because these two problems are polynomial-time equivalent [13, Appendix A.1].*

[3]We are grateful to Theodor Misiakiewicz who pointed this out to us after the publication of the paper [12].

[4]Indeed, the representations produced by MOT algorithms often vary: e.g., the solution is polynomially-sparse for the Ellipsoid and Multiplicative Weights algorithms; and is fully dense but has a polynomial-size representation which supports certain efficient operations for the Sinkhorn algorithm. See [13] for details.

Remark 5.3.7 (Inapproximability reductions are probabilistic)**.** *Inapproximability reductions using Theorem 5.3.3 are probabilistic*[5]*, and thus show inapproximability under the standard complexity assumption* NP $\not\subset$ BPP*, which is informally the stronger "randomized version" of* P $\neq$ NP*.*

Remark 5.3.8 ($p = 0$)**.** *The* $\mathsf{MIN}_C(0)$ *problem is to compute the minimum entry of the tensor* C*. Thus, as a special case of our reductions, it follows that if (approximately) computing the minimum entry of* C *is* NP*-hard, then so is (approximately) computing* MOT_C*. In fact, in our applications in §5.4, §5.5, and §5.6, we prove intractability of* MIN_C*—and thus of* MOT_C*—by showing intractability for this "simple" case* $p = 0$*. However, we mention that in general one cannot restrict only to the case* $p = 0$*: In §5.7, we give a concrete example where* MIN_C *is tractable for* $p = 0$ *but not general* p*.*

Remark 5.3.9 (Extension to regularized MOT)**.** *An entropically regularized version of* MOT *is also of interest in the literature due to its statistical and algorithmic properties; see, e.g., [13, 32, 35, 108, 110, 162, 178], among many others. Our results also extend to this regularized* MOT *problem; for details see the Appendix of the paper [12] upon which this chapter is based.*

■ 5.4 Application: costs with super-constant rank

In direct analogy to the standard concept of the rank of a matrix, a rank-r factorization of a tensor $C \in (\mathbb{R}^n)^{\otimes k}$ is a set of rk vectors $\{u^{(i,\ell)}\}_{i\in[k],\ell\in[r]} \subset \mathbb{R}^n$ satisfying

$$C_{j_1,\dots,j_k} = \sum_{\ell=1}^{r} \prod_{i=1}^{k} u_{j_i}^{(i,\ell)}, \qquad \forall (j_1,\dots,j_k) \in [n]^k. \tag{5.3}$$

The rank of a tensor is the minimal r for which there exists a rank-r factorization. Note that the rank of a tensor in $(\mathbb{R}^n)^{\otimes k}$ can be exponentially large in k. We refer the reader to the detailed survey [133] for background on tensor rank as well as a historical overview.

In the context of MOT, low-rank cost tensors appear, for example, in the Wasserstein barycenter problem and in the problem of projecting a mixture distribution to the transportation polytope [13]. Recent work has given a polynomial time algorithm for approximate MOT when the cost is a constant-rank tensor

[5]It is an interesting question whether the approximate reduction in Theorem 5.3.3 can be de-randomized. This would enable showing our inapproximability results under the assumption P $\neq$ NP rather than NP $\not\subset$ BPP.

given in factored form [13]. A natural algorithmic question is whether the dependence on the rank can be improved: is there an algorithm whose runtime is simultaneously polynomial in n, k, and the rank r? Here we show that, under standard complexity theory assumptions, the answer is no. Our result provides a converse to [13], and justifies the constant-rank regime studied in [13].

Proposition 5.4.1 (Hardness of MOT for low-rank costs)**.** *Assuming* $\mathsf{P} \neq \mathsf{NP}$*, there is no* $\mathrm{poly}(n, k, r)$*-time deterministic algorithm for solving* MOT_C *for costs* C *given by a rank-*r *factorization.*

Our impossibility result further extends to approximate computation.

Proposition 5.4.2 (Hardness of approximate MOT for low-rank costs)**.** *Assuming* $\mathsf{NP} \not\subset \mathsf{BPP}$*, there is no* $\mathrm{poly}(n, k, r, \frac{C_{\max}}{\varepsilon})$*-time randomized algorithm for approximating* MOT_C *to* ε *additive accuracy for costs* C *given by a rank-*r *factorization.*

The proof encodes the hard problem of finding a large clique in a k-partite graph as an instance of MOT_C in which C has an explicit low-rank factorization. We define the following notation. We say a k-partite graph on nk vertices is balanced if each of the k parts has n vertices. For a balanced k-partite graph G on nk vertices $v_{i,j}$ for $i \in [k]$ and $j \in [n]$, let $T_G \in (\mathbb{R}^n)^{\otimes k}$ denote the tensor with $(j_1, \ldots, j_k)$-th entry equal to the number of edges in the induced subgraph of G with vertices $\{v_{1,j_1}, \ldots, v_{k,j_k}\}$.

Lemma 5.4.3 (T_G is low-rank)**.** *For any balanced* k*-partite graph* G *on* nk *vertices,* $\mathrm{rank}(T_G) \leqslant n^2k^2$*. Moreover, a factorization of* T_G *with this rank is computable from* G *in* $\mathrm{poly}(n, k)$ *time.*

Proof. Consider an edge $(v_{i,j_i}, v_{i',j_{i'}})$ between partitions $i, i' \in [k]$. Consider the rank-1 tensor formed by the outer product of the indicator vectors e_{j_i} and $e_{j_{i'}}$ on respective slices i and i', and the all-ones vector $\mathbf{1}_n$ on all other slices $\ell \in [k] \setminus \{i, i'\}$. This tensor takes value 1 on all tuples in $[n]^k$ with i-th coordinate j_i and i'-th coordinate $j_{i'}$, and takes value 0 elsewhere. Summing up such a rank-1 tensor for each edge of G—of which there are at most $(nk)^2$—yields the desired factorization. □

Lemma 5.4.4 (Hardness of MIN for low-rank costs)**.** *Assuming* $\mathsf{P} \neq \mathsf{NP}$*, there is no* $\mathrm{poly}(n, k, r)$*-time deterministic algorithm for solving* $\mathsf{MIN}_C(0)$ *for costs* C *given by a rank-*r *factorization. Moreover, assuming* $\mathsf{NP} \not\subset \mathsf{BPP}$*, there is no* $\mathrm{poly}(n, k, r, \frac{C_{\max}}{\varepsilon})$*-time randomized algorithm for* ε*-approximate additive computation.*

Proof. It is a folklore[6] fact that deciding whether there exists a k-clique in a balanced k-partite graph G on nk vertices is NP-hard. This problem reduces to computing the maximal entry in T_G, which is equivalent to solving $\mathsf{MIN}_C(0)$ for $C = -T_G$. The first statement then follows since a low-rank factorization of $-T_G$ can be found in $\mathrm{poly}(n, k)$ time by Lemma 5.4.3. For the second statement, note that since the entries of $-T_G$ are integral, it is also NP-hard to solve $\mathsf{MIN}_C(0)$ to additive error $C_{\max}/10k^2 \leqslant 0.1$. □

Proof of Proposition 5.4.1. By Theorem 5.3.2, a $\mathrm{poly}(n, k, r)$-time deterministic algorithm for MOT_C on rank-r costs implies a $\mathrm{poly}(n, k, r)$-time deterministic algorithm for $\mathsf{MIN}_C(0)$ on rank-r costs. Assuming P ≠ NP, this contradicts Lemma 5.4.4. □

Proof of Proposition 5.4.2. By Theorem 5.3.3, a $\mathrm{poly}(n, k, r, \frac{C_{\max}}{\varepsilon})$-time randomized algorithm for MOT_C on rank-r costs C implies a $\mathrm{poly}(n, k, r, \frac{C_{\max}}{\varepsilon})$-time randomized algorithm for $\mathsf{MIN}_C(0)$ on rank-r costs C. Assuming NP ⊄ BPP, this contradicts Lemma 5.4.4. □

■ 5.5 Application: costs with full pairwise interactions

Many studied MOT costs, such as the Wasserstein barycenter cost and Coulomb cost, have the following structure: they decompose into a sum of pairwise interactions, as

$$C_{j_1,\dots,j_k} = \sum_{1 \leqslant i < i' \leqslant k} g_{i,i'}(j_i, j_{i'}) \tag{5.4}$$

for some functions $g_{i,i'} : [n] \times [n] \to \mathbb{R}$. This decomposability structure allows for a polynomial-size implicit representation of the cost tensor, and has a variety of applications due to intimate connections with probabilistic graphical models [110]. It is a natural question whether this generic structure can be exploited to obtain polynomial-time algorithms for MOT. We show that the answer is no: there are MOT costs that are decomposable into pairwise interactions, but are NP-hard to solve.

Proposition 5.5.1 (Hardness of MOT for pairwise-decomposable costs)**.** *Assuming* P ≠ NP*, there is no* $\mathrm{poly}(n, k)$*-time deterministic algorithm for solving* MOT_C *for costs* C *of the form* (5.4)*.*

[6] This follows from the fact that finding a k-clique in an n-vertex graph G' is NP-hard [129]. Given G', a balanced k-partite graph G on nk vertices can be constructed in polynomial time by letting vertices $v_{i,j}$ and $v_{i',j'}$ be adjacent in G if and only if $j \neq j'$ and the vertices v'_i and $v'_{i'}$ are adjacent in G'. Under this construction, G has a k-clique if and only if G' has a k-clique.

Our impossibility result further extends to approximate computation.

Proposition 5.5.2 (Hardness of approximate MOT for pairwise-decomposable costs)**.** *Assuming* NP $\not\subset$ BPP*, there is no* $\mathrm{poly}(n, k, \frac{C_{\max}}{\varepsilon})$*-time randomized algorithm for approximating* MOT_C *to* ε *additive accuracy for costs* C *of the form* (5.4)*.*

Proof of Propositions 5.5.1 and 5.5.2. The proofs of Propositions 5.5.1 and 5.5.2 are the same as the proofs of Propositions 5.4.1 and 5.4.2 using the fact that for any graph $G = (V, E)$, the tensor $-T_G$ can be written as a sum of pairwise interactions: $(-T_G)_{j_1,\dots,j_k} = \sum_{1 \leqslant i < i' \leqslant k} -\mathbb{1}[(v_{i,j_i}, v_{i',j_{i'}}) \in E]$. □

Propositions 5.5.1 and 5.5.2 provides converses to the result of [13]. Specifically, [13, §4], considers MOT costs C that decompose into local interactions as $C_{j_1,\dots,j_k} = \sum_{S \in \mathcal{S}} g_S(\{j_i\}_{i \in S})$, and gives a polynomial-time algorithm in the case that the graph with vertices $[k]$ and edges $\{(i, i') : i, i' \in S \text{ for some } S \in \mathcal{S}\}$ has constant treewidth. Conversely, our hardness results in Propositions 5.4.1 and 5.4.2 show that bounded treewidth is necessary for polynomial-time algorithms. This is because costs of the form (5.4) fall under the framework of graphically structured costs in [13, 110] with non-constant treewidth of size $k - 1$.

■ 5.6 Application: repulsive costs

In this section, we investigate several MOT problems with repulsive costs that are of interest in the literature. We prove intractability results that clarify why—despite a growing literature (see, e.g., the survey [80] and the references within)—these problems have resisted algorithmic progress.

■ 5.6.1 Application to determinantal cost

A repulsive cost of interest in the MOT literature and in pure mathematics is the determinant cost (e.g., [54, 80]). This cost is given by:

$$C_{j_1,\dots,j_k} = -|\det(x_{j_1}, \dots, x_{j_k})|, \tag{5.5}$$

where $x_1, \dots, x_n \in \mathbb{R}^k$ and $\det(x_{j_1}, \dots, x_{j_k})$ is the determinant of the $k \times k$ matrix whose columns are $x_{j_1}, \dots, x_{j_k}$. This is a repulsive cost in the sense that tuples with "similar" vectors are penalized with higher cost, see the survey [80]. We prove that the MOT problem with this cost is NP-hard. For convenience of notation, we think of the marginal distributions $\mu_1, \dots, \mu_k$ as distributions in the simplex Δ_n, and write $[\mu_i]_j$ to mean the mass of μ_i on x_j.

Proposition 5.6.1 (Hardness of MOT with determinant cost)**.** *Assuming* $\mathsf{P} \neq \mathsf{NP}$*, there is no* $\mathrm{poly}(n,k)$*-time algorithm that given* $x_1,\dots,x_n \in \mathbb{R}^k$ *solves* MOT_C *for the cost* C *in* (5.5)*.*

Proof. By Theorem 5.3.2, it suffices to prove that the MIN_C problem is NP-hard. We show this is true even if the input weights p are identically 0: in this case the MIN_C problem is to compute $\min_{\vec{j}} C_{\vec{j}} = -\max_{\vec{j}} |\det(x_{j_1},\dots,x_{j_k})|$ given $x_1,\dots,x_n \in \mathbb{R}^k$. This is NP-hard by [173]. □

Rather than show additive inapproximability of MOT with determinant costs, we consider log-determinant costs since additive error on the logarithmic scale amounts to multiplicative error on the natural scale, which is more standard in the combinatorial-optimization literature on determinant maximization. Below, we show inapproximability of MOT with such log-determinant costs. Note that for technical reasons we upper-bound the cost at 0 to avoid unbounded costs for tuples with null determinant:

$$C_{j_1,\dots,j_k} = \min(0, -\log|\det(x_{j_1},\dots,x_{j_k})|). \tag{5.6}$$

Proposition 5.6.2 (Approximation hardness of MOT with log-determinant cost)**.** *Assuming* $\mathsf{NP} \not\subset \mathsf{BPP}$*, there is no* $\mathrm{poly}(n,k,C_{\max}/\varepsilon)$ *time algorithm that given* $x_1,\dots,x_n \in \mathbb{R}^k$ *approximates* MOT_C *to* ε *additive accuracy for the cost* C *in* (5.6)*.*

Proof. Let $x_1,\dots,x_n \in \mathbb{Z}^k$ have $\mathrm{poly}(n,k)$ bits each. It is known that

$$\min_{\vec{j}\in[n]^k} -\log|\det(x_{j_1},\dots,x_{j_k})|$$

is NP-hard to compute within additive error 0.0001 [211, Theorem 3.2]. Since $x_1,\dots,x_n$ span $\mathbb{R}^k$ without loss of generality, this is equivalent to approximating $\min_{j_1,\dots,j_k} C_{j_1,\dots,j_k}$ to within additive error 0.0001. But by Theorem 5.3.3, given access to MOT_C computations with additive accuracy $C_{\max}/\mathrm{poly}(n,k)$, we can approximate $\mathsf{MIN}_C(0) = \min_{j_1,\dots,j_k} C_{j_1,\dots,j_k}$ to within additive error 0.0001 in $\mathrm{poly}(n,k)$ randomized time since $C_{\max}$ is of $\mathrm{poly}(n,k)$ size here. Hence, assuming $\mathsf{BPP} \not\subset \mathsf{NP}$, there is no $\mathrm{poly}(n,k,C_{\max}/\varepsilon)$-time algorithm that solves MOT_C to accuracy ε. □

■ 5.6.2 Application to Density Functional Theory

A popular application of MOT is to formulate a relaxation of the Density Functional Theory problem (DFT) from quantum chemistry. We refer the reader to [70] for an introduction of the MOT formulation of DFT, and sketch the simplest case

below. In the simplest version of the MOT relaxation, we are given k distributions corresponding to k electron clouds in space, and the objective is to couple the electron clouds in a way that minimizes the expected potential energy of the electron configuration. Suppose the electron clouds are given as distributions $\mu_1, \dots, \mu_k$ supported on $x_1, \dots, x_n \in \mathbb{R}^3$; again, for convenience of notation, we think of $\mu_1, \dots, \mu_k$ as distributions in the simplex Δ_n, and write $[\mu_i]_j$ to mean the mass of μ_i on x_j. Then, the MOT relaxation of DFT is to compute a minimum-cost coupling of $\mu_1, \dots, \mu_k$, with cost given by the Coulomb potential

$$C_{j_1,\dots,j_k} = \sum_{1 \leqslant i < i' \leqslant k} \frac{1}{\|x_{j_i} - x_{j_{i'}}\|_2}. \tag{5.7}$$

This is a repulsive cost that encourages tuples $(j_1, \dots, j_n) \in [n]^k$ such that $x_{j_1}, \dots, x_{j_n}$ are spread as far as possible, since the Coulomb potential decreases as two electrons move farther apart. Despite significant algorithmic interest, provable polynomial-time algorithms have not yet been found. We conjecture that in fact solving MOT with the Coulomb potential is NP-hard.

Conjecture 5.6.3. *Assuming* P $\neq$ NP*, there is no* $\mathrm{poly}(n,k)$*-time algorithm solving* MOT_C *with the Coulomb potential cost* (5.7)*.*

In this section, we make progress towards the conjecture by proving hardness of DFT with the related Coulomb-Buckingham potential, which is similar to the Coulomb potential, but has extra energy terms that grow as $1/r^6$ and $\exp(-\Theta(r))$. The Coulomb-Buckingham potential is popular for modeling the structures of ionic crystals [1], and is defined for two particles at distance r with charges $q_1, q_2 \in \{-1, +1\}$ as:

$$U(r, q_1, q_2) = \begin{cases} M, & r = 0 \\ \frac{A_{q_1 q_2}}{\exp(B_{q_1 q_2} r)} - \frac{C_{q_1 q_2}}{r^6} + \frac{q_1 q_2}{r}, & r > 0 \end{cases},$$

where $A_{+1}, A_{-1}, B_{+1}, B_{-1}, C_{+1}, C_{-1}$ are constants determining the relative strengths of the terms in the interaction, and $M > 0$ is a large constant (that should be intuitively thought of as infinite) penalizing two ions being in the same place. Given ions with charges $q_j \in \{-1, +1\}$ at positions $x_j \in \mathbb{R}^3$, the corresponding MOT cost is given by:

$$C_{j_1,\dots,j_k} = \begin{cases} M, & \sum_{i \in [k]} q_{j_i} \neq 0 \\ \sum_{1 \leqslant i < i' \leqslant k} U(\|x_{j_i} - x_{j_{i'}}\|_2, q_{j_i}, q_{j_{i'}}), & \sum_{i \in [k]} q_{j_i} = 0 \end{cases}. \tag{5.8}$$

Proposition 5.6.4 (Hardness of DFT with Coulomb-Buckingham potential)**.** *Assuming* $\mathsf{P} \neq \mathsf{NP}$*, there is no* $\mathrm{poly}(n,k)$*-time algorithm that, given positions* $x_1,\dots,x_n \in \mathbb{R}^3$*, charges* $q_1,\dots,q_n \in \{-1,+1\}$*, parameters* $A_{\pm 1}, B_{\pm 1}, C_{\pm 1}, M > 0$*, and marginals* $\mu_1,\dots,\mu_k \in \Delta_n$*, solves* MOT_C *with cost* C *given by* (5.8)*.*

Proof. For the proof, we show NP-hardness even if the inputs $x_1,\dots,x_n \in \mathbb{R}^3$ are such that $\min_{1\leqslant j'\leqslant j\leqslant n} \|x_j - x_{j'}\|_2 \geqslant 1$, $A_{\pm 1}, B_{\pm 1}, C_{\pm 1} \leqslant \mathrm{poly}(n,k)$, and $2k^2(2 + A_{+1} + A_{-1} + C_{+1} + C_{-1}) \leqslant M \leqslant \mathrm{poly}(n,k)$. In the parameter regime above, we have $C_{\max} = M \leqslant \mathrm{poly}(n,k)$, so by Theorem 5.3.2 and 5.3.3, it suffices to show that computing $\mathsf{MIN}_C(0)$ is NP-hard.

Furthermore, in the parameter regime above, $\mathsf{MIN}_C(0)$ is equal to the following:

$$\min\left\{\frac{1}{2}\sum_{j\in S, j'\in S\setminus\{j\}} U(\|x_j - x_{j'}\|_2, q_j, q_{j'}) : S \subset [n], |S| = k, \sum_{j\in S} q_j = 0\right\} \tag{5.9}$$

This optimization problem is NP-hard by [1, Theorem 5]. □

A similar hardness result also holds for approximate computation, stated next.

Proposition 5.6.5 (Approximation hardness of DFT with Coulomb-Buckingham potential)**.** *If* $\mathsf{NP} \not\subset \mathsf{BPP}$*, there is no* $\mathrm{poly}(n, k, C_{\max}/\varepsilon)$*-time algorithm computing an* ε*-additive approximation to* MOT_C*, where* C *is as in Proposition 5.6.4.*

The proof of Proposition 5.6.5 is identical to the proof of Proposition 5.6.4 once we show that the $\mathsf{MIN}_C(0)$ problem is hard to solve approximately. While [1, Theorem 5] only shows hardness of exactly computing $\mathsf{MIN}_C(0)$, a slightly more careful analysis extends this hardness to approximate computation; details are deferred to the Appendix of the paper [12] upon which this chapter is based.

■ 5.7 Necessity of dual weights

This section fleshes out the details for Remark 5.3.8. Namely, in §5.4, §5.5, and §5.6, we showed that MOT_C was hard to compute for some family of costs C by proving that $\mathsf{MIN}_C(0)$ was hard to compute. Here, we show that such arguments do not use the full power of Theorems 5.3.2 and 5.3.3: we construct a family of cost tensors C for which MOT_C is NP-hard to compute yet $\mathsf{MIN}_C(0)$ is polynomial-time computable.

The cost family is as follows: given a 2-SAT formula $\phi : \{0,1\}^k \to \{0,1\}$, define

$$C_{j_1,\dots,j_k} = -\phi(j_1,\dots,j_k). \tag{5.10}$$

Proposition 5.7.1. *Given a 2-SAT formula ϕ, it is* NP*-hard to solve* MOT_C *for the cost* (5.10)*. However,* $\mathit{MIN}_C(0)$ *can be computed in polynomial time.*

Proof. Observe that $\mathsf{MIN}_C(0) = \min_{j_1,\ldots,j_k} C_{j_1,\ldots,j_k} = -\max_{j_1,\ldots,j_k} \phi(j_1,\ldots,j_k)$ is the satisfiability problem for ϕ, which can be solved in polynomial-time since ϕ is a 2-SAT formula [106].

On the other hand, let $p = (p_1,\ldots,p_k) \in \mathbb{R}^{2\times k}$ be given by $p_1 = p_2 = \cdots = p_k = (0, -1/(2k)) \in \mathbb{R}^2$. Then $\mathsf{MIN}_C(p) = \min_{j_1,\ldots,j_k} -\phi(j_1,\ldots,j_k) - \sum_{i=1}^k [p_i]_{j_i} = -[\max_{j_1,\ldots,j_k} \phi(j_1,\ldots,j_k) - \|\vec{j}\|_1/(2k)]$ solves the problem of finding the minimum weight of a satisfying assignment to ϕ. This problem is NP-hard [106], hence $\mathsf{MIN}_C(p)$ is NP-hard. Therefore MOT_C is NP-hard by Theorem 5.3.2. □

Chapter 6

Polynomial-time algorithms for Multimarginal Optimal Transport problems with structure

The previous chapter established that MOT is intractable even in a number of commonly occuring "structured" settings. That is, under standard complexity theoretic assumptions, there is no algorithm that runs in time which is polynomial in the number of marginals k and their support sizes n. This chapter complements those intractability results by developing a general theory about what structure makes MOT solvable in $\text{poly}(n, k)$ time.

Specifically, this chapter develops a unified algorithmic framework for solving MOT in $\text{poly}(n, k)$ time by characterizing the structure that different algorithms require in terms of simple variants of the dual feasibility oracle. This framework has several benefits. First, it enables us to show that the Sinkhorn algorithm, which is currently the most popular MOT algorithm, requires strictly more structure than other algorithms do to solve MOT in $\text{poly}(n, k)$ time. Second, our framework makes it much simpler to develop $\text{poly}(n, k)$ time algorithms for a given MOT problem. In particular, it is necessary and sufficient to (approximately) solve the dual feasibility oracle—which is much more amenable to standard algorithmic techniques.

We illustrate this ease-of-use by developing $\text{poly}(n, k)$-time algorithms for three general classes of MOT cost structures: (1) graphical structure; (2) set-optimization structure; and (3) low-rank plus sparse structure. For structure (1), we recover the known result that Sinkhorn has $\text{poly}(n, k)$ runtime [110, 216]; moreover, we provide the first $\text{poly}(n, k)$ time algorithms for computing solutions that are exact and sparse. For structures (2)-(3), we give the first $\text{poly}(n, k)$ time algorithms, even for approximate computation. Together, these three structures encompass many—if not most—current applications of MOT.

6.1 Introduction

Recall from Chapter 5 that MOT is the problem of linear programming over joint probability distributions with fixed marginal distributions. That is, given k marginal distributions $\mu_1, \dots, \mu_k$ in the simplex $\Delta_n = \{u \in \mathbb{R}^n_{\geqslant 0} : \sum_{i=1}^n u_i = 1\}$ and a cost tensor C in the k-fold tensor product space $(\mathbb{R}^n)^{\otimes k} = \mathbb{R}^n \otimes \cdots \otimes \mathbb{R}^n$, compute

$$\min_{P \in \mathcal{M}(\mu_1, \dots, \mu_k)} \langle P, C \rangle \tag{MOT}$$

where $\mathcal{M}(\mu_1, \dots, \mu_k)$ is the "transportation polytope" containing entrywise non-negative tensors $P \in (\mathbb{R}^n)^{\otimes k}$ satisfying the marginal constraints

$$\sum_{j_1, \dots, j_{i-1}, j_{i+1}, \dots, j_k} P_{j_1, \dots, j_{i-1}, j, j_{i+1}, \dots, j_k} = [\mu_i]_j$$

for all $i \in \{1, \dots, k\}$ and $j \in \{1, \dots, n\}$.

This MOT problem has many applications throughout machine learning, computer science, and the natural sciences since it arises in tasks that require "stitching" together aggregate measurements. For instance, applications of MOT include inference from collective dynamics [86, 110], information fusion for Bayesian learning [208], averaging point clouds [3, 74], the n-coupling problem [191], quantile aggregation [149, 190], matching for teams [53, 65], image processing [183, 206], random combinatorial optimization [2, 111, 152, 159, 170, 229, 241], Distributionally Robust Optimization [61, 153, 160], simulation of incompressible fluids [35, 50], and Density Functional Theory [33, 52, 70].

However, as discussed in Chapter 5, in most applications the success of MOT is severely limited by the lack of efficient algorithms. Indeed, in general, MOT requires *exponential time* in the number of marginals k and their support sizes n. For instance, applying a linear program solver out-of-the-box takes $n^{\Theta(k)}$ time because MOT is a linear program with n^k variables, n^k non-negativity constraints, and nk equality constraints. Specialized algorithms in the literature such as the Sinkhorn algorithm yield similar $n^{\Theta(k)}$ runtimes. Such runtimes currently limit the applicability of MOT to tiny-scale problems (e.g., $n = k = 15$).

Polynomial-time algorithms for MOT. This chapter develops *polynomial-time* algorithms for MOT, where here and henceforth "polynomial" means in the number of marginals k and their support sizes n—and possibly also $C_{\max}/\varepsilon$ for ε-additive approximation, where $C_{\max}$ is a bound on the entries of C.

At first glance, this may seem impossible for at least two "trivial" reasons. One is that it takes exponential time to read the input cost C since it has n^k

entries. We circumvent this issue by considering costs C with $\text{poly}(n, k)$-size implicit representations, which encompasses essentially all MOT applications.[1] A second obvious issue is that it takes exponential time to write the output variable P since it has n^k entries. We circumvent this issue by returning solutions P with $\text{poly}(n, k)$-size implicit representations, for instance sparse solutions.

But, of course, circumventing these issues of input/output size is not enough to actually solve MOT in polynomial time. See Chapter 5 for examples of NP-hard MOT problems with costs that have $\text{poly}(n, k)$-size implicit representations.

Remarkably, for several MOT problems, there are specially-tailored algorithms that run in polynomial time—notably, for MOT problems with graphically-structured costs of constant treewidth [108, 110, 216], variational mean-field games [34], certain random combinatorial optimization problems [2, 111, 152, 159, 170, 229, 241], computing generalized Euler flows [32], computing low-dimensional Wasserstein barycenters [11, 32, 55], and filtering and estimation tasks based on target tracking [86, 107, 108, 110, 201]. However, the number of MOT problems that are known to be solvable in polynomial time is small, and it is unknown if these techniques can be extended to the many other MOT problems arising in applications. This motivates the central question driving this chapter:

Are there general "structural properties" that make MOT *solvable in* $\text{poly}(n, k)$ *time?*

This chapter is conceptually divided into two parts. In the first part of the chapter, we develop a unified algorithmic framework for MOT that characterizes the structure required for different algorithms to solve MOT in $\text{poly}(n, k)$ time, in terms of simple variants of the dual feasibility oracle. This enables us to prove that some algorithms can solve MOT problems in polynomial time whenever any algorithm can; whereas the popular `SINKHORN` algorithm cannot. Moreover, this algorithmic framework makes it significantly easier to design a $\text{poly}(n, k)$ time algorithm for a given MOT problem (when possible) because it now suffices to solve the dual feasibility oracle—and this is much more amenable to standard algorithmic techniques. In the second part of the chapter, we demonstrate the ease-of-use of our algorithmic framework by applying it to three general classes of MOT cost structures.

Below, we detail these two parts of the chapter in §6.1.1 and §6.1.2, respectively.

[1] E.g., in the MOT problems of Wasserstein barycenters, generalized Euler flows, and Density Functional Theory, C has entries $C_{j_1,\dots,j_k} = \sum_{i,i'=1}^{k} g_{i,i'}(j_i, j_{i'})$ and thus can be implicitly input via the k^2 functions $g_{i,i'} : \{1, \dots, n\}^2 \to \mathbb{R}$.

Algorithm	Oracle	Runtime	Always applicable?	Exact?	Sparse?	Practical?
`ELLIPSOID`	MIN	Theorem 6.4.1	Yes	Yes	Yes	No
`MWU`	AMIN	Theorem 6.4.7	Yes	No	Yes	Yes
`SINKHORN`	SMIN	Theorem 6.4.18	No	No	No	Yes

Table 6.1: These MOT algorithms have polynomial runtime except for a bottleneck "oracle". Each oracle is a simple variant of the dual feasibility oracle for MOT. The number of oracle computations is $\text{poly}(n, k)$ for `ELLIPSOID`, and $\text{poly}(n, k, C_{\max}/\varepsilon)$ for both `MWU` and `SINKHORN`. From a theoretical perspective, the most important aspect of an algorithm is whether it can solve MOT in polynomial time if and only if any algorithm can. We show that `ELLIPSOID` and `MWU` satisfy this (Theorem 6.1.1), but `SINKHORN` does not (Theorem 6.1.3). From a practical perspective, `SINKHORN` is the most scalable when applicable. Other important properties of an algorithm are whether it can compute exact and/or sparse solutions.[2]

6.1.1 Contribution 1: unified algorithmic framework for MOT

In order to understand what structural properties make MOT solvable in polynomial time, we first lay a more general groundwork. The purpose of this is to understand the following fundamental questions:

Q1 What are reasonable candidate algorithms for solving structured MOT problems in polynomial time?

Q2 What structure must an MOT problem have for these algorithms to have polynomial runtimes?

Q3 Is the structure required by a given algorithm more restrictive than the structure required by a different algorithm (or *any* algorithm)?

Q4 How to check if this structure occurs for a given MOT problem?

We detail our answers to these four questions below in §6.1.1.1 to §6.1.1.4, and then briefly discuss practical tradeoffs beyond polynomial-time solvability in §6.1.1.5; see Table 6.1 for a summary. We expect that this general groundwork will prove useful in future investigation of tractable MOT problems.

6.1.1.1 Answer to Q1: candidate $\text{poly}(n, k)$-time algorithms

We consider three algorithms for MOT whose exponential runtimes can be isolated into a single bottleneck—and thus can be implemented in polynomial time

[2]Code for implementing these algorithms and reproducing all numerical simulations in this chapter is provided at `https://github.com/eboix/mot`.

whenever that bottleneck can. These algorithms are the Ellipsoid algorithm `ELLIPSOID` [102], the Multiplicative Weights Update algorithm `MWU` [237], and the natural multidimensional analog of Sinkhorn's scaling algorithm `SINKHORN` [32, 178]. `SINKHORN` is specially tailored to MOT and is currently the predominant algorithm for it. To foreshadow our answer to Q3, the reason that we restrict to these candidate algorithms is: we show that `ELLIPSOID` and `MWU` can solve an MOT problem in polynomial time if and only if any algorithm can.

■ 6.1.1.2 Answer to Q2: structure necessary to run candidate algorithms

These three algorithms only access the cost tensor C through polynomially many calls of their respective bottlenecks. Thus the structure required to implement these candidate algorithms in polynomial time is equivalent to the structure required to implement their respective bottlenecks in polynomial time.

In §6.4, we show that the bottlenecks of these three algorithms are polynomial-time equivalent to natural analogs of the feasibility oracle for the dual LP to MOT. Namely, given weights $p_1, \dots, p_k \in \mathbb{R}^n$, compute

$$\min_{(j_1,\dots,j_k)\in\{1,\dots,n\}^k} C_{j_1,\dots,j_k} - \sum_{i=1}^{k} [p_i]_{j_i} \tag{6.1}$$

either exactly for `ELLIPSOID`, approximately for `MWU`, or with the "min" replaced by a "softmin" for `SINKHORN`. We call these three tasks the MIN, AMIN, and SMIN oracles, respectively. See Remark 6.3.4 for the interpretation of these oracles as variants of the dual feasibility oracle.

These three oracles take n^k time to implement in general. However, for a wide range of structured cost tensors C they can be implemented in $\mathrm{poly}(n, k)$ time, see §6.1.2 below. For such structured costs C, our oracle abstraction immediately implies that the MOT problem with cost C and any input marginals $\mu_1, \dots, \mu_k$ can be (approximately) solved in polynomial time by any of the three respective algorithms.

Our characterization of the algorithms' bottlenecks as variations of the dual feasibility oracle has two key benefits—which are the answers to Q3 and Q4, described below.

■ 6.1.1.3 Answer to Q3: characterizing what MOT problems each algorithm can solve

A key benefit of our characterization of the algorithms' bottlenecks as variations of the dual feasibility oracles is that it enables us to establish whether the structure required by a given MOT algorithm is more restrictive than the structure required by a different algorithm (or by *any* algorithm).

In particular, this enables us to answer the natural question: why restrict to just the three algorithms described above? Can other algorithms solve MOT in $\mathrm{poly}(n, k)$ time in situations when these algorithms cannot? Critically, the answer is no: restricting ourselves to the ELLIPSOID and MWU algorithms is at no loss of generality.

Theorem 6.1.1 (Informal statement of part of Theorems 6.4.1 and 6.4.7)**.** *For any family of costs $C \in (\mathbb{R}^n)^{\otimes k}$:*

- *ELLIPSOID computes an exact solution for MOT in* $\mathrm{poly}(n, k)$ *time if and only if any algorithm can.*
- *MWU computes an ε-approximate solution for MOT in* $\mathrm{poly}(n, k, C_{\max}/\varepsilon)$ *time if and only if any algorithm can.*

The statement for ELLIPSOID is implicit from classical results about LP [102] combined with arguments from [11], see the previous work section §6.1.3. The statement for MWU is new to this chapter.

The oracle abstraction helps us show Theorem 6.1.1 because it reduces this question of what structure is needed for the algorithms to solve MOT in polynomial time, to the question of what structure is needed to solve their respective bottlenecks in polynomial time. Thus Theorem 6.1.1 is a consequence of the following result. (The "if" part of this result is a contribution of this chapter; the "only if" part was shown in [12].)

Theorem 6.1.2 (Informal statement of part of Theorems 6.4.1 and 6.4.7)**.** *For any family of costs $C \in (\mathbb{R}^n)^{\otimes k}$:*

- *MOT can be exactly solved in* $\mathrm{poly}(n, k)$ *time if and only if MIN can.*
- *MOT can be ε-approximately solved in* $\mathrm{poly}(n, k, C_{\max}/\varepsilon)$ *time if and only if AMIN can.*

Interestingly, a further consequence of our unified algorithm-to-oracle abstraction is that it enables us to show that SINKHORN—which is currently the most popular algorithm for MOT by far—requires strictly more structure to solve an MOT problem than other algorithms require. This is in sharp contrast to the complete generality of the other two algorithms shown in Theorem 6.1.1.

Theorem 6.1.3 (Informal statement of Theorem 6.4.19)**.** *Under standard complexity-theoretic assumptions, there exists a family of MOT problems that can be solved exactly in* $\mathrm{poly}(n, k)$ *time using ELLIPSOID, however it is impossible to implement a single iteration of SINKHORN (even approximately) in* $\mathrm{poly}(n, k)$ *time.*

The reason that our unified algorithm-to-oracle abstraction helps us show Theorem 6.1.3 is that it puts SINKHORN on equal footing with the other two classical algorithms in terms of their reliance on variants of the dual feasibility oracle. This reduces proving Theorem 6.1.3 to showing the following separation between the SMIN oracle and the other two oracles.

Theorem 6.1.4 (Informal statement of Lemma 6.3.7)**.** *Under standard complexity-theoretic assumptions, there exists a family of cost tensors $C \in (\mathbb{R}^n)^{\otimes k}$ such that there are* $\mathrm{poly}(n, k)$*-time algorithms for MIN and AMIN, however it is impossible to solve SMIN (even approximately) in* $\mathrm{poly}(n, k)$ *time.*

■ 6.1.1.4 Answer to Q4: ease-of-use for checking if MOT is solvable in polynomial time

The second key benefit of this oracle abstraction is that it is helpful for showing that a given MOT problem (whose cost C is input implicitly through some concise representation) is solvable in polynomial time as it without loss of generality reduces MOT to solving any of the three corresponding oracles in polynomial time. The upshot is that these oracles are more directly amenable to standard algorithmic techniques since they are phrased as more conventional combinatorial-optimization problems. In the second part of the chapter, we illustrate this ease-of-use via applications to three general classes of structured MOT problems; for an overview see §6.1.2.

■ 6.1.1.5 Practical algorithmic tradeoffs beyond polynomial-time solvability

From a theoretical perspective, the most important aspect of an algorithm is whether it can solve MOT in polynomial time if and only if any algorithm can. As we have discussed, this is true for ELLIPSOID and MWU (Theorem 6.1.1) but not for SINKHORN (Theorem 6.1.3). Nevertheless, for a wide range of MOT cost structures, all three oracles can be implemented in polynomial time, which means that all three algorithms ELLIPSOID, MWU, and SINKHORN can be implemented in polynomial time. Which algorithm is best in practice depends on the relative importance of the following considerations for the particular application.

- *Error.* ELLIPSOID computes exact solutions, whereas MWU and SINKHORN only compute low-precision solutions due to $\mathrm{poly}(1/\varepsilon)$ runtime dependence.
- *Solution sparsity.* ELLIPSOID and MWU output solutions with polynomially many non-zero entries (roughly nk), whereas SINKHORN outputs fully dense solutions with n^k non-zero entries (through a polynomial-size implicit representation, see §6.4.3). Solution sparsity enables interpretability, visualization, and efficient downstream computation—benefits which are helpful in diverse

applications, for example ranging from computer graphics [40, 179, 206] to facility location problems [22] to machine learning [11, 71] to ecological inference [158] to fluid dynamics (see §6.5.3), and more. Furthermore, in §6.7.4, we show that sparse solutions for MOT (a.k.a. linear optimization over the transportation polytope) enable efficiently solving certain non-linear optimization problems over the transportation polytope.

- *Practical runtime.* Although all three algorithms enjoy polynomial runtime guarantees, the polynomials are smaller for some algorithms than for others. In particular, `SINKHORN` has remarkably good scalability in practice as long the error ε is not too small and its bottleneck oracle SMIN is practically implementable. By Theorems 6.1.1 and 6.1.3, `MWU` can solve strictly more MOT problems in polynomial time than `SINKHORN`; however, it is less scalable in practice when both `MWU` and `SINKHORN` can be implemented. `ELLIPSOID` is not practical and is used solely as a proof of concept that problems are tractable to solve exactly; in practice, we use Column Generation (see, e.g., [37, §6.1]) rather than `ELLIPSOID` as it has better empirical performance, yet still has the same bottleneck oracle MIN, see §6.4.1.3. Column Generation is not as practically scalable as `SINKHORN` in n and k but has the benefit of computing exact, sparse solutions.

To summarize: which algorithm is best in practice depends on the application. For example, Column Generation produces the qualitatively best solutions for the fluid dynamics application in §6.5.3, `SINKHORN` is the most scalable for the risk estimation application in §6.7.3, and `MWU` is the most scalable for the network reliability application in §6.6.3 (for that application there is no known implementation of `SINKHORN` that is practically efficient).

■ 6.1.2 Contribution 2: applications to general classes of structured MOT problems

In the second part of the chapter, we illustrate the algorithmic framework developed in the first part of the chapter by applying it to three general classes of MOT cost structures:

1. Graphical structure (in §6.5).
2. Set-optimization structure (in §6.6).
3. Low-rank plus sparse structure (in §6.7).

Specifically, if the cost C is structured in any of these three ways, then MOT can be (approximately) solved in $\text{poly}(n, k)$ time for any input marginals $\mu_1, \ldots, \mu_k$.

Previously, it was known how to solve MOT problems with structure (1) using `SINKHORN` [110, 216], but this only computes solutions that are dense (with n^k non-zero entries) and low-precision (due to $\text{poly}(1/\varepsilon)$ runtime dependence). We therefore provide the first solutions that are sparse and exact for structure (1). For structures (2) and (3), we provide the first polynomial-time algorithms, even for approximate computation. These three structures are incomparable: it is in general not possible to model a problem falling under any of the three structures in a non-trivial way using any of the others, for details see Remarks 6.6.7 and 6.7.3. This means that the new structures (2) and (3) enable capturing a wide range of new applications.

Below, we detail these structures individually in §6.1.2.1, §6.1.2.2, and §6.1.2.3. See Table 6.2 for a summary.

Structure	Definition	Complexity measure	Polynomial-time algorithm?	
			Approximate	Exact
Graphical (§6.5)	$C_{\vec{j}} = \sum_{S \in \mathcal{S}} f_S(\vec{j}_S)$	treewidth	[110, 216]	Corollary 6.5.6
Set-optimization (§6.6)	$C_{\vec{j}} = \mathbb{1}[\vec{j} \notin S]$	optimization over S	Corollary 6.6.9	Corollary 6.6.9
Low-rank + sparse (§6.7)	$C = R + S$	rank of R, sparsity of S	Corollary 6.7.5	Unknown

Table 6.2: In the second part of the chapter, we illustrate the ease-of-use of our algorithmic framework by applying it to three general classes of MOT cost structures. These structures encompass many—if not most—current applications of MOT.

■ 6.1.2.1 Graphical structure

In §6.5, we apply our algorithmic framework to MOT problems with graphical structure, a broad class of MOT problems that have been previously studied [108, 110, 216]. Briefly, an MOT problem has graphical structure if its cost tensor C decomposes as

$$C_{j_1,\ldots,j_k} = \sum_{S \in \mathcal{S}} f_S(\vec{j}_S),$$

where $f_S(\vec{j}_S)$ are arbitrary "local interactions" that depend only on tuples $\vec{j}_S := \{j_i\}_{i \in S}$ of the k variables.

In order to derive efficient algorithms, it is necessary to restrict how local the interactions are because otherwise MOT is NP-hard (even if all interaction sets $S \in \mathcal{S}$ have size 2) [12]. We measure the locality of the interactions via the standard complexity measure of the "treewidth" of the associated graphical model.

See §6.5.1 for formal definitions. While the runtimes of our algorithms (and all previous algorithms) depend exponentially on the treewidth, we emphasize that the treewidth is a very small constant (either 1 or 2) in all current applications of MOT falling under this framework; see the related work section.

We show that for MOT cost tensors that have graphical structure of constant treewidth, all three oracles can be implemented in $\text{poly}(n, k)$ time. We accomplish this by leveraging the known connection between graphically structured MOT and graphical models shown in [110]. In particular, the MIN, AMIN, and SMIN oracles are respectively equivalent to the mode, approximate mode, and log-partition function of an associated graphical model. Thus we can implement all oracles in $\text{poly}(n, k)$ time by simply applying classical algorithms from the graphical models community [134, 227].

Theorem 6.1.5 (Informal statement of Theorem 6.5.5)**.** *Let $C \in (\mathbb{R}^n)^{\otimes k}$ have graphical structure of constant treewidth. Then the MIN, AMIN, and SMIN oracles can be computed in* $\text{poly}(n, k)$ *time.*

It is an immediate corollary of Theorem 6.1.5 and our algorithms-to-oracles reduction described in §6.1.1 that one can implement `ELLIPSOID`, `MWU`, and `SINKHORN` in polynomial time. Below, we record the theoretical guarantee of `ELLIPSOID` since it is the best of the three algorithms as it computes exact, sparse solutions.

Theorem 6.1.6 (Informal statement of Corollary 6.5.6)**.** *Let $C \in (\mathbb{R}^n)^{\otimes k}$ have graphical structure of constant treewidth. Then an exact, sparse solution for MOT can be computed in* $\text{poly}(n, k)$ *time.*

Previously, it was known how to solve such MOT problems [110, 216] using `SINKHORN`, but this only computes a solution that is fully dense (with n^k non-zero entries) and low-precision (due to $\text{poly}(1/\varepsilon)$ runtime dependence). Details in the Related Work section. Our result improves over this state-of-the-art algorithm by producing solutions that are *exact* and *sparse* in $\text{poly}(n, k)$ time.

In §6.5.3, we demonstrate the benefit of Theorem 6.1.6 on the application of computing generalized Euler flows, which was historically the motivation of MOT and has received significant attention, e.g., [32, 35, 47, 48, 49, 50]. While there is a specially-tailored version of the `SINKHORN` algorithm for this problem that runs in polynomial time [32, 35], it produces solutions that are approximate and fully dense. Our algorithm produces exact, sparse solutions which lead to sharp visualizations rather than blurry ones (see Figure 6.4).

■ 6.1.2.2 Set-optimization structure

In §6.6, we apply our algorithmic framework to MOT problems whose cost tensors C take value 0 or 1 in each entry. That, is costs C of the form

$$C_{j_1,\ldots,j_k} = \mathbb{1}[(j_1,\ldots,j_k) \notin S],$$

for some subset $S \subseteq [n]^k$. Such MOT problems arise naturally in applications where one seeks to minimize the probability that some event S occurs, given marginal probabilities on each variable j_i, see Example 6.6.1.

In order to derive efficient algorithms, it is necessary to restrict the (otherwise arbitrary) set S. We parametrize the complexity of such MOT problems via the complexity of finding the minimum-weight object in S. This opens the door to combinatorial applications of MOT because finding the minimum-weight object in S is well-known to be polynomial-time solvable for many "combinatorially-structured" sets S of interest—e.g., the set S of cuts in a graph, or the set S of independent sets in a matroid.

We show that for MOT cost tensors with this structure, all three oracles can be implemented efficiently.

Theorem 6.1.7 (Informal statement of Theorem 6.6.8)**.** *Let $C \in (\mathbb{R}^n)^{\otimes k}$ have set-optimization structure. Then the MIN, AMIN, and SMIN oracles can be computed in* $\mathrm{poly}(n,k)$ *time.*

It is an immediate corollary of Theorem 6.1.7 and our algorithms-to-oracles reduction described in §6.1.1 that one can implement `ELLIPSOID`, `MWU`, and `SINKHORN` in polynomial time. Below, we record the theoretical guarantee for `ELLIPSOID` since it is the best of these three algorithms as it computes exact, sparse solutions.

Theorem 6.1.8 (Informal statement of Corollary 6.6.9)**.** *Let $C \in (\mathbb{R}^n)^{\otimes k}$ have set-optimization structure. Then an exact, sparse solution for MOT can be computed in* $\mathrm{poly}(n,k)$ *time.*

This is the first polynomial-time algorithm for this class of MOT problems. We note that a more restrictive class of MOT problems was studied in [241] under the additional restriction that S is upwards-closed.

In §6.6.3, we show how this general class of set-optimization structure captures, for example, the classical application of computing the extremal reliability of a network with stochastic edge failures. Network reliability is a fundamental topic in network science and engineering [28, 29, 95] which is often studied in an average-case setting where each edge fails independently with some given probability [128, 156, 181, 221]. The application investigated here is a robust

notion of network reliability in which edge failures may be maximally correlated (e.g., by an adversary) or minimally correlated (e.g., by a network maintainer) subject to a marginal constraint on each edge's failure probability, a setting that dates back to the 1980s [229, 241]. We show how to express both the minimally and maximally correlated network reliability problems as MOT problems with set-optimization structure, recovering as a special case of our general framework the known polynomial-time algorithms in [229, 241] as well as more practical polynomial-time algorithms that scale to input sizes that are an order-of-magnitude larger.

■ 6.1.2.3 Low-rank and sparse structure

In §6.7, we apply our algorithmic framework to MOT problems whose cost tensors C decompose as

$$C = R + S,$$

where R is a constant-rank tensor, and S is a polynomially-sparse tensor. We assume that R is represented in factored form, and that S is represented through its non-zero entries, which overall yields a $\text{poly}(n, k)$-size representation of C.

We show that for MOT cost tensors with low-rank plus sparse structure, the AMIN and SMIN oracles can be implemented in polynomial time.[3] This may be of independent interest because, by taking all oracle inputs $p_i = 0$ in (6.1), this generalizes the previously open problem of approximately computing the smallest entry of a constant-rank tensor with n^k entries in $\text{poly}(n, k)$ time.

Theorem 6.1.9 (Informal statement of Theorem 6.7.4)**.** *Let $C \in (\mathbb{R}^n)^{\otimes k}$ have low-rank plus sparse structure. Then the AMIN and SMIN oracles can be computed in $\text{poly}(n, k, C_{\max}/\varepsilon)$ time.*

It is an immediate corollary of Theorem 6.1.9 and our algorithms-to-oracles reduction described in §6.1.1 that one can implement `MWU` and `SINKHORN` in polynomial time. Of these two algorithms, `MWU` computes sparse solutions, yielding the following theorem.

Theorem 6.1.10 (Informal statement of Corollary 6.7.5)**.** *Let $C \in (\mathbb{R}^n)^{\otimes k}$ have low-rank plus sparse structure. Then a sparse, ε-approximate solution for MOT can be computed in $\text{poly}(n, k, C_{\max}/\varepsilon)$ time.*

[3] It is an interesting open question if the MIN oracle can similarly be implemented in $\text{poly}(n, k)$ time. This would enable implementing `ELLIPSOID` in $\text{poly}(n, k)$ time by our algorithms-to-oracles reduction, and thus would enable computing exact solutions for this class of MOT problems (cf., Theorem 6.1.10).

This is the first polynomial-time result for this class of MOT problems. We note that the runtime of our MOT algorithm depends exponentially on the rank r of R, hence why we take r to be constant. Nevertheless, such a restriction on the rank is unavoidable since unless P = NP, there does not exist an algorithm with runtime that is jointly polynomial in n, k, and the rank r [12].

We demonstrate this polynomial-time algorithm concretely on two applications. First, in §6.7.3 we consider the risk estimation problem of computing an investor's expected profit in the worst-case over all future prices that are consistent with given marginal distributions. We show that this is equivalent to an MOT problem with a low-rank tensor and thereby provide the first efficient algorithm for it.

Second, in §6.7.4, we consider the fundamental problem of projecting a joint distribution Q onto the transportation polytope. We provide the first polynomial-time algorithm for solving this when Q decomposes into a constant-rank and sparse component, which models mixtures of product distributions with polynomially many corruptions. This application illustrates the versatility of our algorithmic results beyond polynomial-time solvability of MOT, since this projection problem is a *quadratic* optimization over the transportation polytope rather than linear optimization (a.k.a. MOT). In order to achieve this, we develop a simple quadratic-to-linear reduction tailored to this problem that crucially exploits the sparsity of the MOT solutions enabled by the MWU algorithm.

6.1.3 Related work

MOT algorithms fall into two categories. One category consists of general-purpose algorithms that do not depend on the specific MOT cost. For example, this includes running an LP solver out-of-the-box, or running the Sinkhorn algorithm where in each iteration one sums over all n^k entries of the cost tensor to implement the marginalization bottleneck [89, 144, 220]. These approaches are robust in the sense that they do not need to be changed based on the specific MOT problem. However, they are impractical beyond tiny input sizes (e.g., $n = k = 15$) because their runtimes scale as $n^{\Omega(k)}$.

The second category consists of algorithms that are much more scalable but require extra structure of the MOT problem. Specifically, these are algorithms that somehow exploit the structure of the relevant cost tensor C in order to (approximately) solve an MOT problem in $\mathrm{poly}(n, k)$ time [2, 11, 32, 34, 35, 55, 61, 86, 107, 108, 110, 111, 152, 153, 159, 160, 162, 170, 201, 216, 229, 241]. Such a $\mathrm{poly}(n, k)$ runtime is far more tractable—but it is not well understood for which MOT problems such a runtime is possible. The purpose of this chapter is to clarify this question.

To contextualize our answer to this question with the rapidly growing literature

requires further splitting this second category of algorithms.

Sinkhorn algorithm. Currently, the predominant approach in the second category is to solve an entropically regularized version of MOT with the Sinkhorn algorithm, a.k.a. Iterative Proportional Fitting or Iterative Bregman Projections or RAS algorithm or Iterative Scaling algorithm, see e.g., [33, 34, 35, 108, 110, 162, 216]. Recent work has shown that a polynomial number of iterations of this algorithm suffices [89, 144, 220]. However, the bottleneck is that each iteration requires n^k operations in general because it requires marginalizing a tensor with n^k entries. The critical question is therefore: what structure of an MOT problem enables implementing this marginalization bottleneck in polynomial time.

This chapter makes two contributions to this question. First, we identify new broad classes of MOT problems for which this bottleneck can be implemented in polynomial time, and thus `SINKHORN` can be implemented in polynomial time (see §6.1.2). Second, we propose other algorithms that require strictly less structure than `SINKHORN` does in order to solve an MOT problem in polynomial time (Theorem 6.4.19).

Ellipsoid algorithm. The Ellipsoid algorithm is among the most classical algorithms for implicit LP [101, 102, 132], however it has taken a back seat to the `SINKHORN` algorithm in the vast majority of the MOT literature.

In §6.4.1, we make explicit the fact that the variant of `ELLIPSOID` from [11] can solve MOT exactly in $\text{poly}(n, k)$ time if and only if any algorithm can (Theorem 6.4.1). This is implicit from combining several known results [11, 12, 102]. In the process of making this result explicit, we exploit the special structure of the MOT LP to significantly simplify the reduction from the dual violation oracle to the dual feasibility oracle. The previously known reduction is highly impractical as it requires an indirect "back-and-forth" use of the Ellipsoid algorithm [102, page 107]. In contrast, our reduction is direct and simple; this is critical for implementing our practical alternative to `ELLIPSOID`, namely `COLGEN`, with the dual feasibility oracle.

Multiplicative Weights Update algorithm. This algorithm, first introduced by [237], has been studied in the context of Optimal Transport when $k = 2$ [39, 182], in which case implicit LP is not necessary for a polynomial runtime. `MWU` lends itself to implicit LP [237], but is notably absent from the MOT literature.

In §6.4.2, we show that `MWU` can be applied to MOT in polynomial time if and only if the approximate dual feasibility oracle can be solved in polynomial time. To do this, we show that in the special case of MOT, the well-known "softmax-derivative" bottleneck of `MWU` is polynomial-time equivalent to the approximate dual feasibility oracle. Since it is known that the approximate dual feasibility

oracle is polynomial-time reducible to approximate MOT (Theorem 5.3.3), we therefore establish that MWU can solve MOT approximately in polynomial time if and only if any algorithm can (Theorem 6.4.7).

■ 6.1.3.1 Graphically structured MOT problems with constant treewidth

We isolate here graphically structured costs with constant treewidth because this framework encompasses all MOT problems that were previously known to be tractable in polynomial time [110, 216], with the exceptions of the fixed-dimensional Wasserstein barycenter problem and MOT problems related to combinatorial optimization—both of which are described below in §6.1.3.2. This family of costs with treewidth 1 (a.k.a. "tree-structured costs" [108]) includes applications in economics such as variational mean-field games [34], interpolating histograms on trees [7], matching for teams [55, 162]; as well as encompasses applications in filtering and estimation for collective dynamics such as target tracking [86, 107, 108, 110, 201] and Wasserstein barycenters in the case of fixed support [32, 55, 86, 162]. With treewidth 2, this family of costs also includes dynamic multi-commodity flow problems [109], as well as the application of computing generalized Euler flows in fluid dynamics [32, 35, 162], which was historically the original motivation of MOT [47, 48, 49, 50].

Previous polynomial-time algorithms for graphically structured MOT compute approximate, dense solutions. Implementing SINKHORN for graphically structured MOT problems by using belief propagation to efficiently implement the marginalization bottleneck was first proposed twenty years ago in [216]. There have been recent advancements in understanding connections of this algorithm to the Schrödinger bridge problem in the case of trees [108], as well as developing more practically efficient single-loop variations [110].

All of these works prove theoretical runtime guarantees only in the case of tree structure (i.e., treewidth 1). However, this graphical model perspective for efficiently implementing SINKHORN readily extends to any constant treewidth: simply implement the marginalization bottleneck using junction trees. This, combined with the iteration complexity of SINKHORN which is known to be polynomial [89, 144, 220], immediately yields an overall polynomial runtime. This is why we cite [110, 216] throughout this chapter regarding the fact that SINKHORN can be implemented in polynomial time for graphical structure with any constant treewidth.

While the use of SINKHORN for graphically structured MOT is mathematically elegant and can be impressively scalable in practice, it has two drawbacks. The first drawback of this algorithm is that it computes (implicit representations of) solutions that are fully dense with n^k non-zero entries. Indeed, it is well-known

that SINKHORN finds the unique optimal solution to the entropically regularized MOT problem $\min_{P \in \mathcal{M}(\mu_1,\ldots,\mu_k)} \langle P, C \rangle - \eta^{-1} H(P)$, and that this solution is fully dense [178]. For example, in the simple case of cost $C = 0$, uniform marginals μ_i, and any strictly positive regularization parameter $\eta > 0$, this solution P has value $1/n^k$ in each entry.

The second drawback of this algorithm is that it only computes solutions that are low-precision due to $\text{poly}(1/\varepsilon)$ runtime dependence on the accuracy ε. This is because the number of SINKHORN iterations is known to scale polynomially in the entropic regularization parameter η even in the matrix case $k = 2$ [145, §1.2], and it is known that $\eta = \Omega(\varepsilon^{-1} k \log n)$ is necessary for the converged solution of SINKHORN to be an ε-approximate solution to the (unregularized) original MOT problem [144].

Improved algorithms for graphically structured MOT problems. The contribution of this chapter to the study of graphically structured MOT problems is that we give the first $\text{poly}(n, k)$ time algorithms that can compute solutions which are exact and sparse (Corollary 6.5.6). Our framework also directly recovers all known results about SINKHORN for graphically structured MOT problems—namely that it can be implemented in polynomial time for trees [108, 216] and for constant treewidth [110, 216].

■ 6.1.3.2 Tractable MOT problems beyond graphically structured costs

The two new classes of MOT problems that we identify in this chapter—namely, set-optimization structure and low-rank plus sparse structure—are incomparable to each other as well as to graphical structure. Details in Remarks 6.6.7 and 6.7.3. This lets us handle a wide range of new MOT problems that could not be handled before.

There are two other classes of MOT problems studied in the literature which do not fall under the three structures studied in this chapter. We elaborate on both below.

Remark 6.1.11 (Low-dimensional Wasserstein barycenter)**.** *This MOT problem has cost* $C_{j_1,\ldots,j_k} = \sum_{i,i'=1}^{k} \|x_{i,j_i} - x_{i',j_{i'}}\|^2$ *where* $x_{i,j} \in \mathbb{R}^d$ *denotes the* j*-th atom in the distribution* μ_i*. Clearly this cost is not a graphically structured cost of constant treewidth—indeed, representing it through the lens of graphical structure requires the complete graph of interactions, which means a maximal treewidth of* $k - 1$.[4] *Nevertheless, in Chapter 7 we show that this MOT problem can be solved*

[4]We remark that the related but different problem of *fixed-support* Wasserstein barycenters has graphical structure with treewidth 1 [32, 55, 86, 162]. However, the fixed-support Wasserstein barycenter problem is different from the Wasserstein barycenter problem: it only approximates

in $\mathrm{poly}(n, k)$ *time by exploiting the low-dimensional geometric structure of the points* $\{x_{i,j}\}$ *that implicitly define the cost.*

Remark 6.1.12 (Random combinatorial optimization)**.** *MOT problems also appear in the random combinatorial optimization literature since the 1970s, see e.g., [111, 152, 159, 229, 241], although under a different name and in a different community. These papers consider MOT problems with costs of the form* $C(x) = \min_{v \in V} \langle x, v \rangle$ *for polytopes* $V \subseteq \{0,1\}^k$ *given through a list of their extreme points. Applications include PERT (Program Evaluation and Review Technique), extremal network reliability, and scheduling. Recently, applications to Distributionally Robust Optimization were investigated in [61, 153, 160] which considered general polytopes* $V \subset \mathbb{R}^k$*, as well as in [170] which considered MOT costs of the related form* $C(x) = \mathbb{1}[\min_{v \in V} \langle x, v \rangle \geqslant t]$*, and in [2] which considers other combinatorial costs* C *such as sub/supermodular functions. These papers show that these random combinatorial optimization problems are in general intractable, and give sufficient conditions on when they can be solved in polynomial time. In general, these families of MOT problems are different from the three structures studied in this chapter, although some MOT applications fall under multiple umbrellas (e.g., extremal network reliability). It is an interesting question to understand to what extent these structures can be reconciled (as well as the algorithms, which sometimes use extended formulations in these papers).*

■ 6.1.4 Organization

In §6.2 we recall preliminaries about MOT and establish notation. The first part of the chapter then establishes our unified algorithmic framework for MOT. Specifically, in §6.3 we define and compare three variants of the dual feasibility oracle; and in §6.4 we characterize the structure that MOT algorithms require for polynomial-time implementation in terms of these three oracles. For an overview of these results, see §6.1.1. The second part of the chapter applies this algorithmic framework to three general classes of MOT cost structures: graphical structure (§6.5), set-optimization structure (§6.6), and low-rank plus sparse structure (§6.7). For an overview of these results, see §6.1.2. These three application sections are independent of each other and can be read separately. We conclude in §6.8.

---

the latter to ε accuracy if the fixed support is restricted to an $O(\varepsilon)$-net which requires $n = 1/\varepsilon^{\Omega(d)}$ discretization size for the barycenter's support, and thus (i) even in constant dimension, does not lead to high-precision algorithms due to $\mathrm{poly}(1/\varepsilon)$ runtime; and (ii) scales exponentially in the dimension d. See Chapter 7 for further details about the complexity of Wasserstein barycenters.

6.2 Preliminaries

General notation. The set $\{1, \dots, n\}$ is denoted by $[n]$. For shorthand, we write $\mathrm{poly}(t_1, \dots, t_m)$ to denote a function that grows at most polynomially fast in those parameters. Throughout, we assume for simplicity of exposition that all entries of the input C and $\mu_1, \dots, \mu_k$ have bit complexity at most $\mathrm{poly}(n, k)$, and same with the components defining C in structured settings. As such, throughout runtimes refer to the number of arithmetic operations. The set $\mathbb{R} \cup \{-\infty\}$ is denoted by $\bar{\mathbb{R}}$, and note that the value $-\infty$ can be represented efficiently by adding a single flag bit. We use the standard $O(\cdot)$ and $\Omega(\cdot)$ notation, and use $\tilde{O}(\cdot)$ and $\tilde{\Omega}(\cdot)$ to denote that polylogarithmic factors may be omitted.

Tensor notation. The k-fold tensor product space $\mathbb{R}^n \otimes \cdots \otimes \mathbb{R}^n$ is denoted by $(\mathbb{R}^n)^{\otimes k}$, and similarly for $(\mathbb{R}^n_{\geqslant 0})^{\otimes k}$. Let $P \in (\mathbb{R}^n)^{\otimes k}$. Its i-th marginal, $i \in [k]$, is denoted by $m_i(P) \in \mathbb{R}^n$ and has entries $[m_i(P)]_j := \sum_{j_1,\dots,j_{i-1},j_{i+1},\dots,j_k} P_{j_1,\dots,j_{i-1},j,j_{i+1},\dots,j_k}$. For shorthand, we often denote an index $(j_1, \dots, j_k)$ by $\vec{j}$. The sum of P's entries is denoted by $m(P) = \sum_{\vec{j}} P_{\vec{j}}$. The maximum absolute value of P's entries is denoted by $\|P\|_{\max} := \max_{\vec{j}} |P_{\vec{j}}|$, or simply $P_{\max}$ for short. For $\vec{j} \in [n]^k$, we write $\delta_{\vec{j}}$ to denote the tensor with value 1 at entry $\vec{j}$, and 0 elswewhere. The operations $\odot$ and $\otimes$ respectively denote the entrywise product and the Kronecker product. The notation $\otimes_{i=1}^k d_i$ is shorthand for $d_1 \otimes \cdots \otimes d_k$. A non-standard notation we use throughout is that $f[P]$ denotes a function $f : \mathbb{R} \to \mathbb{R}$ (typically exp, log, or a polynomial) applied *entrywise* to a tensor P.

6.2.1 Multimarginal Optimal Transport

The transportation polytope between measures $\mu_1, \dots, \mu_k \in \Delta_n$ is

$$\mathcal{M}(\mu_1, \dots, \mu_k) := \left\{ P \in (\mathbb{R}^n_{\geqslant 0})^{\otimes k} \ : \ m_i(P) = \mu_i, \ \forall i \in [k] \right\}. \tag{6.2}$$

For a fixed cost $C \in (\mathbb{R}^n)^{\otimes k}$, the MOT_C problem is to solve the following linear program, given input measures $\mu = (\mu_1, \dots, \mu_k) \in (\Delta_n)^k$:

$$\min_{P \in \mathcal{M}(\mu_1,\dots,\mu_k)} \langle P, C \rangle. \tag{MOT}$$

In the $k = 2$ matrix case, (MOT) is the Kantorovich formulation of OT [224]. Its dual LP is

$$\max_{p_1,\dots,p_k \in \mathbb{R}^n} \sum_{i=1}^k \langle p_i, \mu_i \rangle \quad \text{subject to} \quad C_{j_1,\dots,j_k} - \sum_{i=1}^k [p_i]_{j_i} \geqslant 0, \ \forall (j_1, \dots, j_k) \in [n]^k. \tag{MOT-D}$$

A basic, folklore fact about MOT is that it always has a sparse optimal solution (e.g., [22, Lemma 3]). This follows from elementary facts about standard-form LP.

Lemma 6.2.1 (Sparse solutions for MOT)**.** *For any cost $C \in (\mathbb{R}^n)^{\otimes k}$ and any marginals $\mu_1, \dots, \mu_k \in \Delta_n$, there exists an optimal solution P to $\mathsf{MOT}_C(\mu)$ that has at most $nk - k + 1$ non-zero entries.*

Definition 6.2.2 (ε-approximate MOT solution)**.** *P is an ε-approximate solution to $\mathsf{MOT}_C(\mu)$ if P is feasible (i.e., $P \in \mathcal{M}(\mu_1, \dots, \mu_k)$) and $\langle C, P\rangle$ is at most ε more than the optimal value.*

■ 6.2.2 Regularization

We introduce two standard regularization operators. First is the Shannon entropy $H(P) := -\sum_{\vec{j}} P_{\vec{j}} \log P_{\vec{j}}$ of a tensor $P \in (\mathbb{R}^n_{\geqslant 0})^{\otimes k}$ with entries summing to $m(P) = 1$. We adopt the standard notational convention that $0 \log 0 = 0$. Second is the softmin operator, which is defined for parameter $\eta > 0$ as

$$\operatorname{smin}_{\eta, i \in [m]} a_i := -\frac{1}{\eta} \log \left(\sum_{i=1}^{m} e^{-\eta a_i} \right). \tag{6.3}$$

This softmin operator naturally extends to $a_i \in \mathbb{R} \cup \{\infty\}$ by adopting the standard notational conventions that $e^{-\infty} = 0$ and $\log 0 = -\infty$.

We make use of the following folklore fact, which bounds the error between the min and smin operators based on the regularization and the number of points. For completeness, we provide a short proof.

Lemma 6.2.3 (Softmin approximation bound)**.** *For any $a_1, \dots, a_m \in \mathbb{R} \cup \{\infty\}$ and $\eta > 0$,*

$$\min_{i \in [m]} a_i \geqslant \operatorname{smin}_{\eta, i \in [m]} a_i \geqslant \min_{i \in [m]} a_i - \frac{\log m}{\eta}.$$

The entropically regularized MOT problem (RMOT for short) is the convex optimization problem

$$\min_{P \in \mathcal{M}(\mu_1, \dots, \mu_k)} \langle P, C\rangle - \eta^{-1} H(P). \tag{RMOT}$$

This is the natural multidimensional analog of entropically regularized OT, which has a rich literature in statistics [142] and transportation theory [232], and has

recently attracted significant interest in machine learning [73, 178]. The convex dual of (RMOT) is the convex optimization problem

$$\max_{p_1,\dots,p_k\in\mathbb{R}^n} \sum_{i=1}^{k} \langle p_i, \mu_i\rangle + \operatorname{smin}_{\eta} \left(C_{\vec{j}} - \sum_{i=1}^{k} [p_i]_{j_i} \right). \tag{RMOT-D}$$

In contrast to MOT, there is no analog of Lemma 6.2.1 for RMOT: the unique optimal solution to RMOT is dense. Further, this solution may not even be "approximately" sparse. For example, when $C = 0$, all μ_i are uniform, and $\eta > 0$ is any positive number, the solution is fully dense with all entries equal to $1/n^k$.

We define P to be an ε-approximate RMOT solution in the analogous way as in Definition 6.2.2. A basic, folklore fact about RMOT is that if the regularization η is sufficiently large, then RMOT and MOT are equivalent in terms of approximate solutions.

Lemma 6.2.4 (MOT and RMOT are close for large regularization η)**.** *Let $P \in \mathcal{M}(\mu_1, \dots, \mu_k)$, $\varepsilon > 0$, and $\eta \geqslant \varepsilon^{-1} k \log n$. If P is an ε-approximate solution to* (RMOT)*, then P is also a (2ε)-approximate solution to* (MOT)*; and vice versa.*

Proof. Since a discrete distribution supported on n^k atoms has entropy at most $k \log n$ [72], the objectives of (MOT) and (RMOT) differ pointwise by at most $\eta^{-1} k \log n \leqslant \varepsilon$. Since (MOT) and (RMOT) also have the same feasible sets, their optimal values therefore differ by at most ε. □

■ 6.3 Oracles

Here we define the three oracle variants described in the introduction and discuss their relations. In the below definitions, let $C \in (\mathbb{R}^n)^{\otimes k}$ be a cost tensor.

Definition 6.3.1 (MIN oracle)**.** *For weights $p = (p_1, \dots, p_k) \in \mathbb{R}^{n\times k}$, $\mathit{MIN}_C(p)$ returns*

$$\min_{\vec{j}\in[n]^k} C_{\vec{j}} - \sum_{i=1}^{k} [p_i]_{j_i}.$$

Definition 6.3.2 (AMIN oracle)**.** *For weights $p = (p_1, \dots, p_k) \in \mathbb{R}^{n\times k}$ and accuracy $\varepsilon > 0$, $\mathit{AMIN}_C(p, \varepsilon)$ returns $\mathit{MIN}_C(p)$ up to additive error ε.*

Definition 6.3.3 (SMIN oracle)**.** *For weights $p = (p_1, \dots, p_k) \in \bar{\mathbb{R}}^{n\times k}$ and regularization parameter $\eta > 0$, $\mathit{SMIN}_C(p, \eta)$ returns*

$$\operatorname{smin}_{\eta} \, C_{\vec{j}} - \sum_{i=1}^{k} [p_i]_{j_i}.$$

An algorithm is said to "solve" or "implement" MIN_C if given input p, it outputs $\mathsf{MIN}_C(p)$. Similarly for AMIN_C and SMIN_C. Note that the weights p that are input to SMIN have values inside $\bar{\mathbb{R}} = \mathbb{R} \cup \{-\infty\}$; this simplifies the notation in the treatment of the `SINKHORN` algorithm below and does not increase the bit-complexity by more than 1 bit by adding a flag for the value $-\infty$.

Remark 6.3.4 (Interpretation as variants of the dual feasibility oracle)**.** *These three oracles can be viewed as variants of the feasibility oracle for* (MOT-D)*. For $\mathsf{MIN}_C(p)$, this relationship is exact: $p \in \mathbb{R}^{n\times k}$ is feasible for* (MOT-D) *if and only if $\mathsf{MIN}_C(p)$ is non-negative. For AMIN_C and SMIN_C, this relationship is approximate, with the approximation depending on how small ε is and how large η is, respectively.*

Since these oracles form the respective bottlenecks of all algorithms from the MOT and implicit linear programming literatures (see the overview in the introduction §6.1.1), an important question is: if one oracle can be implemented in $\mathrm{poly}(n, k)$ time, does this imply that the other can be too?

Two reductions are straightforward: the AMIN oracle can be implemented in $\mathrm{poly}(n, k)$ time whenever either the MIN oracle or the SMIN oracle can be implemented in $\mathrm{poly}(n, k)$ time. We record these simple observations in remarks for easy recall.

Remark 6.3.5 (MIN implies AMIN)**.** *For any accuracy $\varepsilon > 0$, the $\mathsf{MIN}_C(p)$ oracle provides a valid answer to the $\mathsf{AMIN}_C(p, \varepsilon)$ oracle by definition.*

Remark 6.3.6 (SMIN implies AMIN)**.** *For any $p \in \mathbb{R}^{n\times k}$ and regularization $\eta \geqslant \varepsilon^{-1} k \log n$, the $\mathsf{SMIN}_C(p, \eta)$ oracle provides a valid answer to the $\mathsf{AMIN}_C(p, \varepsilon)$ oracle due to the approximation property of the* smin *operator (Lemma 6.2.3).*

In the remainder of this section, we show a separation between the SMIN oracle and both the MIN and AMIN oracles by exhibiting a family of cost tensors C for which there exist polynomial-time algorithms for MIN and AMIN, however there is no polynomial-time algorithm for SMIN. The non-existence of a polynomial-time algorithm of course requires a complexity theoretic assumption; our result holds under #BIS-hardness—which is a by-now standard complexity theory assumption introduced in [82], and in words is the statement that there does not exist a polynomial-time algorithm for counting the number of independent sets in a bipartite graph.

Lemma 6.3.7 (Restrictiveness of the SMIN oracle)**.** *There exists a family of costs $C \in (\mathbb{R}^n)^{\otimes k}$ for which MIN_C and AMIN_C can be solved in* $\mathrm{poly}(n, k)$ *time, however SMIN_C is #BIS-hard.*

Proof. In order to prove hardness for general n, it suffices to exhibit such a family of cost tensors when $n = 2$. Since $n = 2$, it is convenient to abuse notation slightly by indexing a cost tensor $C \in (\mathbb{R}^n)^{\otimes k}$ by $\vec{j} \in \{-1, 1\}^k$ rather than by $\vec{j} \in \{1, 2\}^k$. The family we exhibit is $\{C(A, b) : A \in \mathbb{R}^{k\times k}_{\geqslant 0},\ b \in \mathbb{R}^k\}$, where the cost tensors $C(A, b)$ are parameterized by a non-negative square matrix A and a vector b, and have entries of the form

$$C_{\vec{j}}(A, b) := -\langle \vec{j}, A\vec{j}\rangle - \langle b, \vec{j}\rangle, \qquad \vec{j} \in \{\pm 1\}^k.$$

Polynomial-time algorithm for MIN and AMIN. We show that given a matrix $A \in \mathbb{R}^{k\times k}_{\geqslant 0}$, vector $b \in \mathbb{R}^k$, and weights $p \in \mathbb{R}^{2\times k}$, it is possible to compute $\mathsf{MIN}_C(p)$ on the cost tensor $C(A, b)$ in poly(k) time. Clearly this also implies a poly(k) time algorithm for $\mathsf{AMIN}_C(p, \varepsilon)$ for any $\varepsilon > 0$, see Remark 6.3.5.

To this end, we first re-write the $\mathsf{MIN}_C(p)$ problem on $C(A, b)$ in a more convenient form that enables us to "ignore" the weights p. Recall that $\mathsf{MIN}_C(p)$ is the problem of

$$\mathsf{MIN}_C(p) = \min_{\vec{j}\in\{\pm 1\}^k} -\langle \vec{j}, A\vec{j}\rangle - \langle b, \vec{j}\rangle - \sum_{i=1}^{k} [p_i]_{j_i}.$$

Note that the linear part of the cost is equal to

$$\langle b, \vec{j}\rangle + \sum_{i=1}^{k} [p_i]_{j_i} = \langle \ell, \vec{j}\rangle + d, \tag{6.4}$$

where $\ell \in \mathbb{R}^k$ is the vector with entries $\ell_i = b_i + \frac{1}{2}((p_i)_1 - (p_i)_{-1})$, and d is the scalar $d = \frac{1}{2}\sum_{i=1}^{k}([p_i]_1 + [p_i]_{-1})$. Thus, since d is clearly computable in $O(k)$ time, the MIN_C problem is equivalent to solving

$$\min_{\vec{j}\in\{\pm 1\}^k} -\langle \vec{j}, A\vec{j}\rangle - \langle \ell, \vec{j}\rangle, \tag{6.5}$$

when given as input a non-negative matrix $A \in \mathbb{R}^{k\times k}_{\geqslant 0}$ and a vector $\ell \in \mathbb{R}^k$.

To show that this task is solvable in poly(k) time, note that the objective in (6.5) is a submodular function because it is a quadratic whose Hessian $-A$ has non-positive off-diagonal terms [27, Proposition 6.3]. Therefore (6.5) is a submodular optimization problem, and thus can be solved in poly(k) time using classical algorithms from combinatorial optimization [102, Chapter 10.2].

SMIN oracle is #BIS-hard. On the other hand, by using the definition of the SMIN oracle, the re-parameterization (6.4), and then the definition of the softmin

operator, the value of $\mathsf{SMIN}_C(p, \eta)$ is

$$\operatorname*{smin}_{\eta \atop \vec{j}\in\{\pm 1\}^k} -\langle \vec{j}, A\vec{j}\rangle - \langle b, \vec{j}\rangle - \sum_{i=1}^{k} [p_i]_{j_i} = \operatorname*{smin}_{\eta \atop \vec{j}\in\{\pm 1\}^k} -\langle \vec{j}, A\vec{j}\rangle - \langle \ell, \vec{j}\rangle - d = -\frac{\log Z}{\eta} - d,$$

where $Z = \sum_{\vec{j}\in\{\pm 1\}^k} Q(\vec{j})$ is the partition function of the ferromagnetic Ising model with inconsistent external fields given by

$$Q(\vec{j}) = \exp\left(\eta\langle \vec{j}, A\vec{j}\rangle + \eta\langle \ell, \vec{j}\rangle\right).$$

Because it is #BIS hard to compute the partition function Z of a ferromagnetic Ising model with inconsistent external fields [97], it is #BIS hard to compute the value $-\eta^{-1}\log Z - d$ of the oracle $\mathsf{SMIN}_C(p, \eta)$. □

Remark 6.3.8 (The restrictiveness of SMIN extends to approximate computation)**.** *The separation between the oracles shown in Lemma 6.3.7 further extends to approximate computation of the SMIN oracle under the assumption that #BIS is hard to approximate, since under this assumption it is hard to approximate the partition function of a ferromagnetic Ising model with inconsistent external fields [97].*

6.4 Algorithms to oracles

In this section, we consider three algorithms for MOT. Each is iterative and requires only polynomially many iterations. The key issue for each algorithm is the per-iteration runtime, which is in general exponential (roughly n^k). We isolate the respective bottlenecks of these three algorithms into the three variants of the dual feasibility oracle defined in §6.3. See §6.1.1 and Table 6.1 for a high-level overview of this section's results.

6.4.1 The Ellipsoid algorithm and the MIN oracle

Among the most classical algorithms for implicit LP is the Ellipsoid algorithm [101, 102, 132]. However it has taken a back seat to the `SINKHORN` algorithm in the vast majority of the MOT literature. The very recent paper [11], which focuses on the specific MOT application of computing low-dimensional Wasserstein barycenters, develops a variant of the classical Ellipsoid algorithm specialized to MOT; henceforth this is called `ELLIPSOID`, see §6.4.1.1 for a description of this algorithm. The objective of this section is to analyze `ELLIPSOID` in the context of general MOT problems in order to prove the following.

Theorem 6.4.1. *For any family of cost tensors $C \in (\mathbb{R}^n)^{\otimes k}$, the following are equivalent:*

(i) ELLIPSOID *takes* $\text{poly}(n,k)$ *time to solve the* MOT_C *problem. (Moreover, it outputs a vertex solution represented as a sparse tensor with at most* $nk-k+1$ *non-zeros.)*

(ii) There exists a $\text{poly}(n,k)$ *time algorithm that solves the* MOT_C *problem.*

(iii) There exists a $\text{poly}(n,k)$ *time algorithm that solves the* MIN_C *problem.*

Interpretation of results. In words, the equivalence "(i) $\iff$ (ii)" establishes that ELLIPSOID can solve any MOT problem in polynomial time that any other algorithm can. Thus from a theoretical perspective, this chapter's restriction to ELLIPSOID is at no loss of generality for developing polynomial-time algorithms that exactly solve MOT. In words, the equivalence "(ii) $\iff$ (iii)" establishes that the MOT and MIN problems are polynomial-time equivalent. Thus we may investigate when MOT is tractable by instead investigating the more amenable question of when MIN is tractable (see §6.1.1.4) at no loss of generality.

As stated in the Related Work section, Theorem 6.4.1 is implicit from combining several known results [11, 12, 102]. Our contribution here is to make this result explicit, since this allows us to unify algorithms from the implicit LP literature with the SINKHORN algorithm. We also significantly simplify part of the implication "(iii) $\implies$ (i)", which is crucial for making an algorithm that relies on the MIN oracle practical—namely, the Column Generation algorithm discussed below.

Organization of §6.4.1. In §6.4.1.1, we recall this ELLIPSOID algorithm and how it depends on the violation oracle for (MOT-D). In §6.4.1.2, we give a significantly simpler proof that the violation and feasibility oracles are polynomial-time equivalent in the case of (MOT-D), and use this to prove Theorem 6.4.1. In §6.4.1.3, we describe a practical implementation that replaces the ELLIPSOID outer loop with Column Generation.

■ 6.4.1.1 Algorithm

A key component of the proof of Theorem 6.4.1 is the ELLIPSOID algorithm introduced in [11] for MOT, which we describe below. In order to present this, we first define a variant of the MIN oracle that returns a minimizing tuple rather than the minimizing value.

Definition 6.4.2 (Violation oracle for (MOT-D))**.** *Given weights* $p = (p_1, \dots, p_k) \in \mathbb{R}^{n \times k}$*,* ARGMIN_C *returns the minimizing solution* $\vec{j}$ *and value of* $\min_{\vec{j} \in [n]^k} C_{\vec{j}} - \sum_{i=1}^k [p_i]_{j_i}$.

ARGMIN_C can be viewed as a violation oracle[5] for the decision set to (MOT-D). This is because, given $p = (p_1, \ldots, p_k) \in \mathbb{R}^{n \times k}$, the tuple $\vec{j}$ output by $\mathsf{ARGMIN}_C(p)$ either provides a violated constraint if $C_{\vec{j}} - \sum_{i=1}^{k} [p_i]_{j_i} < 0$, or otherwise certifies p is feasible. In [11] it is proved that MOT can be solved with polynomially many calls to the ARGMIN_C oracle.

Theorem 6.4.3 (`ELLIPSOID` guarantee; Proposition 12 of [11])**.** *Algorithm 6.1 finds an optimal vertex solution for $\mathsf{MOT}_C(\mu)$ using* $\mathrm{poly}(n, k)$ *calls to the ARGMIN_C oracle and* $\mathrm{poly}(n, k)$ *additional time. The solution is returned as a sparse tensor with at most $nk - k + 1$ non-zero entries.*

Input: Cost $C \in (\mathbb{R}^n)^{\otimes k}$, marginals $\mu_1, \ldots, \mu_k \in \Delta_n$
Output: Vertex solution to $\mathsf{MOT}_C(\mu)$

\\ Solve dual
1: Solve (MOT-D) using the Ellipsoid algorithm, with ARGMIN_C as the violation oracle. Let S denote the set of tuples returned by all calls to ARGMIN_C.

\\ Solve primal
2: Solve (6.6) using the Ellipsoid algorithm.

Algorithm 6.1: `ELLIPSOID`: specialization of the classical Ellipsoid algorithm to MOT

Sketch of algorithm. Full details and a proof are in [11]. We give a brief overview here for the convenience of the reader. First, recall from the implicit LP literature that the classical Ellipsoid algorithm can be implemented in polynomial time for an LP with arbitrarily many constraints so long as it has polynomially many variables and the violation oracle for its decision set is solvable in polynomial time [102]. This does not directly apply to the LP (MOT) because that LP has n^k variables. However, it can apply to the dual LP (MOT-D) because that LP only has nk variables.

This suggests a natural two-step algorithm for MOT. First, compute an optimal dual solution by directly applying the Ellipsoid algorithm to (MOT-D). Second, use this dual solution to construct a sparse primal solution. Although this dual-to-primal conversion does not extend to arbitrary LP [37, Exercise 4.17], the paper [11] provides a solution by exploiting the standard-form structure of

[5]Recall that a violation oracle for a polytope $K = \{x : \langle a_i, x \rangle \leqslant b_i, \forall i \in [N]\}$ is an algorithm that given a point p, either asserts p is in K, or otherwise outputs the index i of a violated constraint $\langle a_i, p \rangle > b_i$.

MOT. The procedure is to solve

$$\min_{\substack{P \in \mathcal{M}(\mu_1,\ldots,\mu_k) \\ \text{s.t. } P_{\vec{j}}=0,\ \forall \vec{j} \notin S}} \langle C, P \rangle \tag{6.6}$$

which is the MOT problem restricted to sparsity pattern S, where S is the set of tuples $\vec{j}$ returned by the violation oracle during the execution of step one of Algorithm 6.1. This second step takes $\text{poly}(n, k)$ time using a standard LP solver, because running the Ellipsoid algorithm in the first step only calls the violation oracle $\text{poly}(n, k)$ times, and thus S has $\text{poly}(n, k)$ size, and therefore the LP (6.6) has $\text{poly}(n, k)$ variables and constraints. In [11] it is proved that this produces a primal vertex solution to the original MOT problem.

■ 6.4.1.2 Equivalence of bottleneck to MIN

Although Theorem 6.4.3 shows that `ELLIPSOID` can solve MOT in $\text{poly}(n, k)$ time using the ARGMIN oracle, this is not sufficient to prove the implication "(iii) $\Longrightarrow$ (i)" in Theorem 6.4.1. In order to prove that implication requires showing the polynomial-time equivalence between MIN and ARGMIN.

Lemma 6.4.4 (Equivalence of MIN and ARGMIN)**.** *Each of the oracles $\textsf{MIN}_C$ and $\textsf{ARGMIN}_C$ can be implemented using* $\text{poly}(n, k)$ *calls of the other oracle and* $\text{poly}(n, k)$ *additional time.*

This equivalence follows from classical results about the equivalence of violation and feasibility oracles [239]. However, the known proof of that general result requires an involved and indirect argument based on "back-and-forth" applications of the Ellipsoid algorithm [102, §4.3]. Here we exploit the special structure of MOT to give a direct and elementary proof. This is essential to practical implementations (see §6.4.1.3).

Proof. It is obvious how the $\textsf{MIN}_C$ oracle can be implemented via a single call of the $\textsf{ARGMIN}_C$ oracle; we now show the converse. Specifically, given $p_1, \ldots, p_k \in \mathbb{R}^n$, we show how to compute a solution $\vec{j} = (j_1, \ldots, j_k) \in [n]^k$ for $\textsf{ARGMIN}_C([p_1, \ldots, p_k])$ using nk calls to the $\textsf{MIN}_C$ oracle and polynomial additional time. We use the first n calls to compute the first index j_1 of the solution, the next n calls to compute the next index j_2, and so on.

Formally, for $s \in [k]$, let us say that $(j_1^*, \ldots, j_s^*) \in [n]^s$ is a "partial solution" of size s if there exists a solution $j \in [n]^k$ for $\textsf{ARGMIN}_C([p_1, \ldots, p_k])$ that satisfies $j_i = j_i^*$ for all $i \in [s]$. Then it suffices to show that for every $s \in [k]$, it is possible to compute a partial solution $(j_1^*, \ldots, j_s^*)$ of size s from a partial solution $(j_1^*, \ldots, j_{s-1}^*)$ of size $s - 1$ using n calls to the $\textsf{MIN}_C$ oracle and polynomial additional time.

The simple but key observation enabling this is the following. Below, for $i \in [k]$ and $j \in [n]$, define $q_{i,j}$ to be the vector in $\mathbb{R}^n$ with value $[p_i]_j$ on entry j, and value $-M$ on all other entries. In words, the following observation states that if the constant M is sufficiently large, then for any indices j'_i, replacing the vectors p_i with the vectors q_{i,j'_i} in a MIN oracle query effectively performs a MIN oracle query on the original input $p_1, \ldots, p_k$ except that now the minimization is only over $\vec{j} \in [n]^k$ satisfying $j_i = j'_i$.

Observation 6.4.5. *Set* $M := 2C_{\max} + 2\sum_{i=1}^{k} \|p_i\|_{\max} + 1$. *Then for any* $s \in [k]$ *and any* $(j'_1, \ldots, j'_s) \in [n]^s$,

$$\mathit{MIN}_C([q_{1,j'_1}, \ldots, q_{s,j'_s}, p_{s+1}, \ldots, p_k]) = \min_{\substack{\vec{j} \in [n]^k \\ s.t.\ j_1 = j'_1, \ldots, j_s = j'_s}} C_{\vec{j}} - \sum_{i=1}^{k} [p_i]_{j_i}.$$

Proof. By definition of the MIN oracle,

$$\mathsf{MIN}_C([q_{1,j'_1}, \ldots, q_{s,j'_s}, p_{s+1}, \ldots, p_k]) = \min_{\vec{j} \in [n]^k} C_{\vec{j}} - \sum_{i=1}^{s} [q_{i,j'_i}]_{j_i} - \sum_{i=s+1}^{k} [p_i]_{j_i}$$

It suffices to prove that every minimizing tuple $\vec{j} \in [n]^k$ for the right hand side satisfies $j_i = j'_i$ for all $i \in [s]$. Suppose not for sake of contradiction. Then there exists a minimizing tuple $\vec{j} \in [n]^k$ for which $j_\ell \neq j'_\ell$ for some $\ell \in [s]$. But then $[q_{\ell,j'_\ell}]_{j_\ell} = -M$, so the objective value of $\vec{j}$ is at least

$$C_{\vec{j}} - \sum_{i=1}^{s} [q_{i,j'_i}]_{j_i} - \sum_{i=s+1}^{k} [p_i]_{j_i} \geqslant M - C_{\max} - \sum_{i=1}^{k} \|p_i\|_{\max} = C_{\max} + \sum_{i=1}^{k} \|p_i\|_{\max} + 1.$$

But this is strictly larger (by at least 1) than the value of any tuple with prefix $(j'_1, \ldots, j'_s)$, contradicting the optimality of $\vec{j}$. □

Thus, given a partial solution $(j^*_1, \ldots, j^*_{s-1})$ of length $s-1$, we construct a partial solution $(j^*_1, \ldots, j^*_s)$ of length s by setting j^*_s to be a minimizer of

$$\min_{j'_s \in [n]} \mathsf{MIN}_C([q_{1,j^*_1}, \ldots, q_{s-1,j^*_{s-1}}, q_{s,j'_s}, p_{s+1}, \ldots, p_k]). \tag{6.7}$$

The runtime bound is clear; it remains to show correctness. To this end, note that

$$\begin{aligned}
\min_{\substack{\vec{j}\in[n]^k \\ \text{s.t. } j_1=j_1^*,\ldots,j_s=j_s^*}} C_{\vec{j}} - \sum_{i=1}^{k}[p_i]_{j_i} &= \mathsf{MIN}_C([q_{1,j_1^*},\ldots,q_{s,j_s^*},p_{s+1},\ldots,p_k]) \\
&= \min_{j_s'\in[n]} \mathsf{MIN}_C([q_{1,j_1^*},\ldots,q_{s-1,j_{s-1}^*},q_{s,j_s'},p_{s+1},\ldots,p_k]) \\
&= \min_{j_s'\in[n]} \min_{\substack{\vec{j}\in[n]^k \\ \text{s.t. } j_1=j_1^*,\ldots,j_{s-1}=j_{s-1}^*,j_s=j_s'}} C_{\vec{j}} - \sum_{i=1}^{k}[p_i]_{j_i} \\
&= \min_{\substack{\vec{j}\in[n]^k \\ \text{s.t. } j_1=j_1^*,\ldots,j_{s-1}=j_{s-1}^*}} C_{\vec{j}} - \sum_{i=1}^{k}[p_i]_{j_i} \\
&= \mathsf{MIN}_C([p_1,\ldots,p_k]),
\end{aligned}$$

where above the first and third steps are by Observation 6.4.5, the second step is by construction of j_s^*, the fourth step is by simplifying, and the final step is by definition of $(j_1^*,\ldots,j_{s-1}^*)$ being a partial solution of size $s-1$. We conclude that $(j_1^*,\ldots,j_s^*)$ is a partial solution of size s, as desired. □

We can now conclude the proof of the main result of §6.4.1.

Proof of Theorem 6.4.1. The implication "(i) $\implies$ (ii)" is trivial, and the implication "(ii) $\implies$ (iii)" is shown in [12]. It therefore suffices to show the implication "(iii) $\implies$ (i)". This follows from combining the fact that `ELLIPSOID` solves MOT_C in polynomial time given an efficient implementation of ARGMIN_C (Theorem 6.4.3), with the fact that the MIN_C and ARGMIN_C oracles are polynomial-time equivalent (Lemma 6.4.4). □

■ 6.4.1.3 Practical implementation via Column Generation

Although `ELLIPSOID` enjoys powerful theoretical runtime guarantees, it is slow in practice because the classical Ellipsoid algorithm is. Nevertheless, whenever `ELLIPSOID` is applicable (i.e., whenever the MIN_C oracle can be efficiently implemented), we can use an alternative practical algorithm, namely the delayed Column Generation method `COLGEN`, to compute exact, sparse solutions to MOT.

For completeness, we briefly recall the idea behind `COLGEN`; for further details see the standard textbook [37, §6.1]. `COLGEN` runs the Simplex method, keeping only basic variables in the tableau. Each time that `COLGEN` needs to find a Simplex variable on which to pivot, it solves the "pricing problem" of finding a variable with negative reduced cost. This is the key subroutine in `COLGEN`. In the

present context of the MOT LP, this pricing problem is equivalent to a call to the ARGMIN violation oracle (see [37, Definition 3.2] for the definition of reduced costs). By the polynomial-time equivalence of the ARGMIN and MIN oracles shown in Lemma 6.4.4, this bottleneck subroutine in `COLGEN` can be computed in polynomial time whenever the MIN oracle can. For easy recall, we summarize this discussion as follows.

Theorem 6.4.6 (Standard guarantee for `COLGEN`; §6.1 of [37])**.** *For any $T > 0$, one can implement T iterations of `COLGEN` in* $\mathrm{poly}(n, k, T)$ *time and calls to the* MIN_C *oracle. When `COLGEN` terminates, it returns an optimal vertex solution, which is given as a sparse tensor with at most $nk - k + 1$ non-zero entries.*

Note that `COLGEN` does not have a theoretical guarantee stating that it terminates after a polynomial number of iterations. But it often performs well in practice and terminates after a small number of iterations, leading to much better empirical performance than `ELLIPSOID`.

■ 6.4.2 The Multiplicative Weights Update algorithm and the AMIN oracle

The second classical algorithm for solving implicitly-structured LPs that we study in the context of MOT is the Multiplicative Weights Update algorithm `MWU` [237]. The objective of this section is to prove the following guarantees for its specialization to MOT.

Theorem 6.4.7. *For any family of cost tensors $C \in (\mathbb{R}^n)^{\otimes k}$, the following are equivalent:*

(i) For any $\varepsilon > 0$, `MWU` takes $\mathrm{poly}(n, k, C_{\max}/\varepsilon)$ *time to solve the* MOT_C *problem ε-approximately. (Moreover, it outputs a sparse solution with at most* $\mathrm{poly}(n, k, C_{\max}/\varepsilon)$ *non-zero entries.)*

(ii) There exists a $\mathrm{poly}(n, k, C_{\max}/\varepsilon)$*-time algorithm that solves the* MOT_C *problem ε-approximately for any $\varepsilon > 0$.*

(iii) There exists a $\mathrm{poly}(n, k, C_{\max}/\varepsilon)$*-time algorithm that solves the* AMIN_C *problem ε-approximately for any $\varepsilon > 0$.*

Interpretation of results. Similarly to the analogous Theorem 6.4.1 for `ELLIPSOID`, the equivalence "(i) $\iff$ (ii)" establishes that `MWU` can approximately solve any MOT problem in polynomial time that any other algorithm can. Thus, from a theoretical perspective, restricting to `MWU` for approximately solving MOT problems is at no loss of generality. In words, the equivalence "(ii) $\iff$ (iii)" establishes that approximating MOT and approximating MIN are polynomial-time equivalent.

Thus we may investigate when MOT is tractable to approximate by instead investigating the more amenable question of when MIN is tractable (see §6.1.1.4) at no loss of generality.

Theorem 6.4.7 is new to this work. In particular, equivalences between problems with polynomially small error do not fall under the purview of classical LP theory, which deals with exponentially small error [102]. Our use of the `MWU` algorithm exploits a simple reduction of MOT to a mixed packing-covering LP that has appeared in the $k = 2$ matrix case of Optimal Transport in [39, 182], where implicit LP is not necessary for polynomial runtime.

Organization of §6.4.2. In §6.4.2.1 we present the specialization of Multiplicative Weights Update to MOT, and recall how it runs in polynomial time and calls to a certain bottleneck oracle. In §6.4.2.2, we show that this bottleneck oracle is equivalent to the AMIN oracle, and then use this to prove Theorem 6.4.7.

■ 6.4.2.1 Algorithm

Here we present the `MWU` algorithm, which combines the generic Multiplicative Weights Update algorithm of [237] specialized to MOT, along with a final rounding step that ensures feasibility of the solution.

In order to present `MWU`, it is convenient to assume that the cost C has entries in the range $[1, 2] \subset \mathbb{R}$, which is at no loss of generality by simply translating and rescaling the cost (see §6.4.2.2), and can be done implicitly given a bound on $C_{\max}$. This is why in the rest of this subsection, every runtime dependence on ε is polynomial in $1/\varepsilon$ for costs in the range $[1, 2]$; after transformation back to $[-C_{\max}, C_{\max}]$, this is polynomial dependence in the standard scale-invariant quantity $C_{\max}/\varepsilon$.

Since the cost C is assumed to have non-negative entries, for any $\lambda \in [1, 2]$, the polytope

$$K(\lambda) = \{P \in \mathcal{M}(\mu_1, \ldots, \mu_k) : \langle C, P\rangle \leqslant \lambda\}$$

of couplings with cost at most λ is a mixed packing-covering polytope (i.e., all variables are non-negative and all constraints have non-negative coefficients). Note that $K(\lambda)$ is non-empty if and only if $\mathsf{MOT}_C(\mu)$ has value at most λ. Thus, modulo a binary search on λ, this reduces computing the value of $\mathsf{MOT}_C(\mu)$ to the task of detecting whether $K(\lambda)$ is empty. Since $K(\lambda)$ is a mixed packing-covering polytope, the Multiplicative Weights Update algorithm of [237] determines whether $K(\lambda)$ is empty, and runs in polynomial time apart from one bottleneck, which we now define.

In order to define the bottleneck, we first define a potential function. For this,

we define the softmax analogously to the softmin as

$$\operatorname{smax}(a_1, \ldots, a_t) = -\operatorname{smin}(-a_1, \ldots, -a_t) = \log\left(\sum_{i=1}^{t} e^{a_i}\right).$$

Here we use regularization parameter $\eta = 1$ for simplicity, since this suffices for analyzing `MWU`, and thus we have dropped this index η for shorthand.

Definition 6.4.8 (Potential function for `MWU`)**.** *Fix a cost $C \in (\mathbb{R}^n)^{\otimes k}$, target marginals $\mu \in (\Delta_n)^k$, and target value $\lambda \in \mathbb{R}$. Define the potential function $\Phi := \Phi_{C,\mu,\lambda} : (\mathbb{R}^n_{\geqslant 0})^{\otimes k} \to \mathbb{R}$ by*

$$\Phi(P) = \operatorname{smax}\left(\frac{\langle C, P\rangle}{\lambda}, \frac{m_1(P)}{\mu_1}, \ldots, \frac{m_k(P)}{\mu_k}\right).$$

The softmax in the above expression is interpreted as a softmax over the $nk + 1$ values in the concatenation of vectors and scalars in its input. (This slight abuse of notation significantly reduces notational overhead.)

Given this potential function, we now define the bottleneck operation for `MWU`: find a direction $\vec{j} \in [n]^k$ in which P can be increased such that the potential is increased as little as possible.

Definition 6.4.9 (Bottleneck oracle for `MWU`)**.** *Given iterate $P \in (\mathbb{R}^n_{\geqslant 0})^{\otimes k}$, target marginals $\mu \in (\Delta_n)^k$, target value $\lambda \in \mathbb{R}$, and accuracy $\varepsilon > 0$, the algorithm $\mathsf{MWU\_BOTTLENECK}_C(P, \mu, \lambda, \varepsilon)$ either:*

- *Outputs "null", certifying that $\min_{\vec{j} \in [n]^k} \frac{\partial}{\partial h}\Phi(P + h \cdot \delta_{\vec{j}}) \mid_{h=0} > 1$, or*
- *Outputs $\vec{j} \in [n]^k$ such that $\frac{\partial}{\partial h}\Phi(P + h \cdot \delta_{\vec{j}}) \mid_{h=0} \leqslant 1 + \varepsilon$.*

(If $\min_{\vec{j} \in [n]^k} \frac{\partial}{\partial h}\Phi(P + h \cdot \delta_{\vec{j}}) \mid_{h=0}$ is within $(1, 1 + \varepsilon]$, then either return behavior is a valid output.)

Pseudocode for the `MWU` algorithm is given in Algorithm 6.2. We prove that `MWU` runs in polynomial time given access to this bottleneck oracle.

Theorem 6.4.10. *Let the entries of the cost C lie in the range $[1, 2]$. Given $\lambda \in \mathbb{R}$ and accuracy parameter $\varepsilon > 0$, `MWU` either certifies that $\mathsf{MOT}_C(\mu) \geqslant \lambda$, or returns a $\operatorname{poly}(n, k, 1/\varepsilon)$-sparse $P \in \mathcal{M}(\mu_1, \ldots, \mu_k)$ satisfying $\langle C, P\rangle \leqslant \lambda + 8\varepsilon$.*

Furthermore, the loop in Step 1 runs in $\tilde{O}(nk/\varepsilon^2)$ iterations, and Step 2 runs in $\operatorname{poly}(n, k, 1/\varepsilon)$ time.

Require: Accuracy $\varepsilon > 0$, marginals $\mu_1, \dots, \mu_k \in \Delta_n$, target value $\lambda > 0$
Ensure: Either certifies $\mathsf{MOT}_C(\mu) > \lambda$ by returning "infeasible", or returns a solution P with $\langle C, P\rangle \leqslant \lambda + 8\,\varepsilon$

\\ Step 1: Multiplicative Weights Update
$P \leftarrow 0 \in (\mathbb{R}^n_{\geqslant 0})^{\otimes k}$, $\eta \leftarrow 2(\log(nk+1))/\varepsilon$
while $m(P) < \eta$ **do** ▷ While total mass is small
  $\vec{j} \leftarrow \mathsf{MWU\_BOTTLENECK}_C(P, \mu, \lambda, \varepsilon)$ ▷ Find good direction
  if $\vec{j}$ ="null" **then return** "infeasible" ▷ No good direction
  else $P \leftarrow P + \delta_{\vec{j}} \cdot \varepsilon \cdot \min(\lambda / C_{\vec{j}}, \min_i [\mu_i]_{j_i})$ ▷ Increase in good direction
$P \leftarrow P/(\eta(1+\varepsilon)^4)$ ▷ Rescale

\\ Step 2: round to transportation polytope
while $m(P) < 1$ **do** ▷ While infeasible
  $j_i \leftarrow \arg\max_{j \in [n]}([\mu_i]_j - [m_i(P)]_j), \quad \forall i \in [k]$ ▷ Find violated constraints
  $\alpha \leftarrow \min_{i \in [k]}([\mu_i]_{j_i} - [m_i(P)]_{j_i})$ ▷ Maximal mass to add
  $P \leftarrow P + \alpha \cdot \delta_{\vec{j}}$ ▷ Add mass to saturate constraints

Algorithm 6.2: `MWU`: specialization of Multiplicative Weights Update [237] to MOT. Assumes cost C satisfies $C_{\vec{j}} \in [1, 2]$ for all $\vec{j} \in [n]^k$ (wlog by rescaling). Step 1 repeatedly adds mass to P in directions that do not increase the potential much. Step 2 repeatedly finds violated constraints and adds mass to P to saturate them.

The `MWU` algorithm can be used to output a $O(\varepsilon)$-approximate solution for MOT time via an outer loop that performs binary search over λ; this only incurs $O(\log(1/\varepsilon))$-multiplicative overhead in runtime.

Proof. We analyze Step 1 (Multiplicative Weights Update) and Step 2 (rounding) of `MWU` separately.

Lemma 6.4.11 (Correctness of Step 1)**.** *Step 1 of Algorithm 6.2 runs in $\tilde{O}(nk/\varepsilon^2)$ iterations. It either returns (i) "infeasible", certifying that $K(\lambda)$ is empty; or (ii) finds a $\mathrm{poly}(n, k, 1/\varepsilon)$-sparse tensor $P \in (\mathbb{R}^n_{\geqslant 0})^{\otimes k}$ that is approximately in $K(\lambda)$, i.e., P satisfies:*

$$m(P) \geqslant 1 - 4\,\varepsilon, \quad \langle C, P\rangle \leqslant \lambda, \quad \text{and} \quad m_i(P) \leqslant \mu_i \text{ for all } i \in [k]$$

Step 1 is the Multiplicative Weights Update algorithm of [237] applied to the polytope $K(\lambda)$, so correctness follows from the analysis of [237]. We briefly recall the main idea behind this algorithm for the convenience of the reader. The main idea behind the algorithm is that on each iteration, $\vec{j} \in [n]^k$ is chosen so that the increase in the potential $\Phi(P)$ is approximately bounded by the increase in the total mass $m(P)$. If this is impossible, then the bottleneck oracle returns null,

which means $K(\lambda)$ is empty. So assume otherwise. Then once the total mass has increased to $m(P) = \eta + O(\varepsilon)$, the potential $\Phi(P)$ must be bounded by $\eta(1+O(\varepsilon))$. By exploiting the inequality between the max and the softmax, this means that $\max(\langle C, P\rangle/\lambda, \max_{i\in[n],j\in[k]}[m_i(P)]_j/[\mu_i]_j) \leqslant \Phi(P) \leqslant \eta(1 + O(\varepsilon))$ as well. Thus, rescaling P by $1/(\eta(1 + O(\varepsilon)))$ in Line 6 satisfies $m(P) \geqslant 1 - O(\varepsilon)$, $\langle C, P\rangle/\lambda \leqslant 1$, and $m_i(P)/\mu_i \leqslant 1$. For full details and a proof of the runtime and sparsity claims, see the Appendix of the paper [13] upon which this chapter is based.

Lemma 6.4.12 (Correctness of Step 2)**.** *Step 2 of Algorithm 6.2 runs in* $\mathrm{poly}(n, k, 1/\varepsilon)$ *time and returns* $P \in \mathcal{M}(\mu_1, \ldots, \mu_k)$ *satisfying* $\langle C, P\rangle \leqslant \lambda + 8\,\varepsilon$*. Furthermore,* P *only has* $\mathrm{poly}(n, k, 1/\varepsilon)$ *non-zero entries.*

Proof of Lemma 6.4.12. By Lemma 6.4.11, P satisfies $m_i(P) \leqslant \mu_i$ for all $i \in [k]$. Observe that this is an invariant that holds throughout the execution of Step 2. This, along with the fact that $\sum_{j=1}^{n}[m_i(P)]_j = m(P)$ is equal for all i, implies that the indices $(j_1, \ldots, j_k)$ found in Line 8 satisfy $[\mu_i]_{j_i} - [m_i(P)]_{j_i} > 0$ for each $i \in [k]$. Thus in particular $\alpha > 0$ in Line 9. It follows that Line 10 makes at least one more constraint satisfied (in particular the constraint "$[\mu_i]_{j_i} = [m_i(P)]_{j_i}$" where i is the minimizer in Line 9). Since there are nk constraints total to be satisfied, Step 2 terminates in at most nk iterations. Each iteration increases the number of non-zero entries in P by at most one, thus P is $\mathrm{poly}(n, k, 1/\varepsilon)$ sparse throughout. That P is sparse also implies that each iteration can be performed in $\mathrm{poly}(n, k, 1/\varepsilon)$ time, thus Step 2 takes $\mathrm{poly}(n, k, 1/\varepsilon)$ time overall.

Finally, we establish the quality guarantee on $\langle C, P\rangle$. By Lemma 6.4.11, this is at most λ before starting Step 2. During Step 2, the total mass added to P is equal to $1 - m(P)$. This is upper bounded by $4\,\varepsilon$ by Lemma 6.4.11. Since $C_{\max} \leqslant 2$, we conclude that the value of $\langle C, P\rangle$ is increased by at most $8\,\varepsilon$ in Step 2. □

Combining Lemmas 6.4.11 and 6.4.12 concludes the proof of Theorem 6.4.10. □

■ 6.4.2.2 Equivalence of bottleneck to AMIN

In order to prove Theorem 6.4.7, we show that the `MWU` algorithm can be implemented in polynomial time and calls to the AMIN oracle. First, we prove this fact for the ARGAMIN oracle, which differs from the AMIN oracle in that it also returns a tuple $\vec{j} \in [n]^k$ that is an approximate minimizer.

Definition 6.4.13 (Approximate violation oracle for (MOT-D))**.** *Given weights* $p = (p_1, \ldots, p_k) \in \mathbb{R}^{n\times k}$ *and accuracy* $\varepsilon > 0$*,* $\textsc{ArgAmin}_C$ *returns* $\vec{j} \in [n]^k$ *that*

minimizes $\min_{\vec{j}\in[n]^k} C_{\vec{j}} - \sum_{i=1}^k [p_i]_{j_i}$ *up to additive error* ε*, and its value up to additive error* ε*.*

Lemma 6.4.14. *Let the entries of the cost* C *lie in the range* $[1,2]$*. The* `MWU` *algorithm (Algorithm 6.2), can be implemented by* $\mathrm{poly}(n,k,1/\varepsilon)$ *time and calls to the* $\mathsf{ARGAMIN}_C$ *oracle with accuracy parameter* $\varepsilon' = \Theta(\varepsilon^2/(nk))$*.*

Proof. We show that on each iteration of Step 1 of Algorithm 6.2 we can emulate the call to the $\mathsf{MWU\_BOTTLENECK}$ oracle with one call to the $\mathsf{ARGAMIN}$ oracle. Recall that $\mathsf{MWU\_BOTTLENECK}_C(P,\mu,\lambda,\varepsilon)$ seeks to find $\vec{j}\in[n]^k$ such that

$$V_{\vec{j}} := \frac{\partial}{\partial h}\Phi(P + h\delta_{\vec{j}})\Big|_{h=0}$$

is at most $1+\varepsilon$, or to certify that for all $\vec{j}$ it is greater than 1. By explicit computation,

$$\begin{aligned} V_{\vec{j}} &= \frac{\partial}{\partial h}\log\left(\exp\left(\left(\langle C,P\rangle + hC_{\vec{j}}\right)/\lambda + \sum_{s=1}^k\sum_{t=1}^n \exp\left(\left([m_s(P)]_t + \delta_{t,j_s}\right)/[\mu_s]_t\right)\right)\right)\Bigg|_{h=0} \\ &= \left(C_{\vec{j}} - \sum_{i=1}^k [p_i]_{j_i}\right)\frac{\exp(\langle C,P\rangle/\lambda)/\lambda}{\exp(\langle C,P\rangle/\lambda) + \sum_{s=1}^k\sum_{t=1}^n \exp([m_s(P)]_t/[\mu_s]_t)}, \end{aligned} \tag{6.8}$$

where the weights $p = (p_1,\ldots,p_k)\in\mathbb{R}^{k\times n}$ in the last line are defined as

$$[p_i]_j = -\frac{\lambda}{\exp(\langle C,P\rangle/\lambda)}\cdot\frac{\exp([m_i(P)]_j/[\mu_i]_j)}{[\mu_i]_j}, \qquad \forall i\in[k], j\in[n].$$

Note that the second term in the product in (6.8) is positive and does not depend on $\vec{j}$. This suggests that in order to minimize (6.8), it suffices to compute $\vec{j}\leftarrow \mathsf{ARGAMIN}_C(p,\varepsilon')$ for some accuracy parameter $\varepsilon'>0$.

The main technical difficulty with formalizing this intuitive approach is that the weights p are not necessarily efficiently computable. Nevertheless, using $\mathrm{poly}(n,k)$ extra time on each iteration, we can compute the marginals $m_1(P),\ldots,m_k(P)$. Since the $\mathsf{ARGAMIN}$ oracle returns an ε'-additive approximation of the cost, we can also compute a running estimate $\tilde{c}$ of the cost such that, on iteration T,

$$\tilde{c} - T\,\varepsilon' \leqslant \langle C,P\rangle \leqslant \tilde{c} + T\,\varepsilon'\,.$$

Therefore, we define weights $\tilde{p}\in\mathbb{R}^{n\times k}$, which approximate p and which can be computed in $\mathrm{poly}(n,k)$ time on each iteration:

$$[\tilde{p}_i]_j = -\frac{\lambda}{\exp(\tilde{c}/\lambda)}\cdot\frac{\exp([m_i(P)]_j/[\mu_i]_j)}{[\mu_i]_j}, \qquad \forall i\in[k], j\in[n].$$

We also define the approximate value for any $\vec{j} \in [n]^k$:

$$\tilde{V}_{\vec{j}} := \left(C_{\vec{j}} - \sum_{i=1}^{k} [\tilde{p}_i]_{j_i} \right) \frac{\exp(\tilde{c}/\lambda)/\lambda}{\exp(\tilde{c}/\lambda) + \sum_{s=1}^{k} \sum_{t=1}^{n} \exp([m_s(P)]_t/[\mu_s]_t)}$$

It holds that $\mathsf{ARGAMIN}_C(\tilde{p}, \varepsilon')$ returns a $\vec{j} \in [n]^k$ that minimizes $C_{\vec{j}} - \sum_{i=1}^{k} [\tilde{p}_i]_j$ up to multiplicative error $1/(1-\varepsilon')$, because the entries of the cost C are lower-bounded by 1, and $[\tilde{p}_i]_j \leqslant 0$ for all $i \in [n], j \in [k]$. In particular, $\mathsf{ARGAMIN}_C(\tilde{p}, \varepsilon')$ minimizes $\tilde{V}_{\vec{j}}$ up to multiplicative error $1/(1-\varepsilon')$. We prove the following claim relating $V_{\vec{j}}$ and $\tilde{V}_{\vec{j}}$:

Observation 6.4.15. *For any $\vec{j} \in [n]^k$, on iteration T, it holds that $V_{\vec{j}}/\tilde{V}_{\vec{j}} \in [\exp(-2T\,\varepsilon'\,/\lambda), \exp(2T\,\varepsilon'\,/\lambda)]$.*

By the above claim, therefore $\mathsf{ARGAMIN}_C(\tilde{p}, \varepsilon')$ minimizes $V_{\vec{j}}$ up to multiplicative error $\exp(4T\,\varepsilon'\,/\lambda)/(1-\varepsilon') \leqslant (1 + \varepsilon\,/3)$ if we choose $\varepsilon' = \Omega(\lambda\,\varepsilon\,/T)$. Thus one can implement $\mathsf{MWU\_BOTTLENECK}_C(p, \mu, \lambda, \varepsilon)$ by returning the value of $\mathsf{ARGAMIN}_C(\tilde{p}, \varepsilon')$ if its value is estimated to be at most $1 + \varepsilon\,/3$, and returning "null" otherwise. The bound on the accuracy $\varepsilon' = \tilde{\Omega}(\varepsilon^2\,/(nk))$ follows since $\lambda \in [1, 2]$ follows since $\lambda \in [1, 2]$ and $T = \tilde{O}(nk/\,\varepsilon^2)$ by Theorem 6.4.10.

Proof of Claim. We compare the expressions for $V_{\vec{j}}$ and $\tilde{V}_{\vec{j}}$. Each of these is a product of two terms. Since $C_{\vec{j}} \geqslant 0$, and $[\tilde{p}_i]_{j_i}, [p_i]_{j_i} \leqslant 0$ for all i, the ratio of the first terms is

$$\frac{C_{\vec{j}} - \sum_{i=1}^{k} [\tilde{p}_i]_{j_i}}{C_{\vec{j}} - \sum_{i=1}^{k} [p_i]_{j_i}} \in [\min_i [\tilde{p}_i]_{j_i}/[p_i]_{j_i}, \max_i [\tilde{p}_i]_{j_i}/[p_i]_{j_i}] \subset [\exp(-T\,\varepsilon'\,/\lambda), \exp(T\,\varepsilon'\,/\lambda)],$$

where we have used that, for all $i \in [k]$,

$$[\tilde{p}_i]_{j_i}/[p_i]_{j_i} = \exp(\langle C, P \rangle/\lambda)/\exp(\tilde{c}/\lambda) \in [\exp(-T\,\varepsilon'\,/\lambda), \exp(T\,\varepsilon'\,/\lambda)].$$

Similarly the ratio of the second terms in the expression for $V_{\vec{j}}$ and $\tilde{V}_{\vec{j}}$ is also in the range $[\exp(-T\,\varepsilon'\,/\lambda), \exp(T\,\varepsilon'\,/\lambda)]$. This concludes the proof of the claim. □

□

Finally, we show that the ARGAMIN oracle can be reduced to the AMIN oracle, which completes the proof that `MWU` can be run with AMIN.

Lemma 6.4.16 (Equivalence of AMIN and ARGAMIN)**.** *Each of the oracles AMIN_C and $\mathsf{ARGAMIN}_C$ with accuracy parameter $\varepsilon > 0$ can be implemented using* $\mathrm{poly}(n, k)$ *calls of the other oracle with accuracy parameter $\Theta(\varepsilon\,/k)$ and* $\mathrm{poly}(n, k)$ *additional time.*

It is worth remarking that the equivalence that we show between AMIN and ARGAMIN is *not* known to hold for the feasibility and separation oracles of general LPs, since the known result for general LPs requires exponentially small error in nk [102, §4.3]. However, in the case of MOT the equivalence follows from a direct and practical reduction, similar to the proof of the equivalence of the exact oracles (Lemma 6.4.4). The main difference is that some care is needed to bound the propagation of the errors of the approximate oracles. For a full proof of Lemma 6.4.16, see the Appendix of the paper [13] upon which this chapter is based.

We conclude by proving Theorem 6.4.7.

Proof of Theorem 6.4.7. The implication "(i) $\implies$ (ii)" is trivial, and the implication "(ii) $\implies$ (iii)" is shown in [12]. It therefore suffices to show the implication "(iii) $\implies$ (i)". For costs C with entries in the range $[1, 2]$, this follows from combining the fact that MWU can be implemented to solve MOT_C in $\mathrm{poly}(n, k, 1/\varepsilon)$ time given an efficient implementation of $\mathsf{ARGAMIN}_C$ with polynomially-sized accuracy parameter $\varepsilon' = \mathrm{poly}(1/n, 1/k, \varepsilon)$ (Lemma 6.4.14), along with the fact that the AMIN_C and $\mathsf{ARGAMIN}_C$ oracles are polynomially-time equivalent with polynomial-sized accuracy parameter (Lemma 6.4.16).

The assumption that C has entries within the range $[1, 2]$ can be removed with no loss by translating and rescaling the original cost $C' \leftarrow (C+3C_{\max})/(2C_{\max})$ and running Algorithm 6.2 on C' with approximation parameter $\varepsilon' \leftarrow \varepsilon/(2C_{\max})$. Each τ'-approximate query to the $\mathsf{AMIN}_{C'}$ oracle can be simulated by a τ-approximate query to the AMIN_C oracle, where $\tau = 2C_{\max}\tau'$. □

Remark 6.4.17 (Practical optimizations)**.** *Our numerical implementation of* MWU *has two modifications that provide practical speedups. One is maintaining a cached list of the tuples $\vec{j} \in [n]^k$ previously returned by calls to* MWU_BOTTLENECK*. Whenever* MWU_BOTTLENECK *is called, we first check whether any tuple $\vec{j}$ in the cache satisfies the desiderata $\frac{\partial}{\partial h}\Phi(P + h \cdot \delta_{\vec{j}}) \mid_{h=0} \leqslant 1 + \varepsilon$, in which case we use this $\vec{j}$ to answer the oracle query. Otherwise, we answer the oracle query using* AMIN *as explained above. In practice, this cache allows us to avoid many calls to the potentially expensive* AMIN *bottleneck. Our second optimization is that, at each iteration of* MWU*, we check whether the current iterate P can be rescaled in order to satisfy the guarantees in Lemma 6.4.11 required from Step 1. If so, we stop Step 1 early and use this rescaled P.*

■ 6.4.3 The Sinkhorn algorithm and the SMIN oracle

The Sinkhorn algorithm (SINKHORN) is specially tailored to MOT, and does not apply to general exponential-size LP. Currently it is by far the most popular

algorithm in the MOT literature (see §6.1.3). However, in general each iteration of SINKHORN takes exponential time $n^{\Theta(k)}$, and it is not well-understood when it can be implemented in polynomial-time. The objective of this section is to show that this bottleneck is polynomial-time equivalent to the SMIN oracle, and in doing so put SINKHORN on equal footing with classical implicit LP algorithms in terms of their reliance on variants of the dual feasibility oracle for MOT. Concretely, this lets us establish the following two results.

First, SINKHORN can solve MOT in polynomial time whenever SMIN can be solved in polynomial time.

Theorem 6.4.18. *For any family of cost tensors $C \in (\mathbb{R}^n)^{\otimes k}$ and accuracy $\varepsilon > 0$, SINKHORN solves MOT_C to ε accuracy in $\mathrm{poly}(n, k, C_{\max}/\varepsilon)$ time and $\mathrm{poly}(n, k, C_{\max}/\varepsilon)$ calls to the SMIN_C oracle with regularization $\eta = (2k \log n)/\varepsilon$. (The solution is output through a polynomial-size implicit representation, see §6.4.3.1.)*

Second, we show that SINKHORN requires strictly more structure than other algorithms do to solve an MOT problem. This is why the results about ELLIPSOID (Theorem 6.4.1) and MWU (Theorem 6.4.7) state that those algorithms solve an MOT problem whenever possible, whereas Theorem 6.4.18 cannot be analogously extended to such an "if and only if" characterization.

Theorem 6.4.19. *There is a family of cost tensors $C \in (\mathbb{R}^n)^{\otimes k}$ for which ELLIPSOID solves MOT_C exactly in $\mathrm{poly}(n, k)$ time, however it is #BIS-hard to implement a single iteration of SINKHORN in $\mathrm{poly}(n, k)$ time.*

Organization of §6.4.3. In §6.4.3.1, we recall this SINKHORN algorithm and how it depends on a certain marginalization oracle. In §6.4.3.2, we show that this marginalization oracle is polynomial-time equivalent to the SMIN oracle, and use this to prove Theorems 6.4.18 and 6.4.19.

■ 6.4.3.1 Algorithm

Here we recall SINKHORN and its known guarantees. To do this, we first define the following oracle. While this oracle does not have an interpretation as a dual feasibility oracle, we show below that it is polynomial-time equivalent to SMIN, which is a specific type of approximate dual feasibility oracle (Remark 6.3.6).

Definition 6.4.20 (MARG)**.** *Given scalings $d = (d_1, \ldots, d_k) \in \mathbb{R}^{n \times k}_{\geqslant 0}$, regularization $\eta > 0$, and an index $i \in [k]$, the marginalization oracle $\mathsf{MARG}_C(d, \eta, i)$ returns the vector $m_i((\otimes_{i'=1}^{k} d_{i'}) \odot \exp[-\eta C]) \in \mathbb{R}^n_{\geqslant 0}$.*

It is known that SINKHORN can solve MOT with only polynomially many calls to this oracle [144]. The approximate solution that SINKHORN computes is a fully dense tensor with n^k non-zero entries, but it is output implicitly in $O(nk)$ space through "scaling vectors" and "rounding vectors", described below.

Theorem 6.4.21 (SINKHORN guarantee, [144])**.** *Algorithm 6.3 computes an ε-approximate solution to $\mathsf{MOT}_C(\mu)$ using $\mathrm{poly}(n, k, C_{\max}/\varepsilon)$ calls to the MARG_C oracle with parameter $\eta = (2k \log n)/\varepsilon$, and $\mathrm{poly}(n, k, C_{\max}/\varepsilon)$ additional time. The solution is of the form*

$$P = \left(\otimes_{i=1}^{k} d_i\right) \odot \exp[-\eta C] + \left(\otimes_{i=1}^{k} v_i\right), \tag{6.9}$$

and is output implicitly via the scaling vectors $d_1, \ldots, d_k \in \mathbb{R}^n_{\geqslant 0}$ and rounding vectors $v_1, \ldots, v_k \in \mathbb{R}^n_{\geqslant 0}$.

Input: Cost $C \in (\mathbb{R}^n)^{\otimes k}$, marginals $\mu_1, \ldots, \mu_k \in \Delta_n$
Output: Implicit representation of tensor (6.9) that is an ε-approximate solution to $\mathsf{MOT}_C(\mu)$

\\ Step 1: scale
$d_1, \ldots, d_k \leftarrow \mathbf{1}$ and $\eta \leftarrow (2k \log n)/\varepsilon$ ▷ Initialize (no scaling)
for $\mathrm{poly}(n, k, C_{\max}/\varepsilon)$ iterations **do**
  Choose $i \in [k]$ ▷ Round-robin, greedily, or randomly
  $\tilde{\mu}_i \leftarrow \mathsf{MARG}_C(d, \eta, i)$ ▷ Bottleneck: compute i-th marginal
  $d_i \leftarrow d_i \odot (\mu_i / \tilde{\mu}_i)$ ▷ Rescale i-th marginal (division is entrywise)

\\ Step 2: round to transportation polytope
for $i = 1, \ldots, k$ **do** ▷ Rescale each marginal to be below marginal constraints
  $\tilde{\mu}_i \leftarrow \mathsf{MARG}_C(d, \eta, i)$ ▷ Bottleneck: compute i-th marginal
  $d_i \leftarrow d_i \odot \min[\mathbf{1}, \mu_i / \tilde{\mu}_i]$ ▷ Rescale i-th marginal (operations are entrywise)
$v_i \leftarrow \mu_i - \mathsf{MARG}_C(d, \eta, i)$ for each $i \in [k]$ ▷ Add back mass
$v_1 \leftarrow v_1 / \|v\|_1^{k-1}$ ▷ Rescale so that (6.9) is feasible
return $d_1, \ldots, d_k$ and $v_1, \ldots, v_k$ ▷ Implicit representation of solution (6.9)

Algorithm 6.3: SINKHORN: multidimensional analog of classical Sinkhorn scaling

Sketch of algorithm. Full details and a proof are in [144]. We give a brief overview here for completeness. The main idea of SINKHORN is to solve RMOT, the entropically regularized variant of MOT described in §6.2.2. On one hand, this provides an ε-approximate solution to MOT by taking the regularization parameter $\eta = \Theta(\varepsilon^{-1} k \log n)$ sufficiently high (Lemma 6.2.4). On the other hand, solving RMOT rather than MOT enables exploiting the first-order optimality conditions

of RMOT, which imply that the unique solution to RMOT is the unique tensor in $\mathcal{M}(\mu_1, \dots, \mu_k)$ of the form

$$P^* = (\otimes_{i=1}^k d_i^*) \odot K, \tag{6.10}$$

where K denotes the entrywise exponentiated tensor $\exp[-\eta C]$, and $d_1^*, \dots, d_k^* \in \mathbb{R}^n_{\geqslant 0}$ are non-negative vectors. The SINKHORN algorithm approximately computes this solution in two steps.

The first and main step of Algorithm 6.3 is the natural multimarginal analog of the Sinkhorn scaling algorithm [203]. It computes an approximate solution $P = (\otimes_{i=1}^k d_i) \odot K$ by finding $d_1, \dots, d_k$ such that P is nearly feasible in the sense that $m_i(P) \approx \mu_i$ for each $i \in [k]$. Briefly, it does this via alternating optimization: initialize d_i to the all-ones vector $\mathbf{1} \in \mathbb{R}^n$, and then iteratively update one d_i so that the i-th marginal $m_i(P)$ of the current scaled iterate $P = (\otimes_{i=1}^k d_i) \odot K$ is μ_i. Although correcting one marginal can detrimentally affect the others, this algorithm nevertheless converges—in fact, in a polynomial number of iterations [144].

The second step of Algorithm 6.3 is the natural multimarginal analog of the rounding algorithm [19, Algorithm 2]. It rounds the solution $P = (\otimes_{i=1}^k d_i) \odot K$ found in step one to the transportation polytope $\mathcal{M}(\mu_1, \dots, \mu_k)$. Briefly, it performs this by scaling each marginal $m_i(P)$ to be entrywise less than the desired μ_i, and then adding mass back to P so that all marginals constraints are exactly satisfied. The former adjustment is done by adjusting the diagonal scalings $d_1, \dots, d_k$, and the latter adjustment is done by adding a rank-1 term $\otimes_{i=1}^k v_i$.

Critically, note that Algorithm 6.3 takes polynomial time except for possibly the calls to the MARG_C oracle. In the absence of structure in the cost tensor C, evaluating this MARG_C oracle takes exponential time because it requires computing marginals of a tensor with n^k entries.

We conclude this discussion with several remarks about SINKHORN.

Remark 6.4.22 (Choice of update index in SINKHORN)**.** *In line 3 there are several ways to choose update indices, all of which lead to the polynomial iteration complexity we desire. Iteration-complexity bounds are shown for a greedy choice in [89, 144]. Similar bounds can be shown for random and round-robin choices by adapting the techniques of [17]. These latter two choices do not incur the overhead of k MARG computations per iteration required by the greedy choice, which is helpful in practice. Empirically, we observe that round-robin works quite well, and we use this in our experiments.*

Remark 6.4.23 (Alternative implementations of SINKHORN)**.** *For simplicity, Algorithm 6.3 provides pseudocode for the "vanilla" version of SINKHORN as it performs well in practice and it achieves the polynomial iteration complexity we desire.*

There are several variants in the literature, including accelerated versions and first rounding small entries of the marginals—these variants have iteration-complexity bounds with better polynomial dependence on ε and k, albeit sometimes at the expense of larger polynomial factors in n [144, 220].

■ 6.4.3.2 Equivalence of bottleneck to SMIN

Although Theorem 6.4.21 shows that SINKHORN solves MOT in polynomial time using the MARG oracle, this is neither sufficient to prove the implication "(ii) $\Longrightarrow$ (i)" in Theorem 6.4.18, nor to prove Theorem 6.4.19. In order to prove these results, we show that SMIN and MARG are polynomial-time equivalent.

Lemma 6.4.24 (Equivalence of MARG and SMIN)**.** *For any regularization $\eta > 0$, each of the oracles MARG_C and SMIN_C can be implemented using* $\mathrm{poly}(n)$ *calls of the other oracle and* $\mathrm{poly}(n, k)$ *additional time.*

Proof. Reduction from SMIN to MARG. First, we show how to compute $\mathsf{SMIN}_C(p, \eta)$ using one call to the marginalization oracle and $O(n)$ additional time. Consider the entrywise exponentiated matrix $d = \exp[\eta p] \in \mathbb{R}^{n\times k}_{\geqslant 0}$, and let $\mu_1 = m_1((\otimes_{i=1}^k d_i) \odot \exp[-\eta C])$ be the answer to $\mathsf{MARG}_C(d, \eta, 1)$. Observe that

$$\begin{aligned}
-\eta^{-1}\log\left(\sum_{j_1=1}^n [\mu_1]_{j_1}\right) &= -\eta^{-1}\log\left(\sum_{j_1=1}^n \sum_{j_2,\ldots,j_k\in[n]} \prod_{i=1}^k [d_i]_{j_i} e^{-\eta C_{\vec{j}}}\right) \\
&= -\eta^{-1}\log\left(\sum_{\vec{j}\in[n]^k} e^{-\eta(C_{\vec{j}} - \sum_{i=1}^k [p_i]_{j_i})}\right) \\
&= \operatorname*{smin}_{\eta,\ \vec{j}\in[n]^k}\left(C_{\vec{j}} - \sum_{i=1}^k [p_i]_{j_i}\right),
\end{aligned}$$

where above the first step is by definition of μ_1, the second step is by definition of d and combining the sums, and the third step is by definition of smin. We conclude that $-\eta^{-1}\log\sum_{j_1=1}^n [\mu_1]_{j_1}$ is a valid answer to $\mathsf{SMIN}_C(p, \eta)$. Since this is clearly computable from μ_1 in $O(n)$ time, this establishes the claimed reduction.

Reduction from MARG to SMIN. Next, we show that for any marginalization index $i \in [k]$ and entry $\ell \in [n]$, it is possible to compute the ℓ-th entry of the vector $\mathsf{MARG}_C(d, \eta, i)$ using one call to the SMIN_C oracle and $\mathrm{poly}(n, k)$ additional time. Define $v \in \mathbb{R}^n$ to be the vector with ℓ-th entry equal to $[d_i]_\ell$, and all other entries 0. Define the matrix $p = \eta^{-1}\log[d_1, \ldots, d_{i-1}, v, d_{i+1}, \ldots, d_k] \in \bar{\mathbb{R}}^{n\times k}$, where recall that $\log 0 = -\infty$ (see §6.2). Let $s \in \mathbb{R}$ denote the answer to $\mathsf{SMIN}_C(p, \eta)$.

Observe that

$$e^{-\eta s} = \sum_{\vec{j}\in[n]^k} e^{-\eta(C_{\vec{j}} - \sum_{i=1}^k [p_i]_{j_i})} = \sum_{\vec{j}\in[n]^k \,:\, \vec{j}_i = \ell} \prod_{i=1}^k [d_i]_{j_i} e^{-\eta C_{\vec{j}}} = \left[m_i\left((\otimes_{i=1}^k d_i) \odot \exp[-\eta C]\right)\right]_\ell ,$$

where above the first step is by definition of s, the second step is by definition of p and v, and the third step is by definition of the marginalization notation $m_i(\cdot)$. We conclude that $\exp(-\eta s)$ is a valid answer for the ℓ-th entry of the vector $\mathsf{MARG}_C(d, \eta, i)$. This establishes the claimed reduction since we may repeat this procedure n times to compute all n entries of the the vector $\mathsf{MARG}_C(d, \eta, i)$. □

We can now conclude the proofs of the main results of §6.4.3.

Proof of Theorem 6.4.18. This follows from the fact that `SINKHORN` approximates MOT_C in polynomial time given a efficient implementation of MARG_C (Theorem 6.4.21), combined with the fact that the MARG_C and SMIN_C oracles are polynomial-time equivalent (Lemma 6.4.24). □

Proof of Theorem 6.4.19. Consider the family of cost tensors in Lemma 6.3.7 for which the MIN_C oracle admits a polynomial-time algorithm, but for which the SMIN_C oracle is #BIS-hard. Then on one hand, the `ELLIPSOID` algorithm solves MOT_C in polynomial time by Theorem 6.4.1. And on the other hand, it is #BIS-hard to implement a single iteration of `SINKHORN` because that requires implementing the MARG_C oracle, which is polynomial-time equivalent to the SMIN_C oracle by Lemma 6.4.24. □

■ 6.5 Application: MOT problems with graphical structure

In this section, we illustrate our algorithmic framework on MOT problems with graphical structure. Although a polynomial-time algorithm is already known for this particular structure [110, 216], that algorithm computes solutions that are approximate and dense; see the Related Work section for details. By combining our algorithmic framework developed above with classical facts about graphical models, we show that it is possible to compute solutions that are exact and sparse in polynomial time.

The section is organized as follows. In §6.5.1, we recall the definition of graphical structure. In §6.5.2, we show that the MIN, AMIN, and SMIN oracles can be implemented in polynomial time for cost tensors with graphical structure; from this it immediately follows that all of the MOT algorithms discussed in part 1 of this chapter can be implemented in polynomial time. Finally, in §6.5.3,

we demonstrate our results on the popular application of computing generalized Euler flows, which was the original motivation of MOT. Numerical simulations demonstrate how the exact, sparse solutions produced by our new algorithms provide qualitatively better solutions than previously possible in polynomial time.

6.5.1 Setup

We begin by recalling preliminaries about undirected graphical models, a.k.a., Markov Random Fields. We recall only the relevant background; for further details we refer the reader to the textbooks [134, 227].

In words, graphical models provide a way of encoding the independence structure of a collection of random variables in terms of a graph. The formal definition is as follows. Below, all graphs are undirected, and the notation 2^V means the power set of V (i.e., the set of all subsets of V).

Definition 6.5.1 (Graphical model structure)**.** *Let $\mathcal{S} \subset 2^{[k]}$. The graphical model structure corresponding to $\mathcal{S}$ is the graph $G_{\mathcal{S}} = (V, E)$ with vertices $V = [k]$ and edges $E = \{(i, j) : i, j \in S, \text{ for some } S \in \mathcal{S}\}$.*

Definition 6.5.2 (Graphical model)**.** *Let $\mathcal{S} \subset 2^{[k]}$. A probability distribution P over $\{X_i\}_{i\in[k]}$ is a graphical model with structure $\mathcal{S}$ if there exist functions $\{\psi_S\}_{S\in\mathcal{S}}$ and normalizing constant Z such that*

$$P\Big(\{x_i\}_{i\in[k]}\Big) = \frac{1}{Z}\prod_{S\in\mathcal{S}} \psi_S\Big(\{x_i\}_{i\in S}\Big).$$

A standard measure of complexity for graphical models is the treewidth of the underlying graphical model structure $G_{\mathcal{S}}$ because this captures not just the storage complexity, but also the algorithmic complexity of performing fundamental tasks such as computing the mode, log-partition function, and marginal distributions [134, 227]. There are a number of equivalent definitions of treewidth [42]. Each requires defining intermediate combinatorial concepts. We recall here the definition that is based on the concept of a junction tree because this is perhaps the most standard definition in the graphical models community.

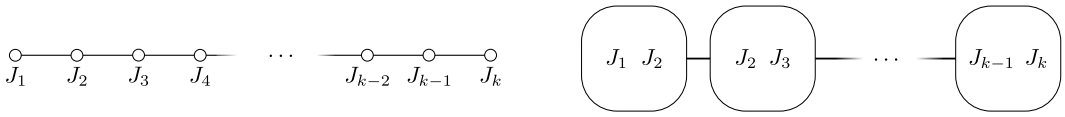
Definition 6.5.3 (Junction tree, treewidth)**.** *A junction tree $T = (V_T, E_T, \{B_u\}_{u\in V_T})$ for a graph $G = (V, E)$ consists of a tree (V_T, E_T) and a set of bags $\{B_u \subseteq V\}_{u\in V_T}$ satisfying:*

- *For each variable $i \in V$, the set of nodes $U_i = \{u \in V_T : i \in B_u\}$ induces a subtree of T.*
- *For each edge $e \in E$, there is some bag B_u containing both endpoints of e.*

The width of the junction tree is one less than the size of the largest bag, i.e., is $\max_{u \in V_T} |B_u| - 1$. *The treewidth of a graph is the width of its minimum-width junction tree.*

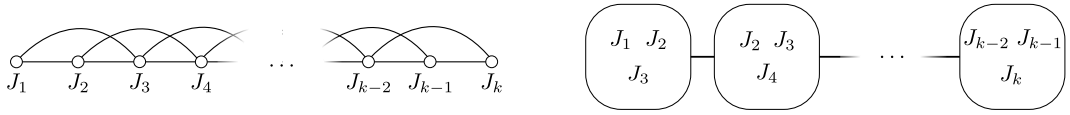
See Figures 6.1, 6.2, and 6.3 for illustrated examples.

We now formally recall the definition of graphical structure for MOT.



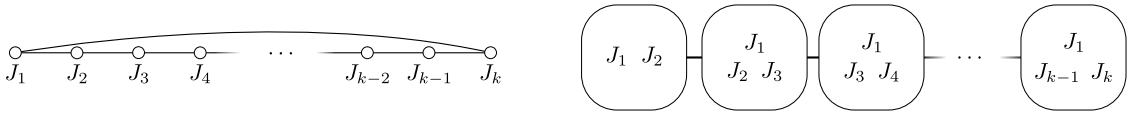
(a) Path graph. (b) Junction tree.

Figure 6.1: The path graph (left) has treewidth 1 because the corresponding junction tree (right) has bags of size at most 2.



(a) Window graph with window size 2. (b) Junction tree.

Figure 6.2: The graph that has an edges between all vertices of distance at most two when ordered sequentially (left) has treewidth 2 because the corresponding junction tree (right) has bags of size at most 3.



(a) Cycle graph. (b) Junction tree.

Figure 6.3: The cycle graph (left) has treewidth 2 because the corresponding junction tree (right) has bags of size at most 3.

Definition 6.5.4 (Graphical structure for MOT)**.** *An MOT cost tensor* $C \in (\mathbb{R}^n)^{\otimes k}$ *has graphical structure with treewidth* ω *if there is a graphical model structure* $\mathcal{S} \subset 2^{[k]}$ *and functions* $\{f_S\}_{S \in \mathcal{S}}$ *such that*

$$C_{\vec{j}} = \sum_{S \in \mathcal{S}} f_S\Big(\{j_i\}_{i \in S}\Big), \qquad \forall \vec{j} := (j_1, \ldots, j_k) \in [n]^k, \tag{6.11}$$

and such that the graph $G_{\mathcal{S}}$ *has treewidth* ω.

We make three remarks about this structure. First, note that the functions $\{f_S\}_{S\in\mathcal{S}}$ can be arbitrary so long as the corresponding graphical model structure has treewidth at most ω.

Second, if Definition 6.5.4 did not constrain the treewidth ω, then every tensor C would trivially have graphical structure with maximal treewidth $\omega = k - 1$ (take $\mathcal{S}$ to be the singleton containing $[k]$, $G_{\mathcal{S}}$ to be the complete graph, and $f_{[k]}$ to be C). Just like all previous algorithms, our algorithms have runtimes that depend exponentially (only) on the treewidth of $G_{\mathcal{S}}$. This is optimal in the sense that unless $\mathsf{P} = \mathsf{NP}$, there is no algorithm with jointly polynomial runtime in the input size and treewidth [12]. We also point out that in all current applications of graphically structured MOT, the treewidth is either 1 or 2, see §6.1.3.

Third, as in all previous work on graphically structured MOT, we make the natural assumptions that the cost C is input implicitly through the functions $\{f_S\}_{S\in\mathcal{S}}$, and that each function f_S can be evaluated in polynomial time, since otherwise graphical structure is useless for designing polynomial-time algorithms. In all applications in the literature, these two basic assumptions are always satisfied. Note also that if the treewidth of the graphical structure is constant, then there is a linear-time algorithm to compute the treewidth and a corresponding minimum-width junction tree [41].

■ 6.5.2 Polynomial-time algorithms

By our oracle reductions in §6.4, in order to design polynomial-time algorithms for MOT with graphical structure, it suffices to design polynomial-time algorithms for the MIN, AMIN, or SMIN oracles. This follows directly from classical algorithmic results in the graphical models literature [134].

Theorem 6.5.5 (Polynomial-time algorithms for the MIN, AMIN, and SMIN oracles for costs with graphical structure)**.** *Let $C \in (\mathbb{R}^n)^{\otimes k}$ be a cost tensor that has graphical structure with constant treewidth ω (see Definition 6.5.4). Then the MIN_C, AMIN_C, and SMIN_C oracles can be computed in* $\mathrm{poly}(n, k)$ *time.*

Input: Cost C with graphical structure, matrix $p \in \mathbb{R}^{n\times k}$
Output: Solution to $\mathsf{MIN}_C(p)$

\\ Use the classical max-product algorithm [134, §13.3]
1: $\vec{j} \leftarrow$ mode of the graphical model P in (6.12)
2: **return** $C_{\vec{j}} - \sum_{i=1}^{k} [p_i]_{j_i}$

Algorithm 6.4: Polynomial-time algorithm for MIN for graphically structured costs (Theorem 6.5.5).

Input: Cost C with graphical structure, matrix $p \in \bar{\mathbb{R}}^{n\times k}$, regularization $\eta > 0$
Output: Solution to $\mathsf{SMIN}_C(p, \eta)$

\\ Use the classical sum-product algorithm [134, §10.2]
1: $Z \leftarrow$ partition function of the graphical model P in (6.12)
2: **return** $-\eta^{-1} \log Z$

Algorithm 6.5: Polynomial-time algorithm for SMIN for graphically structured costs (Theorem 6.5.5).

Proof. Consider input p for the oracles. Let P denote the probability distribution on $[n]^k$ given by

$$P(\vec{j}) = \frac{1}{Z} \exp\left(-\eta\left(C_{\vec{j}} - \sum_{i=1}^{k} [p_i]_{j_i}\right)\right), \qquad \forall \vec{j} \in [n]^k, \tag{6.12}$$

where $Z = \sum_{\vec{j}\in[n]^{[k]}} \exp(-\eta(C_{\vec{j}} - \sum_{i=1}^{k}[p_i]_{j_i}))$ ensures P is normalized. Observe that the MIN_C oracle amounts[6] to computing the mode of the distribution P because $\mathsf{MIN}_C(p) = C_{\vec{j}} - \sum_{i=1}^{k}[p_i]_{j_i}$, where $\vec{j} \in [n]^k$ is a maximizer of $P_{\vec{j}}$. Also, the SMIN_C oracle amounts to computing the partition function Z because $\mathsf{SMIN}_C(p) = -\eta^{-1} \log Z$. Thus it suffices to compute the mode and partition function of P in polynomial time. (The AMIN_C oracle follows from the MIN_C oracle by Remark 6.3.5).

To this end, observe that by assumption on C, there is a graphical model structure $\mathcal{S} \in 2^{[k]}$ and functions $\{f_S\}_{S\in\mathcal{S}}$ such that the corresponding graph $G_{\mathcal{S}}$ has treewidth ω and the distribution P factors as

$$P(\vec{j}) = \exp\left(-\eta\left(\sum_{S\in\mathcal{S}} f_S\left(\{j_i\}_{i\in S}\right) - \sum_{i=1}^{k}[p_i]_{j_i}\right)\right).$$

It follows that P is a graphical model with respect to the same graphical model structure $\mathcal{S}$ because the "vertex potentials" $\exp(\eta[p_i]_{j_i})$ do not affect the underlying graphical model structure. Thus P is a graphical model with constant treewidth ω, so we may compute the mode and partition function of P in $\mathrm{poly}(n, k)$ time using, respectively, the classical max-product and sum-product algorithms [134, Chapters 13.3 and 10.2]. For convenience, pseudocode summarizing this discussion is provided in Algorithms 6.4 and 6.5. □

An immediate consequence of Theorem 6.5.5 combined with our oracle reductions is that all candidate MOT algorithms in §6.4 can be efficiently implemented

[6] In fact, for the purpose of computing MIN_C, the distribution $P(\vec{j})$ can be defined using any $\eta > 0$.

for MOT problems with graphical structure. From a theoretical perspective, ELLIPSOID gives the best guarantee since it produces an exact, sparse solution.

Corollary 6.5.6 (Polynomial-time algorithms for MOT problems with graphical structure)**.** *Let $C \in (\mathbb{R}^n)^{\otimes k}$ be a cost tensor that has graphical structure with constant treewidth ω (see Definition 6.5.4). Then:*

- *The ELLIPSOID algorithm in §6.4.1 computes an exact solution to MOT_C in* $\mathrm{poly}(n, k)$ *time.*
- *The MWU algorithm in §6.4.2 computes an ε-approximate solution to MOT_C in* $\mathrm{poly}(n, k, C_{\max}/\varepsilon)$ *time.*
- *The SINKHORN algorithm in §6.4.3 computes an ε-approximate solution to MOT_C in* $\mathrm{poly}(n, k, C_{\max}/\varepsilon)$ *time.*
- *The COLGEN algorithm in §6.4.1.3 can be run for T iterations in* $\mathrm{poly}(n, k, T)$ *time.*

Moreover, ELLIPSOID, MWU, and COLGEN output a polynomially sparse tensor, whereas SINKHORN outputs a fully dense tensor through the implicit representation described in §6.4.3.1.

Proof. Follows immediately from combining the polynomial-time implementations of the oracles in Theorem 6.5.5 with the polynomial-time algorithm-to-oracle reductions in Theorems 6.4.1, 6.4.7, 6.4.18, and 6.4.6, respectively. □

■ 6.5.3 Application vignette: fluid dynamics

In this section, we numerically demonstrate our new results for graphically structured MOT—namely the ability to compute exact, sparse solutions in polynomial time (Corollary 6.5.6). We illustrate this on the problem of computing generalized Euler flows—an MOT application which has received significant interest and which was historically the motivation of MOT, see e.g., [32, 33, 47, 48, 49, 50]. This MOT problem is already known to be tractable via a popular, specially-tailored modification of SINKHORN [32]—which can be interpreted as implementing SINKHORN using graphical structure [110, 216]. However, that algorithm is based on SINKHORN and thus unavoidably produces solutions that are low-precision (due to $\mathrm{poly}(1/\varepsilon)$ runtime dependence), fully dense (with n^k non-zero entries), and have well-documented numerical precision issues. We offer the first polynomial-time algorithm for computing exact and/or sparse solutions.

We briefly recall the premise of this MOT problem; for further background see [32, 50]. An incompressible fluid (e.g., water) is modeled by n particles

which are uniformly distributed in space (due to incompressibility) at all times $t \in \{1, \dots, k+1\}$. We observe each particle's location at initial time $t = 1$ and final time $t = k+1$. The task is to infer the particles' locations at all intermediate times $t \in \{2, \dots, k\}$, and this is modeled by an MOT problem as follows.

Specifically, the locations of the fluid particles are discretized to points $\{x_j\}_{j \in [n]} \subset \mathbb{R}^d$, and σ is a known permutation on this set that encodes the relation between each initial location x_j at time $t = 1$ and final location $\sigma(x_j)$ at time $t = k+1$. The total movement of a particle that takes the trajectory $x_{j_1}, x_{j_2}, \dots, x_{j_k}, \sigma(x_{j_1})$ is given by

$$C_{j_1,\dots,j_k} = \|\sigma(x_{j_1}) - x_{j_k}\|^2 + \sum_{t=1}^{k-1} \|x_{j_{t+1}} - x_{j_t}\|^2, \tag{6.13}$$

By the principle of least action, the generalized Euler flow problem of inferring the most likely trajectories of the fluid particles is given by the solution to the MOT problem with this cost C and uniform marginals $\mu_t = \mathbf{1}_n/n \in \Delta_n$ which impose the constraint that the fluid is incompressible.

Corollary 6.5.7 (Exact, sparse solutions for generalized Euler flows)**.** *The MOT problem with cost* (6.13) *can be solved in* $d \cdot \mathrm{poly}(n, k)$ *time. The solution is returned as a sparse tensor with at most* $nk - k + 1$ *non-zeros.*

Proof. This cost tensor C can be expressed in graphical form $C_{\vec{j}} = \sum_{S \in \mathcal{S}} f_S(\{j_i\})$ where $\mathcal{S}$ consists of the sets $\{1,2\}, \dots, \{k-1, k\}$ of adjacent time points as well as the set $\{1, k\}$. Moreover, each function $f_S : [n]^2 \to \mathbb{R}$ can be computed in $O(dn^2)$ time since this simply requires computing $\|x_j - x_{j'}\|^2$ for n^2 pairs of points $x_j, x_{j'} \in \mathbb{R}^d$. Once this graphical representation is computed, Corollary 6.5.6 implies a $\mathrm{poly}(n, k)$ time algorithm for this MOT problem because the graphical model structure $\mathcal{S}$ is a cycle graph and thus has treewidth 2 (cf., Figure 6.3). □

Figure 6.4 illustrates how the exact, sparse solutions found by our new algorithm provide visually sharper estimates than the popular modification of SINKHORN in [32], which blurs the trajectories. The latter is the state-of-the-art algorithm in the literature and in particular is the only previously known non-heuristic algorithm that has polynomial-time guarantees. Note that this algorithm is identical to implementing SINKHORN by exploiting the graphical structure to perform exact marginalization efficiently [110, 216].

The numerical simulation is on a standard benchmark problem used in the literature (see e.g., [32, Figure 9] and [50, Figure 2]) in which the particle at initial location $x \in [0, 1]$ moves to final location $\sigma(x) = x + \frac{1}{2} \pmod 1$. This is run with

[7] All experiments in this chapter are run on a standard-issue Apple MacBook Pro 2020 laptop with an M1 Chip.

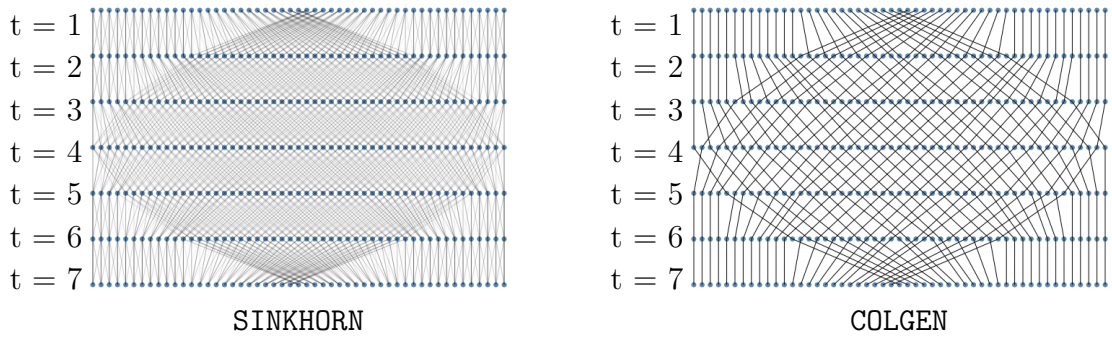


Figure 6.4: Transport maps computed by the fast implementation of `SINKHORN` [32] (left) and our `COLGEN` implementation (right) on a standard fluid dynamics benchmark problem in dimension $d = 1$ [50]. The pairwise transport maps between successive timesteps are plotted with opacity proportional to the mass. The `SINKHORN` algorithm is run at the highest precision (i.e., smallest regularization) before serious numerical precision issues (NaNs). It returns a dense, approximate solution in 2.25 seconds.[7] `COLGEN` returns an exact, sparse solution in 9.52 seconds. Furthermore, in this particular problem instance, the `COLGEN` method returns a Monge solution, i.e., the sparsity is n so that the particles never split in the computed trajectories.

$k = 6$ and marginals $\mu_1 = \cdots = \mu_k$ uniformly supported on $n = 51$ positions in $[0, 1]$. For numerics on other standard benchmark instances, see the Appendix of the paper [13] upon which this chapter is based. Note that this amounts to solving an MOT LP with $n^k = 51^6 \approx 1.8 \times 10^{10}$ variables, which is infeasible for standard LP solvers. Our algorithm is the first to compute exact solutions for problem instances of this scale.

Two important remarks. First, since this MOT problem is a discretization of the underlying PDE, an exact solution is of course not necessary; however, there is an important—even qualitative—difference between low-precision solutions (computable with $\text{poly}(1/\varepsilon)$ runtime) and high-precision solutions (computable with $\text{polylog}(1/\varepsilon)$ runtime) for the discretized problem. Second, a desirable feature of `SINKHORN` that should be emphasized is its practical scalability, which might make it advantageous for problems where very fine discretization is required. It is an interesting direction of practical relevance to develop algorithms that can compute high-precision solutions at a similarly large scale in practice (see the discussion in §6.8).

6.6 Application: MOT problems with set-optimization structure

In this section, we consider MOT problems whose cost tensors C take values 0 and 1—or more generally any two values, by a straightforward reduction[8]. Such MOT problems arise naturally in applications where one seeks to minimize or maximize the probability that some event occurs given marginal probabilities on each variable (see Example 6.6.1). We establish that this general class of MOT problems can be solved in polynomial time under a condition on the sparsity pattern of C that is often simple to check due its connection to classical combinatorial optimization problems.

The section is organized as follows. In §6.6.1 we formally describe this setup and discuss why it is incomparable to all other structures discussed in this chapter. In §6.6.2, we show that for costs with this structure, the MIN, AMIN, and SMIN oracles can be implemented in polynomial time; from this it immediately follows that the `ELLIPSOID`, `MWU`, `SINKHORN`, and `COLGEN` algorithms discussed in part 1 of this chapter can be implemented in polynomial time. In §6.6.3, we illustrate our results via a case study on network reliability.

6.6.1 Setup

Example 6.6.1 (Motivation for binary-valued MOT costs: minimizing/maximizing probability of an event)**.** *Let $S \subset [n]^k$. If $C_{\vec{j}} = \mathbb{1}[\vec{j} \in S]$, then the MOT_C problem amounts to minimizing the probability that event S occurs, given marginals on each variable. On the other hand, if $C_{\vec{j}} = \mathbb{1}[\vec{j} \notin S]$, then the MOT_C problem amounts to maximizing the probability that event S occurs since*

$$\mathsf{MOT}_C(\mu_1, \dots, \mu_k) = \min_{P \in \mathcal{M}(\mu_1,\dots,\mu_k)} \mathbb{P}_{\vec{j}\sim P}[\vec{j} \notin S] = 1 - \max_{P \in \mathcal{M}(\mu_1,\dots,\mu_k)} \mathbb{P}_{\vec{j}\sim P}[\vec{j} \in S].$$

Even if the cost is binary-valued, there is no hope to solve MOT in polynomial time without further assumptions—essentially because in the worst case, any algorithm must query all n^k entries if C is a completely arbitrary $\{0,1\}$-valued tensor.

We show that MOT is polynomial-time solvable under the general and often simple-to-check condition that the MIN, AMIN, and SMIN oracles introduced in §6.3 are polynomial-time solvable when restricted to the set S of indices $\vec{j} \in [n]^k$ for which $C_{\vec{j}} = 0$. For simplicity, our definition of these set oracles removes the cost $C_{\vec{j}}$ as it is constant on S. Of course it is also possible to remove the negative

[8] If C takes two values $a < b$, then define the tensor $\tilde{C}$ with $\{0,1\}$-entries by $\tilde{C}_{\vec{j}} = (C_{\vec{j}} - a)/(b-a)$. It is straightforward to see that the MOT problems with costs C and $\tilde{C}$ have identical solutions.

sign in $-p$ by re-parameterizing the inputs as $w = -p$; however, we keep this notation in order to parallel the original oracles.

Definition 6.6.2 (MIN set oracle)**.** *Let $S \subset [n]^k$. For weights $p = (p_1, \dots, p_k) \in \mathbb{R}^{n \times k}$, $\mathsf{MIN}_{C,S}(p)$ returns*

$$\min_{\vec{j} \in S} -\sum_{i=1}^{k} [p_i]_{j_i}.$$

Definition 6.6.3 (AMIN set oracle)**.** *Let $S \subset [n]^k$. For weights $p = (p_1, \dots, p_k) \in \mathbb{R}^{n \times k}$ and accuracy $\varepsilon > 0$, $\mathsf{AMIN}_{C,S}(p, \varepsilon)$ returns $\mathsf{MIN}_{C,S}(p)$ up to additive error ε.*

Definition 6.6.4 (SMIN set oracle)**.** *Let $S \subset [n]^k$. For weights $p = (p_1, \dots, p_k) \in \bar{\mathbb{R}}^{n \times k}$ and regularization parameter $\eta > 0$, $\mathsf{SMIN}_{C,S}(p, \eta)$ returns*

$$\operatorname{smin}_{\eta, \vec{j} \in S} -\sum_{i=1}^{k} [p_i]_{j_i}.$$

The key motivation behind these set oracle definitions (aside from the syntactic similarity to the original oracles) is that they encode the problem of (approximately) finding the min-weight object in S. This opens the door to combinatorial applications of MOT because finding the min-weight object in S is well-known to be polynomial-time solvable for many "combinatorial-structured" sets S of interest—e.g., the set S of cuts in a graph, or the set S of independent sets in a matroid. See §6.6.3 for fully-detailed applications.

Definition 6.6.5 (Set-optimization structure for MOT)**.** *An MOT cost tensor $C \in (\mathbb{R}^n)^{\otimes k}$ has exact, approximate, or soft set-optimization structure of complexity β if*

$$C_{\vec{j}} = \mathbb{1}[\vec{j} \notin S]$$

for a set $S \subset [n]^k$ for which there is an algorithm solving $\mathsf{MIN}_{C,S}$, $\mathsf{AMIN}_{C,S}$, or $\mathsf{SMIN}_{C,S}$, respectively, in β time.

We make two remarks about this structure.

Remark 6.6.6 (Only require set oracle for $C^{-1}(0)$, not for $C^{-1}(1)$)**.** *Note that Definition 6.6.5 only requires the set oracles for the set S of entries where C is 0, and does* not *need the set oracles for the set $[n]^k \setminus S$ where C is 1. The fact that both set oracles are not needed makes set-optimization structure easier to check than the original oracles in §6.3, because those effectively require optimization over both S and $[n]^k \setminus S$.*

Remark 6.6.7 (Set-optimization structure is incomparable to graphical and low-rank plus sparse structure)**.** *Costs C that satisfy Definition 6.6.5 in general do not have non-trivial graphical structure or low-rank plus sparse structure. Specifically, there are costs C that satisfy Definition 6.6.5, yet require maximal $k-1$ treewidth to model via graphical structure, and super-constant rank or exponential sparsity to model via low-rank plus sparse structure. (A concrete example is the network reliability application in §6.6.3.) Because of the* NP*-hardness of MOT problems with $(k-1)$-treewidth graphical structure or super-constant rank [12], simply modeling such problems with graphical structure or low-rank plus rank structure is therefore useless for the purpose of designing polynomial-time MOT algorithms.*

■ 6.6.2 Polynomial-time algorithms

By our oracle reductions in part 1 of this chapter, in order to design polynomial-time algorithms for MOT with set-optimization structure, it suffices to design polynomial-time algorithms for the MIN, AMIN, or SMIN oracles. We show how to do this for all three oracles in a straightforward way by exploiting the set-optimization structure.

Theorem 6.6.8 (Polynomial-time algorithms for the MIN, AMIN, and SMIN oracles for costs with set-optimization structure)**.** *If $C \in (\mathbb{R}^n)^{\otimes k}$ is a cost tensor with exact, approximate, or soft set-optimization structure of complexity β (see Definition 6.6.5), then the MIN_C, AMIN_C, and SMIN_C oracles, respectively, can be computed in $\beta + \mathrm{poly}(n,k)$ time.*

Input: Access to C via $\mathsf{MIN}_{C,S}$ oracle, matrix $p \in \mathbb{R}^{n\times k}$
Output: Solution to $\mathsf{MIN}_C(p)$

1: $a \leftarrow \mathsf{MIN}_{C,S}(p)$ ▷ One oracle call
2: $x \leftarrow -\sum_{i=1}^{k} \max_{j\in[n]} [p_i]_j$ ▷ Takes $O(nk)$ time
3: **return** a if $a \leqslant x$, or $\min(a, 1+x)$ otherwise ▷ Takes $O(1)$ time

Algorithm 6.6: Polynomial-time algorithm for MIN for costs with exact set-optimization structure (Theorem 6.6.8).

Proof. Polynomial-time algorithm for MIN. We first claim that Algorithm 6.6 implements the $\mathsf{MIN}_C(p)$ oracle. To this end, define

$$a := \mathsf{MIN}_{C,S}(p) = \min_{\substack{\vec{j}\in[n]^k \\ \text{s.t. } C_{\vec{j}}=0}} -\sum_{i=1}^{k} [p_i]_{j_i} \qquad \text{and} \qquad b := \min_{\substack{\vec{j}\in[n]^k \\ \text{s.t. } C_{\vec{j}}=1}} -\sum_{i=1}^{k} [p_i]_{j_i}. \quad (6.14)$$

Input: Access to C via $\mathsf{SMIN}_{C,S}$ oracle, matrix $p \in \bar{\mathbb{R}}^{n\times k}$, regularization η
Output: Solution to $\mathsf{SMIN}_C(p, \eta)$

1: $a \leftarrow \exp(-\eta \cdot \mathsf{SMIN}_{C,S}(p))$ ▷ One oracle call
2: $x \leftarrow \prod_{i=1}^{k} \sum_{j=1}^{n} \exp(\eta[p_i]_{j_i})$ ▷ Takes $O(nk)$ time
3: **return** $-\eta^{-1} \log(e^{-\eta}x + (1 - e^{-\eta})a)$ ▷ Takes $O(1)$ time

Algorithm 6.7: Polynomial-time algorithm for SMIN for costs with soft set-optimization structure (Theorem 6.6.8).

By re-arranging the sum and max, it follows that

$$x := -\sum_{i=1}^{k} \max_{j\in[n]} [p_i]_j = -\max_{\vec{j}\in[n]^k} \sum_{i=1}^{k} [p_i]_{j_i} = \min_{\vec{j}\in[n]^k} \sum_{i=1}^{k} -[p_i]_{j_i} = \min(a, b). \tag{6.15}$$

Therefore

$$\begin{aligned}\mathsf{MIN}_C(p) = \min_{\vec{j}\in[n]^k} C_{\vec{j}} - \sum_{i=1}^{k} [p_i]_{j_i} = \min(a, 1+b) &= \begin{cases} a & \text{if } a \leqslant b \\ \min(a, 1 + \min(a, b)) & \text{if } a > b \end{cases} \\ &= \begin{cases} a & \text{if } a \leqslant x \\ \min(a, 1+x) & \text{if } a > x \end{cases}, \end{aligned} \tag{6.16}$$

where above the first step is by definition of MIN_C; the second step is by partitioning the minimization over $\vec{j} \in [n]^k$ into $\vec{j}$ such that $C_{\vec{j}} = 0$ or $C_{\vec{j}} = 1$, and then plugging in the definitions of a and b; the third step is by manipulating $\min(a, 1 + b)$ in both cases; and the last step is because $x = \min(a, b)$ as shown above. We conclude that Algorithm 6.6 correctly outputs $\mathsf{MIN}_C(p)$. Since the algorithm uses one call to the $\mathsf{MIN}_{C,S}$ oracle and $O(nk)$ additional time, the claim is proven.

Polynomial-time algorithm for AMIN. Next, we claim that the same Algorithm 6.6, now run with the approximate oracle $\mathsf{AMIN}_{C,S}(p, \varepsilon)$ in the first step instead of the exact oracle $\mathsf{MIN}_{C,S}(p)$, computes a valid solution to $\mathsf{AMIN}_C(p, \varepsilon)$. To prove this, let a, b, and x be as defined in (6.14) and (6.15) for the MIN analysis, and let $\tilde{a} = \mathsf{AMIN}_{C,S}(p, \varepsilon)$. By the same logic as in (6.16), except now reversed, the output

$$\begin{cases} \tilde{a} & \text{if } \tilde{a} \leqslant x \\ \min(\tilde{a}, 1 + x) & \text{if } \tilde{a} > x \end{cases}$$

is equal to $\min(\tilde{a}, 1+b)$. Now because $\tilde{a}$ is within additive ε error of a (by definition of the $\mathsf{AMIN}_{C,S}$ oracle), it follows that the above output is within ε additive error of

$$\min(a, 1+b) = \min_{\vec{j} \in [n]^k} C_{\vec{j}} - \sum_{i=1}^{k} [p_i]_{j_i} = \mathsf{MIN}_C(p).$$

Thus the output is a valid answer to $\mathsf{AMIN}_C(p, \varepsilon)$, establishing correctness. The runtime claim is obvious.

Polynomial-time algorithm for SMIN. Finally, we claim that Algorithm 6.7 implements the $\mathsf{SMIN}_C(p, \eta)$ oracle. To this end, define

$$a := e^{-\eta \cdot \mathsf{SMIN}_{C,S}(p,\eta)} = \sum_{\substack{\vec{j} \in [n]^k \\ \text{s.t. } C_{\vec{j}}=0}} e^{\eta \sum_{i=1}^{k} [p_i]_{j_i}} \qquad \text{and} \qquad b := \sum_{\substack{\vec{j} \in [n]^k \\ \text{s.t. } C_{\vec{j}}=1}} e^{\eta \sum_{i=1}^{k} [p_i]_{j_i}}.$$

By re-arranging products and sums, it follows that

$$x := \prod_{i=1}^{k} \sum_{j=1}^{n} e^{\eta [p_i]_{j_i}} = \sum_{\vec{j} \in [n]^k} \prod_{i=1}^{k} e^{\eta [p_i]_{j_i}} = a + b.$$

Therefore

$$\begin{aligned} \mathsf{SMIN}_C(p, \eta) &= -\frac{1}{\eta} \log \left(\sum_{\vec{j} \in [n]^k} e^{-\eta (C_{\vec{j}} - \sum_{i=1}^{k} [p_i]_{j_i})} \right) \\ &= -\frac{1}{\eta} \log \left(a + e^{-\eta} b \right) = -\frac{1}{\eta} \log \left(e^{-\eta} x + (1 - e^{-\eta}) a \right), \end{aligned}$$

where above the first step is by definition of SMIN_C; the second step is by partitioning the sum over $\vec{j} \in [n]^k$ into $\vec{j}$ such that $C_{\vec{j}} = 0$ or $C_{\vec{j}} = 1$, and then plugging in the definitions of a and b; and the third step is because $x = a + b$ as shown above. We conclude that Algorithm 6.7 correctly outputs $\mathsf{SMIN}_C(p, \eta)$. Since the algorithm uses one call to the $\mathsf{SMIN}_{C,S}$ oracle and $O(nk)$ additional time, the claim is proven. □

An immediate consequence of Theorem 6.6.8 combined with our oracle reductions is that all of the candidate MOT algorithms described in §6.4 can be efficiently implemented for MOT problems with set-optimization structure. From a theoretical perspective, the `ELLIPSOID` algorithm gives the best guarantee since it produces an exact, sparse solution in polynomial time.

Corollary 6.6.9 (Polynomial-time algorithms for MOT problems with set-optimization structure)**.** *Let $C \in (\mathbb{R}^n)^{\otimes k}$ be a cost tensor that has set-optimization structure with* $\mathrm{poly}(n,k)$ *complexity (see Definition 6.6.5).*

- *Exact set-optimization structure. The* `ELLIPSOID` *algorithm in §6.4.1 computes an exact solution to* MOT_C *in* $\mathrm{poly}(n,k)$ *time. Also, the* `COLGEN` *algorithm in §6.4.1.3 can be run for T iterations in* $\mathrm{poly}(n,k,T)$ *time.*
- *Approximate set-optimization structure. The* `MWU` *algorithm in §6.4.2 computes an ε-approximate solution to* MOT_C *in* $\mathrm{poly}(n,k,C_{\max}/\varepsilon)$ *time.*
- *Soft set-optimization structure. The* `SINKHORN` *algorithm in §6.4.3 computes an ε-approximate solution to* MOT_C *in* $\mathrm{poly}(n,k,C_{\max}/\varepsilon)$ *time.*

Moreover, `ELLIPSOID`*,* `MWU`*, and* `COLGEN` *output a polynomially sparse tensor, whereas* `SINKHORN` *outputs a fully dense tensor through the implicit representation described in §6.4.3.1.*

Proof. Follows immediately from combining the polynomial-time implementations of the oracles in Theorem 6.7.4 with the polynomial-time algorithm-to-oracle reductions in Theorems 6.4.1, 6.4.6, 6.4.7, and 6.4.18, respectively. □

6.6.3 Application vignette: network reliability with correlations

In this section, we illustrate this class of MOT structures via an application to network reliability, a central topic in network science, engineering, and operations research, see e.g., the textbooks [28, 29, 95]. The basic network reliability question: is given an undirected graph $G = (V, E)$ where each edge $e \in E$ is reliable with some probability q_e and fails with probability $1 - q_e$, what is the probability that all vertices are reachable from all others? This connectivity is desirable in applications, e.g., if G is a computer cluster, the vertices are the machines, and the edges are communication links, then connectivity corresponds to the reachability of all machines. See the aforementioned textbooks for many other applications.

Of course, the above network reliability question is not yet well-defined since the edge failures are only prescribed up to their marginal distributions. *Which joint distribution* greatly impacts the answer.

The most classical setup posits that edge failures are independent [156]. Denote the network reliability probability for this setting by ρ^{ind}. This quantity ρ^{ind} is #P-complete [181, 221] and thus NP-hard to compute, but there exist fully polynomial randomized approximation schemes (a.k.a. FPRAS) for multiplicatively approximating both the connection probability ρ^{ind} [128] and the failure probability $1 - \rho^{\mathrm{ind}}$ [104].

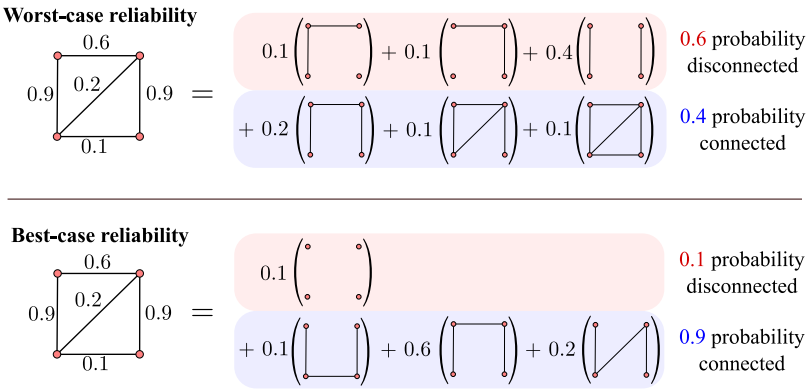


Figure 6.5: Optimal decompositions for the the worst-case (top) and best-case (bottom) reliability problems on the same graph G and edge reliability probabilities q_e (left). Coordinating edge failures yields significantly different connection probabilities: $\rho^{\min} = 40\%$, $\rho^{\text{ind}} \approx 60\%$, and $\rho^{\max} = 90\%$.

Here we investigate the setting of *coordinated* edge failures, which dates back to the 1980s [229, 241]. This coordination may optimize for disconnection (e.g., by an adversary), or for connection (e.g., maximize the time a network is connected while performing maintenance on each edge e during $1 - q_e$ fraction of the time). We define these notions below; see also Figure 6.5 for an illustration. Below, $\text{Ber}(q_e)$ denotes the Bernoulli distribution with parameter q_e.

Definition 6.6.10 (Network reliability with correlations)**.** *For an undirected graph $G = (V, E)$ and edge reliability probabilities $\{q_e\}_{e \in E}$:*

- *The* worst-case network reliability *is*

$$\rho^{\min} := \min_{P \in \mathcal{M}(\{\text{Ber}(q_e)\}_{e \in E})} \mathbb{P}_{H \sim P} \left[H \text{ is a connected subgraph of } G \right].$$

- *The* best-case network reliability *is*

$$\rho^{\max} := \max_{P \in \mathcal{M}(\{\text{Ber}(q_e)\}_{e \in E})} \mathbb{P}_{H \sim P} [H \text{ is a connected subgraph of } G].$$

Clearly $\rho^{\min} \leqslant \rho^{\text{ind}} \leqslant \rho^{\max}$. These gaps can be large (e.g., see Figure 6.5), which promises large opportunities for applications in which coordination is possible. However, in order to realize such an opportunity requires being able to compute $\rho^{\min}$ and $\rho^{\max}$, and both of these problems require solving an exponentially large LP with $2^{|E|}$ variables. Below we show how to use set-optimization structure to compute these quantities in $\text{poly}(|E|)$ time, thereby recovering as a special case of

our general framework the known polynomial-time algorithms for this particular problem in [229, 241], as well as more practical polynomial-time algorithms that scale to input sizes that are an order-of-magnitude larger.

Corollary 6.6.11 (Polynomial-time algorithm for network reliability with correlations)**.** *The worst-case and best-case network reliability can both be computed in* $\mathrm{poly}(|E|)$ *time.*

Proof. By the observation in Example 6.6.1, the optimization problems defining $\rho^{\min}$ and $1-\rho^{\max}$ are instances of MOT in which $k = |E|$, $n = 2$, $\mu_e = \mathrm{Ber}(q_e)$, and each entry of the cost $C \in \{0,1\}^{|E|}$ is the indicator of whether that subset of edges is a connected or disconnected subgraph of G, respectively. It therefore suffices to show that both of these MOT cost tensors satisfy exact set-optimization structure (Definition 6.6.5) since that implies a polynomial-time algorithm for exactly solving MOT (Corollary 6.6.9).

Set-optimization structure for $1-\rho^{\max}$. In this case, S is the set of connected subgraphs of G. Thus the $\mathsf{MIN}_{C,S}$ problem is: given weights $p \in \mathbb{R}^{2\times|E|}$, compute

$$\min_{\text{connected subgraph } H \text{ of } G} -\sum_{e\in H} p_{2,e} - \sum_{e\notin H} p_{1,e}.$$

Note that this objective is equal to $\sum_{e\in H} x_e - \sum_{e\in E} p_{1,e}$ where $x_e := p_{1,e} - p_{2,e}$. Since the latter sum is independent of H, the $\mathsf{MIN}_{C,S}$ problem therefore reduces to the problem of finding a minimum-weight connected subgraph in G; that is, given edge weights $x \in \mathbb{R}^{|E|}$, compute

$$\min_{\text{connected subgraph } H \text{ of } G} \sum_{e\in H} x_e. \tag{6.17}$$

We first show how to solve this in polynomial time in the case that all edge weights x_e are positive. In this case, the optimal solution H is a minimum-weight spanning tree of G. This can be found by Kruskal's algorithm in $O(|E|\log|E|)$ time [137].

For the general case of arbitrary edge weights, note that the edges e with non-positive weight $x \leqslant 0$ can be added to any solution without worsening the cost or feasibility. Thus these edges are without loss of generality in every solution H, and so it suffices to solve the same problem (6.17) on the graph G' obtained by contracting these non-positively-weighted edges in G. This reduces (6.17) to the same problem of finding a minimum-weight connected subgraph, except now in the special case that all edge weights are positive. Since we have already shown how to solve this case in polynomial time, the proof is complete.

Set-optimization structure for $\rho^{\min}$. In this case, S is the set of disconnected subgraphs of G. We may simplify the $\mathsf{MIN}_{C,S}$ problem for $\rho^{\min}$ by re-parameterizing the input $p \in \mathbb{R}^{2\times|E|}$ to edge weights $x \in \mathbb{R}^{|E|}$ as done above in (6.17) for $1-\rho^{\max}$. Thus the $\mathsf{MIN}_{C,S}$ problem for $\rho^{\min}$ is: given weights $x \in \mathbb{R}^{|E|}$, compute

$$\min_{\text{disconnected subgraph } H \text{ of } G} \sum_{e\in H} x_e. \tag{6.18}$$

We first show how to solve this in the case that all edge weights x_e are negative. In that case, the optimal solution is of the form $H = E \setminus C$, where C is a maximum-weight cut of the graph G with weights x_e. Equivalently, by negating all edge weights, C is a minimum-weight cut of the graph G with weights $-x_e$. Since a minimum-weight cut of a graph with positive weights can be found in polynomial time [210], the problem (6.18) can be solved in polynomial time when all x_e are negative.

Now in the general case of arbitrary edge weights, note that the edges e with non-negative weight $x \geqslant 0$ can be removed from any solution without worsening the cost or feasibility. Thus these edges are without loss of generality not in every solution H, and so it suffices to solve the same problem (6.18) on the graph G' obtained by deleting these non-negatively-weighted edges in G. This reduces (6.18) to the same problem of finding a minimum-weight disconnected subgraph, except now in the special case that all edge weights are negative. Since we have already shown how to solve this case in polynomial time, the proof is complete. □

In Figure 6.6, we compare the numerical performance of the algorithms in Corollary 6.6.11—`COLGEN` and `MWU` with polynomial-time implementation of their bottlenecks—with the fastest previous algorithms for both best-case and worst-case network reliability. Previously, the fastest algorithms that apply to this problem are (1) out-of-the-box LP solvers run on MOT, (2) the brute-force implementation of `SINKHORN` which marginalizes over all $n^k = 2^{|E|}$ entries in each iteration, and (3) this `COLGEN` algorithm that we recover [229, 241]. It is unknown if there is a practically efficient implementation of the $\mathsf{SMIN}_{C,S}$ oracle (and thus of `SINKHORN`) for both best-case or worst-case reliability. Since the previous algorithms (1) and (2) have exponential runtime that scales as $n^{\Omega(k)} = 2^{\Omega(|E|)}$, they do not scale past tiny input sizes. In contrast, the algorithms in Corollary 6.6.11 scale to much larger inputs. Indeed, the `COLGEN` algorithm that our framework recovers can compute exact solutions roughly an order-of-magnitude faster than the other algorithms, and the new `MWU` algorithm computes reasonably approximate solutions beyond $k = 400$, which amounts to an MOT LP with $n^k = 2^{400} \approx 2.6 \times 10^{120}$ variables.

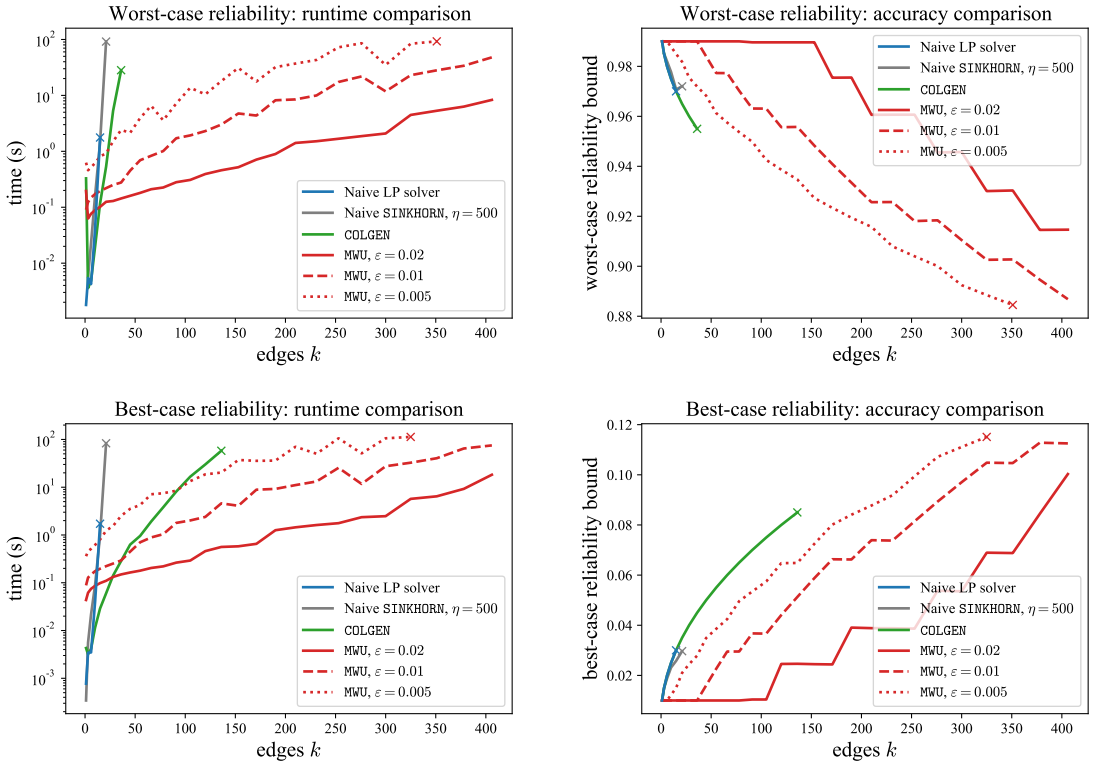


Figure 6.6: Top: comparison of the runtime (left) and accuracy (right) of the algorithms described in the main text, for the worst-case reliability of a clique graph on t vertices and $k = \binom{t}{2}$ edges with reliability probabilities $q_e = 0.99$. Bottom: same, but for best-case reliability and reliability probabilities $q_e = 0.01$. For worst-case reliability, the algorithms compute an upper bound, so smaller value is better; reverse for best-case reliability. The algorithms are cut off at 2 minutes, denoted by an "x". `SINKHORN` is run at the highest precision (i.e., highest η) before numerical precision issues. The `COLGEN` algorithm that our framework recovers computes exact solutions an order-of-magnitude faster than the other algorithms, and the new `MWU` algorithm computes reasonably approximate solutions for $k = 400$, which amounts to an MOT LP with $n^k = 2^{400} \approx 2.6 \times 10^{120}$ variables.

6.7 Application: MOT problems with low-rank plus sparse structure

In this section, we consider MOT problems whose cost tensors C decompose into low-rank and sparse components. We propose the first polynomial-time algorithms for this general class of MOT problems.

The section is organized as follows. In §6.7.1 we formally describe this setup and discuss why it is incomparable to all other structures discussed in this chapter. In §6.7.2, we show that for costs with this structure, the AMIN and SMIN oracles can be implemented in polynomial time; from this it immediately follows that MWU and SINKHORN can be implemented in polynomial time. Finally, in §6.7.3 and §6.7.4, we provide two illustrative applications of these algorithms. The former regards portfolio risk management and is a direct application of our result for MOT with low-rank cost tensors. The latter regards projecting mixture distributions to the transportation polytope and illustrates the versality of our algorithmic results since this problem is quadratic optimization over the transportation polytope rather than linear (a.k.a. MOT).

6.7.1 Setup

We begin by recalling the definition of tensor rank. It is the direct analog of the standard concept of matrix rank. See the survey [133] for further background.

Definition 6.7.1 (Tensor rank)**.** *A rank-r factorization of a tensor $R \in (\mathbb{R}^n)^{\otimes k}$ is a collection of rk vectors $\{u_{i,\ell}\}_{i\in[k],\ell\in[r]} \subset \mathbb{R}^n$ satisfying*

$$R = \sum_{\ell=1}^{r} \bigotimes_{i=1}^{k} u_{i,\ell}.$$

The rank of a tensor is the minimal r for which there exists a rank-r factorization.

In this section we consider MOT problems with the following "low-rank plus sparse" structure.

Definition 6.7.2 (Low-rank plus sparse structure for MOT)**.** *An MOT cost tensor $C \in (\mathbb{R}^n)^{\otimes k}$ has low-rank plus sparse structure of rank r and sparsity s if it decomposes as*

$$C = R + S, \tag{6.19}$$

where R is a rank-r tensor and S is an s-sparse tensor.

Throughout, we make the natural assumption that S is input through its s non-zero entries, and that R is input through a rank-r factorization. We also make

the natural assumption that the entries of both R and S are of size $O(C_{\max})$—this rules out the case of having extremely large entries of R and S, one positive and one negative, which cancel to yield a small entry of $C = R + S$.

Remark 6.7.3 (Neither low-rank structure nor sparse structure can be modeled by graphical structure or set-optimization structure)**.** *In general, both rank-*1 *costs and polynomially sparse costs do not have non-trivial graphical structure. Specifically, modeling these costs with graphical structure requires the complete graph (a.k.a., maximal treewidth of* $k - 1$*)—and because* MOT *problems with graphical structure of treewidth* $k - 1$ *are* NP*-hard to solve in the absence of further structure [12], modeling such problems with graphical structure is useless for the purpose of designing polynomial-time* MOT *algorithms. It is also clear that neither low-rank structure nor sparse structure can be modeled by set-optimization structure because in general, neither* R *nor* S *nor* $R + S$ *has binary-valued entries.*

■ 6.7.2 Polynomial-time algorithms

From a technical perspective, the main result of this section is that there is a polynomial-time algorithm for approximating the minimum entry of a tensor that decomposes into constant-rank and sparse components. Previously, this was not known even for constant-rank tensors. This result may be of independent interest. We remark that this result is optimal in the sense that unless P = NP, there does not exist an algorithm with runtime that is *jointly* polynomial in the input size and the rank r [12].

Theorem 6.7.4 (Polynomial-time algorithm solving AMIN and SMIN for low-rank + sparse costs)**.** *Consider cost tensors* $C \in (\mathbb{R}^n)^{\otimes k}$ *that have low-rank plus sparse structure of rank* r *and sparsity* s *(see Definition 6.7.2). For any fixed* r*, Algorithm 6.8 runs in* $\mathrm{poly}(n, k, s, C_{\max}/\varepsilon)$ *time and solves the* ε*-approximate* AMIN_C *oracle. Furthermore, it also solves the* $\mathsf{SMIN}_{\tilde{C}}$ *oracle for* $\eta = (2k \log n)/\varepsilon$ *on some cost tensor* $\tilde{C} \in (\mathbb{R}^n)^{\otimes k}$ *satisfying* $\|C - \tilde{C}\|_{\max} \leqslant \varepsilon/2$.

We make three remarks about Theorem 6.7.4. First, we are unaware of any polynomial-time implementation of SMIN_C for the cost C. Instead, Theorem 6.7.4 solves the $\mathsf{SMIN}_{\tilde{C}}$ oracle for an $O(\varepsilon)$-approximate cost tensor $\tilde{C}$ since this is sufficient for implementing SINKHORN on the original cost tensor C (see Corollary 6.7.5 below). Second, it is an interesting open question if the $\mathrm{poly}(n, k, C_{\max}/\varepsilon)$ runtime for the ε-approximate AMIN_C oracle can be improved to $\mathrm{poly}(n, k, \log(C_{\max}/\varepsilon))$, as this would imply a $\mathrm{poly}(n, k)$ runtime for the MIN_C oracle and thus for this class of MOT problems (see also Footnote 3 in the introduction). Third, we remark about practical efficiency: the runtime of Algorithm 6.8 is not just polynomially

small in s and n, but in fact linear in s and near-linear in n. However, since this improved runtime is not needed for the theoretical results in the sequel, we do not pursue this further.

Combining the efficient oracle implementations in Theorem 6.7.4 with our algorithm-to-oracles reductions in §6.4 implies the first polynomial-time algorithms for MOT problems with costs that have constant-rank plus sparse structure. This is optimal in the sense that unless P = NP, there does not exist an algorithm with runtime that is *jointly* polynomial in the input size and the rank r [12].

Corollary 6.7.5 (Polynomial-time algorithms solving MOT for low-rank + sparse costs)**.** *Consider cost tensors $C \in (\mathbb{R}^n)^{\otimes k}$ that have low-rank plus sparse structure of constant rank r and* $\mathrm{poly}(n,k)$ *sparsity s (see Definition 6.7.2). For any $\varepsilon > 0$:*

- *The* `MWU` *algorithm in §6.4.2 computes an ε-approximate solution to MOT_C in* $\mathrm{poly}(n, k, C_{\max}/\varepsilon)$ *time.*
- *The* `SINKHORN` *algorithm in §6.4.3 computes an ε-approximate solution to MOT_C in* $\mathrm{poly}(n, k, C_{\max}/\varepsilon)$ *time.*

Moreover, `MWU` *outputs a polynomially sparse tensor, whereas* `SINKHORN` *outputs a fully dense tensor through the implicit representation described in §6.4.3.1.*

Proof. For `MWU`, simply combine the polynomial-time reduction to the AMIN_C oracle (Theorem 6.4.7) with the polynomial-time algorithm for the AMIN oracle (Theorem 6.7.4). For `SINKHORN`, combining the polynomial-time reduction to the $\mathsf{SMIN}_{\tilde{C}}$ oracle (Theorem 6.4.18) with the polynomial-time algorithm for the $\mathsf{SMIN}_{\tilde{C}}$ oracle (Theorem 6.7.4) yields a $\mathrm{poly}(n, k, C_{\max}/\varepsilon)$ algorithm for $\varepsilon/2$-approximating the MOT problem with cost tensor $\tilde{C}$. It therefore suffices to show that the values of the MOT problems with cost tensors C and $\tilde{C}$ differ by at most $\varepsilon/2$, that is,

$$\left| \min_{P \in \mathcal{M}(\mu_1,\ldots,\mu_k)} \langle P, C\rangle - \min_{P \in \mathcal{M}(\mu_1,\ldots,\mu_k)} \langle P, \tilde{C}\rangle \right| \leqslant \varepsilon/2.$$

But this holds because both MOT problems have the same feasible set, and for any feasible $P \in \mathcal{M}(\mu_1, \ldots, \mu_k)$ it follows from Hölder's inequality that the objectives of the two MOT problems differ by at most

$$\left| \langle P, C\rangle - \langle P, \tilde{C}\rangle \right| \leqslant \|P\|_1 \|C - \tilde{C}\|_{\max} \leqslant \varepsilon/2.$$

□

Below, we describe the algorithm in Theorem 6.7.4. Specifically, in §6.7.2.1, we give four helper lemmas which form the technical core of our algorithm; and then in §6.7.2.2, we combine these ingredients to design the algorithm and prove its correctness. Throughout, recall that we use the bracket notation $f[A]$ to denote the *entrwise* application of a univariate function f (e.g., exp, log, or a polynomial) to A.

■ 6.7.2.1 Technical ingredients

At a high level, our approach to designing the algorithm in Theorem 6.7.4 is to approximately compute the SMIN oracle in polynomial time by synthesizing four facts:

1. By expanding the softmin and performing simple operations, it suffices to compute the total sum of all n^k entries of the entrywise exponentiated tensor $\exp[-\eta R]$ (modulo simple transforms).

2. Although $\exp[-\eta R]$ is in general a full-rank tensor, we can exploit the fact that R is a low-rank tensor in order to approximate $\exp[-\eta R]$ by a low-rank tensor L. (Moreover, we can efficiently compute a low-rank factorization of L in closed form.)

3. There is a simple algorithm for computing the sum of all n^k entries of L in polynomial time because L is low-rank. (And thus we may approximate the sum of all n^k entries of $\exp[-\eta R]$ as desired in step 1.)

4. This approximation is sufficient for computing both the AMIN and SMIN oracle in Theorem 6.7.4.

Of these four steps, the main technical step is the low-rank approximation in step two. Below, we formalize these four steps individually in Lemmas 6.7.6, 6.7.7, 6.7.8, and 6.7.9. Further detail on how to synthesize these four steps is then provided afterwards, in the proof of Theorem 6.7.4.

It is convenient to write the first lemma in terms of an approximate tensor $\tilde{C} = \tilde{R} + S$ rather than the original cost $C = R + S$.

Lemma 6.7.6 (Softmin for cost with sparse component)**.** *Let $\tilde{C} = \tilde{R} + S$ and $p_1, \dots, p_k \in \mathbb{R}^n$. Then*

$$\operatorname{smin}_{\eta, \vec{j} \in [n]^k} \tilde{C}_{\vec{j}} - \sum_{i=1}^{k} [p_i]_{j_i} = -\eta^{-1} \log(a + b),$$

where $d_i := \exp[\eta p_i] \in \mathbb{R}^n_{\geqslant 0}$,

$$a := \sum_{\substack{\vec{j}\in[n]^k \\ s.t.\ S_{\vec{j}}\neq 0}} \prod_{i=1}^{k} [d_i]_{j_i} \cdot e^{-\eta \tilde{R}_{\vec{j}}} \cdot (e^{-\eta S_{\vec{j}}} - 1) \tag{6.20}$$

and

$$b := \sum_{\vec{j}\in[n]^k} \prod_{i=1}^{k} [d_i]_{j_i} \cdot e^{-\eta \tilde{R}_{\vec{j}}}. \tag{6.21}$$

Proof. By expanding the definition of softmin, and then substituting p_i with d_i and $\tilde{C}$ with $\tilde{R} + S$,

$$\begin{aligned}
\operatorname{smin}_{\eta, \vec{j}\in[n]^k} \tilde{C}_{\vec{j}} - \sum_{i=1}^{k} [p_i]_{j_i} &= -\frac{1}{\eta} \log \left(\sum_{\vec{j}\in[n]^k} e^{\eta \sum_{i=1}^{k} [p_i]_{j_i}} e^{-\eta \tilde{C}_{\vec{j}}} \right) \\
&= -\frac{1}{\eta} \log \left(\sum_{\vec{j}\in[n]^k} \prod_{i=1}^{k} [d_i]_{j_i} \cdot e^{-\eta \tilde{R}_{\vec{j}}} e^{-\eta S_{\vec{j}}} \right).
\end{aligned}$$

By simple manipulations, we conclude that the above quantity is equal to the desired quantity:

$$\begin{aligned}
\cdots &= -\frac{1}{\eta} \log \left(\sum_{\substack{\vec{j}\in[n]^k \\ \text{s.t. } S_{\vec{j}}\neq 0}} \prod_{i=1}^{k} [d_i]_{j_i} \cdot e^{-\eta \tilde{R}_{\vec{j}}} e^{-\eta S_{\vec{j}}} + \sum_{\substack{\vec{j}\in[n]^k \\ \text{s.t. } S_{\vec{j}}=0}} \prod_{i=1}^{k} [d_i]_{j_i} \cdot e^{-\eta \tilde{R}_{\vec{j}}} \right) \\
&= -\frac{1}{\eta} \log \left(\sum_{\substack{\vec{j}\in[n]^k \\ \text{s.t. } S_{\vec{j}}\neq 0}} \prod_{i=1}^{k} [d_i]_{j_i} \cdot e^{-\eta \tilde{R}_{\vec{j}}} \left(e^{-\eta S_{\vec{j}}} - 1\right) + \sum_{\vec{j}\in[n]^k} \prod_{i=1}^{k} [d_i]_{j_i} \cdot e^{-\eta \tilde{R}_{\vec{j}}} \right) \\
&= -\frac{1}{\eta} \log(a + b).
\end{aligned}$$

Above, the first step is by partitioning the sum over $\vec{j} \in [n]^k$ based on if $S_{\vec{j}} = 0$, the second step is by adding and subtracting $\sum_{\vec{j}\in[n]^k \text{ s.t. } S_{\vec{j}}\neq 0} \prod_{i=1}^{k} [d_i]_{j_i} \cdot e^{-\eta \tilde{R}_{\vec{j}}}$, and the last step is by definition of a and b. □

Lemma 6.7.7 (Low-rank approximation of the exponential of a low-rank tensor)**.** *There is an algorithm that given $R \in (\mathbb{R}^n)^{\otimes k}$ in rank-r factored form, $\eta > 0$, and a precision $\tilde{\varepsilon} < e^{-\eta R_{\max}}$, takes $n \cdot \mathrm{poly}(k, \tilde{r})$ time to compute a rank-$\tilde{r}$ tensor $L \in (\mathbb{R}^n)^{\otimes k}$ in factored form satisfying $\|L - \exp[-\eta R]\|_{\max} \leqslant \tilde{\varepsilon}$, where*

$$\tilde{r} \leqslant \binom{r + O(\log \frac{1}{\tilde{\varepsilon}})}{r}. \tag{6.22}$$

Proof. By classical results from approximation theory (see, e.g., [218]), there exists a polynomial q of degree $m = O(\log 1/\tilde{\varepsilon})$ satisfying

$$|\exp(-\eta x) - q(x)| \leqslant \tilde{\varepsilon}, \qquad \forall x \in [-R_{\max}, R_{\max}].$$

For instance, the Taylor or Chebyshev expansion of $x \mapsto \exp(-\eta x)$ suffices. Thus the tensor L with entries

$$L_{\vec{j}} = q(R_{\vec{j}})$$

approximates $\exp[-\eta R]$ to error

$$\|L - \exp[-\eta R]\|_{\max} \leqslant \tilde{\varepsilon}.$$

We now show that L has rank $\tilde{r} \leqslant \binom{r+m}{r}$, and moreover that a rank-$\tilde{r}$ factorization can be computed in $n \cdot \mathrm{poly}(k, \tilde{r})$ time. Denote $q(x) = \sum_{t=0}^{m} a_t x^t$ and $R = \sum_{\ell=1}^{r} \otimes_{i=1}^{k} u_{i,\ell}$. By definition of L, definition of q and R, and then the Multinomial Theorem,

$$L_{\vec{j}} = q(R_{\vec{j}}) = \sum_{t=0}^{m} a_t \left(\sum_{\ell=1}^{r} \prod_{i=1}^{k} [u_{i,\ell}]_{j_i} \right)^t = \sum_{\alpha \in \mathbb{N}_0^r \,:\, |\alpha| \leqslant m} \binom{|\alpha|}{\alpha} a_{|\alpha|} \prod_{\ell=1}^{r} \prod_{i=1}^{k} [u_{i,\ell}]_{j_i}^{\alpha_i},$$

where the sum is over r-tuples α with non-negative entries summing to at most m. Thus

$$L = \sum_{\alpha \in \mathbb{N}_0^r \,:\, |\alpha| \leqslant m} \bigotimes_{i=1}^{k} v_{i,\alpha},$$

where $v_{i,\alpha} \in \mathbb{R}^n$ denotes the vector with j-th entry $\binom{|\alpha|}{\alpha} a_{|\alpha|} \prod_{\ell=1}^{r} [u_{i,\ell}]_j^{\alpha_i}$ for $i = 1$, and $\prod_{\ell=1}^{r} [u_{i,\ell}]_j^{\alpha_i}$ for $i > 1$. This yields the desired low-rank factorization of L because

$$\tilde{r} \leqslant \#\{\alpha \in \mathbb{N}_0^r \,:\, |\alpha| \leqslant m\} = \binom{r+m}{r}.$$

Finally, since each of the $k\tilde{r}$ vectors $v_{i,\alpha}$ in the factorization of L can be computed efficiently from the closed-form expression above, the desired runtime follows. □

Lemma 6.7.8 (Marginalizing a scaled low-rank tensor)**.** *Given vectors $d_1, \dots, d_k \in \mathbb{R}^n$ and a tensor $L \in (\mathbb{R}^n)^{\otimes k}$ through a rank $\tilde{r}$ factorization, we can compute $m((\otimes_{i=1}^k d_i) \odot L)$ in $O(nk\tilde{r})$ time.*

Proof. Denote the factorization of L by $L = \sum_{\ell=1}^{\tilde{r}} \otimes_{i=1}^k v_{i,\ell}$. Then

$$\begin{aligned} m((\otimes_{i=1}^k d_i) \odot L) &= \sum_{\vec{j} \in [n]^k} \left[(\otimes_{i=1}^k d_i) \odot L\right]_{\vec{j}} = \sum_{\vec{j} \in [n]^k} \sum_{\ell=1}^{\tilde{r}} \prod_{i=1}^k [d_i]_{j_i} [v_{i,\ell}]_{j_i} \\ &= \sum_{\ell=1}^{\tilde{r}} \prod_{i=1}^k \sum_{j=1}^n [d_i]_j [v_{i,\ell}]_j = \sum_{\ell=1}^{\tilde{r}} \prod_{i=1}^k \langle d_i, v_{i,\ell} \rangle, \end{aligned}$$

where the first step is by definition of the $m(\cdot)$ operation that sums over all entries, the second step is by definition of L, and the third step is by swapping products and sums. Thus computing the desired quantity amounts to computing $\tilde{r}k$ inner products of n-dimensional vectors. This can be done in $O(nr\tilde{k})$ time. □

Lemma 6.7.9 (Precision of the low-rank approximation)**.** *Let $\varepsilon \leqslant 1$. Suppose $L \in (\mathbb{R}^n)^{\otimes k}$ satisfies $\|L - \exp[-\eta R]\|_{\max} \leqslant \frac{\varepsilon}{3} e^{-\eta R_{\max}}$. Then the matrix $\tilde{C} := -\frac{1}{\eta} \log[L] + S$ satisfies*

$$\|\tilde{C} - C\|_{\max} \leqslant \frac{\varepsilon}{2}. \tag{6.23}$$

Proof. Observe that the minimum entry of L is at least

$$e^{-\eta R_{\max}} - \tfrac{\varepsilon}{3} e^{-\eta R_{\max}} \geqslant \tfrac{2}{3} e^{-\eta R_{\max}}. \tag{6.24}$$

Since this is strictly positive, the tensor $\tilde{R} := -\eta^{-1} \log[L]$ is well defined. Furthermore,

$$\|\eta\tilde{R} - \eta R\|_{\max} = \max_{\vec{j} \in [n]^k} \left|\eta \tilde{R}_{\vec{j}} - \eta R_{\vec{j}}\right| \leqslant \max_{\vec{j} \in [n]^k} \frac{\left|L_{\vec{j}} - e^{-\eta R_{\vec{j}}}\right|}{\min(L_{\vec{j}}, e^{-\eta R_{\vec{j}}})} \leqslant \frac{\frac{\varepsilon}{3} e^{-\eta R_{\max}}}{\frac{2}{3} e^{-\eta R_{\max}}} = \frac{\varepsilon}{2},$$

where above the first step is by definition of the max norm; the second step is by the elementary inequality $|\log x - \log y| \leqslant |x - y| / \min(x, y)$ which holds for positive scalars x and y [10, Lemma K]; and the third step is by (6.24) and the approximation bound of L. Since $\eta \geqslant 1$, we therefore conclude that $\|\tilde{R} - R\|_{\max} \leqslant \varepsilon/2$. By adding and subtracting S, this implies $\|\tilde{C} - C\|_{\max} = \|\tilde{R} - R\|_{\max} \leqslant \varepsilon/2$. □

■ 6.7.2.2 Proof of Theorem 6.7.4

We are now ready to state the algorithm in Theorem 6.7.4. Pseudocode is in Algorithm 6.8. Note that $\tilde{R} = -\eta^{-1}\log[L]$ and $\tilde{C} = \tilde{R} + S$ are never explicitly computed because in both Lines 3 and 4, the algorithm performs the relevant operations only through the low-rank tensor L and the sparse tensor S.

Input: Low-rank tensor R, sparse tensor S, matrix $p \in \bar{\mathbb{R}}^{n\times k}$, accuracy $\varepsilon > 0$
Output: Solution to both $\mathsf{AMIN}_C(p,\varepsilon)$ on cost tensor $C = R + S$, and also $\mathsf{SMIN}_{\tilde{C}}(p, (2k\log n)/\varepsilon)$ on some approximate cost tensor $\tilde{C}$ satisfying $\|C - \tilde{C}\|_{\max} \leqslant \varepsilon/2$

1: $\eta \leftarrow (2k\log n)/\varepsilon$
2: Compute low-rank approximation L of $\exp[-\eta R]$ via Lemma 6.7.7, for precision $\tilde{\varepsilon} = \frac{\varepsilon}{3}e^{-\eta\mathbb{R}_{\max}}$
3: Compute a in (6.20) directly by enumerating over the polynomially many non-zero entries of S, where $\tilde{R} = -\eta^{-1}\log[L]$
4: Compute b in (6.21) via Lemma 6.7.8, where $\tilde{R} = -\eta^{-1}\log[L]$
5: **return** $-\eta^{-1}\log(a+b)$

Algorithm 6.8: Polynomial-time algorithm for AMIN and SMIN for low-rank + sparse costs (Theorem 6.7.4).

Proof of Theorem 6.7.4. Proof of correctness for SMIN. Consider any oracle inputs $p = (p_1, \ldots, p_k) \in \bar{\mathbb{R}}^{n\times k}$. By Lemma 6.7.9, the tensor $\tilde{C} = \tilde{R} + S = -\eta^{-1}\log L + S$ satisfies $\|\tilde{C} - C\|_{\max} \leqslant \varepsilon/2$. Therefore it suffices to show that Algorithm 6.8 correctly computes $\mathsf{SMIN}_{\tilde{C}}(p,\eta)$. This is true because that quantity is equal to $-\eta^{-1}\log(a+b)$ by Lemma 6.7.6.

Proof of correctness for AMIN. We have just established that Algorithm 6.8 computes $\mathsf{SMIN}_{\tilde{C}}(p,\eta)$. Because $\eta = (2k\log n)/\varepsilon$ and the fact that SMIN is a special case of AMIN (Remark 6.3.6), it follows that $\mathsf{SMIN}_{\tilde{C}}(p,\eta)$ is within additive accuracy $\varepsilon/2$ of $\mathsf{MIN}_{\tilde{C}}(p,\eta)$. Therefore, by the triangle inequality, it suffices to show that $\mathsf{MIN}_{\tilde{C}}(p)$ is within $\varepsilon/2$ additive accuracy of $\mathsf{MIN}_C(p)$. That is, it suffices to show that

$$\left| \min_{\vec{j}\in[n]^k} C_{\vec{j}} - \sum_{i=1}^k [p_i]_{j_i} - \min_{\vec{j}\in[n]^k} \tilde{C}_{\vec{j}} - \sum_{i=1}^k [p_i]_{j_i} \right| \leqslant \varepsilon/2.$$

But this is true because $\|C - \tilde{C}\|_{\max} \leqslant \varepsilon/2$ by Lemma 6.7.9, and thus the quantities $C_{\vec{j}} - \sum_{i=1}^k [p_i]_{j_i}$ and $\tilde{C}_{\vec{j}} - \sum_{i=1}^k [p_i]_{j_i}$ are within additive accuracy $\varepsilon/2$ for each $\vec{j} \in [n]^k$.

Proof of runtime. We prove the claimed runtime bound simultaneously for the AMIN and SMIN computation because we use the same algorithm for both. To

this end, we first bound the rank $\tilde{r}$ of the low-rank approximation L computed in Lemma 6.7.7. Note that since $\tilde{\varepsilon} = \frac{\varepsilon}{3}e^{-\eta R_{\max}}$ and since it is assumed that $R_{\max} = O(C_{\max})$, we have $\log 1/\tilde{\varepsilon} = O(\frac{C_{\max}}{\varepsilon} k \log n)$. Therefore

$$\tilde{r} \leqslant \binom{r + O(\log 1/\tilde{\varepsilon})}{r} = O(\log 1/\tilde{\varepsilon})^r = O(\tfrac{C_{\max}}{\varepsilon} k \log n)^r = \mathrm{poly}(\log n, k, C_{\max}/\varepsilon).$$

Above, the first step is by Lemma 6.7.7, and the final step is because r is assumed constant.

Therefore Line 2 in Algorithm 6.8 takes polynomial time by Lemma 6.7.7, Line 3 takes polynomial time by simply enumerating over the s non-zero entries of S, and Line 4 takes polynomial time by Lemma 6.7.8. □

■ 6.7.3 Application vignette: risk estimation

Here we consider an application to portfolio risk management. For simplicity of exposition, let us first describe the setting of 1 financial instrument ("stock"). Consider investing in one unit of a stock for k years. For $i \in \{0, \dots, k\}$, let X_i denote the price of the stock at year i. Suppose that the return $\rho_i = X_i/X_{i-1}$ of the stock between years $i-1$ and i is believed to follow some distribution $\rho_i \sim \mu_i$. A fundamental question about the riskiness of this stock is to compute the investor's expected profit in the worst-case over all joint probability distributions on future returns $(\rho_1, \dots, \rho_k)$ that are consistent with the modeled marginal distributions $(\mu_1, \dots, \mu_k)$. This is an MOT problem with cost C given by

$$C(\rho_1, \dots, \rho_k) = \prod_{i \in [k]} \rho_i,$$

where here we view C as a function rather than a tensor for notational simplicity. If each return ρ_i has n possible values (e.g., after quantization), then the cost C is equivalently represented as a rank-1 tensor in $(\mathbb{R}^n)^{\otimes k}$ (by assigning an index to each of the n possible values of each ρ_i). Therefore our result Corollary 6.7.5 provides a polynomial-time algorithm for solving this MOT problem defining the investor's worst-case profit.

Rather than formalize this proof for 1 stock, we directly generalize to the general case of investing in r stocks, $r \geqslant 1$. This is essentially identical to the simple case of $r = 1$ stock, modulo additional notation.

Corollary 6.7.10 (Polynomial-time algorithm for expected profit given marginals on the returns)**.** *Suppose an investor holds* 1 *unit of r stocks for k years. For each stock $\ell \in [r]$ and each year $i \in [k]$, let $\rho_{i,\ell}$ denote the relative price of stock ℓ between years i and $i-1$. Suppose $\rho_{i,\ell}$ has distribution $\mu_{i,\ell}$, and that each $\mu_{i,\ell}$ has*

at most n atoms. Let $\mathbb{R}_{\max} = \max_{\{\rho_{i,\ell}\}} \sum_{\ell=1}^{r} \prod_{i=1}^{k} \rho_{i,\ell}$ denote the maximal possible return. For any constant number of stocks r, there is a $\mathrm{poly}(n, k, \mathbb{R}_{\max}/\varepsilon)$ time algorithm for ε-approximating the expected profit in the worst-case over all futures that are consistent with the returns' marginal distributions.

Proof. This is the optimization problem

$$\min_{P \in \mathcal{M}(\{\mu_{i,\ell}\}_{i\in[k],\ell\in[r]})} \mathbb{E}_{\{\rho_{i,\ell}\}_{i\in[k],\ell\in[r]}\sim P} \left[\sum_{\ell=1}^{r} \prod_{i=1}^{k} \rho_{i,\ell} \right]$$

over all joint distributions P on the returns $\{\rho_{i,\ell}\}_{i\in[k],\ell\in[k]}$ that are consistent with the marginal distibutions $\{\mu_{i,\ell}\}_{i\in[k],\ell\in[k]}$. This is an MOT problem with $k' = rk$ marginals, each over n atoms, with cost function

$$C\Big(\{\rho_{i,\ell}\}_{i\in[k],\ell\in[r]}\Big) = \sum_{\ell'\in[r]} \prod_{(i,\ell)\in[k]\times[r]\cong[k']} \big(\rho_{i,\ell} \cdot \mathbb{1}[\ell = \ell'] + \mathbb{1}[\ell \neq \ell']\big). \tag{6.25}$$

By viewing this cost function C as a cost tensor in the natural way (i.e., assigning an index to each of the n possible values of $\rho_{i,\ell}$), this representation (6.25) shows that the corresponding cost tensor $C \in (\mathbb{R}^n)^{\otimes k'}$ has rank r. Moreover, observe that the maximum entry of the cost is $\mathbb{R}_{\max}$. Therefore we may appeal to our polynomial-time MOT algorithms in Corollary 6.7.5 for costs with constant rank. □

The algorithm is readily generalized, e.g., if the investor has different units of a stock, or if a stock is held for a different number of years. The former is modeled simply by adding an extra year in which the return of stock ℓ is equal to the number of units, with probability 1. The latter is modeled simply by setting the return of stock ℓ to be 1 for all years after it is held, with probability 1.

In Figure 6.7, we provide a numerical illustration comparing our new polynomial-time algorithms for this risk estimation task with the previous fastest algorithms. Previously, the fastest algorithms that apply to this problem are out-of-the-box LP solvers run on MOT, and the brute-force implementation of `SINKHORN` which marginalizes over all n^k entries in each iteration. Since both of these previous algorithms have exponential runtime that scales as $n^{\Omega(k)}$, they do not scale beyond tiny input sizes of $n = 10$ and $k = 8$ even with two minutes of computation time. In contrast, our new polynomial-time algorithms compute high-quality solutions for problems that are orders-of-magnitude larger. For example, our polynomial-time implementation of `SINKHORN` takes less than a second to solve an MOT LP with $n^k = 10^{30}$ variables.

Details for this numerical experiment: we consider $r = 1$ stock over k timesteps, where each marginal distribution μ_i is uniform on $[1, 1 + 1/k]$, discretized with

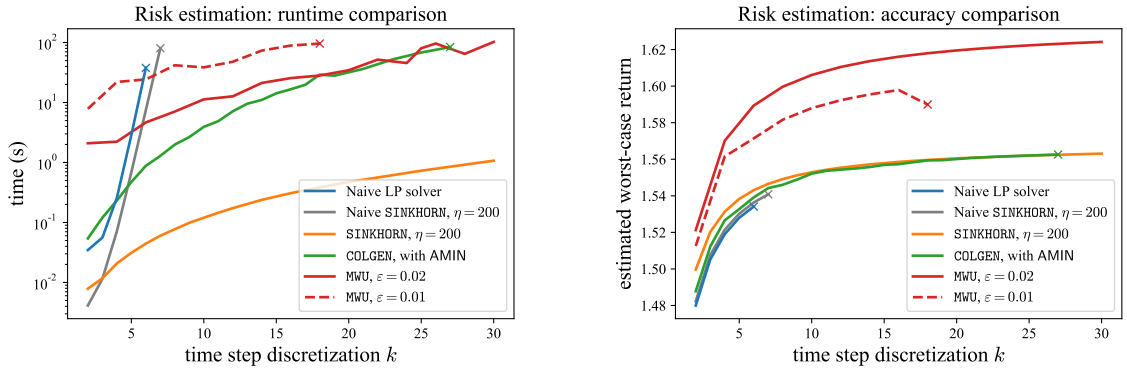


Figure 6.7: Comparison of the runtime (left) and accuracy (right) of the fastest existing algorithms (naive LP solver and naive SINKHORN which both have exponential runtimes that scale as $n^{\Omega(k)}$) with our algorithms (SINKHORN, MWU, and COLGEN and MWU with polynomial-time implementations of their bottlenecks) for the risk estimation problem described in the main text. The algorithms are cut off at 2 minutes, denoted by an "x". Our new polynomial-time implementation of SINKHORN returns high-quality solutions for problems that are orders-of-magnitude larger than previously possible: e.g., it takes less than a second to solve the problem for $k = 30$, which amounts to an MOT LP with 10^{30} variables.

$n = 10$. We implement the AMIN and SMIN oracle efficiently by using our above algorithm to exploit the rank-one structure of the cost tensor. In particular, the polynomial approximation we use here to approximate $\exp[-\eta C]$ is the degree-5 Taylor approximation (cf., Lemma 6.7.7). This lets us run SINKHORN and MWU in polynomial time, as described above. In the numerical experiment, we also implement an approximate version of COLGEN using our polynomial-time implementation of the approximate violation oracle AMIN. Since the algorithms compute an upper bound, lower value is better in the right plot of Figure 6.7. We observe that MWU yields the loosest approximation for this application, whereas our implementations of SINKHORN and COLGEN produce high-quality approximations, as is evident by comparing to the exact LP solver in the regime that the latter is tractable to run.

■ 6.7.4 Application vignette: projection to the transportation polytope

Here we consider the fundamental problem of projecting a joint probability distribution Q onto the transportation polytope $\mathcal{M}(\mu_1, \dots, \mu_k)$, i.e.,

$$\operatorname*{arg\,min}_{P \in \mathcal{M}(\mu_1,\dots,\mu_k)} \sum_{\vec{j}} (P_{\vec{j}} - Q_{\vec{j}})^2. \tag{6.26}$$

We provide the first polynomial-time algorithm for solving this problem in the case where Q is a distribution that decomposes into a low-rank component plus a sparse component. The low-rank component enables modeling mixtures of product distributions (e.g., mixtures of isotropic Gaussians), which arise frequently in statistics and machine learning; see, e.g., [87]. In such applications, the number of product distributions in the mixture corresponds to the tensor rank. The sparse component further enables modeling arbitrary corruptions to the distribution in polynomially many entries.

We emphasize that this projection problem (6.26) is *not* an MOT problem since the objective is quadratic rather than linear. This illustrates the versatility of our algorithmic results. Our algorithm is based on a reduction from quadratic optimization to linear optimization over $\mathcal{M}(\mu_1, \dots, \mu_k)$ that is tailored to this problem. Crucial to this reduction is the fact that the MOT algorithms in §6.4 can compute *sparse* solutions. In particular, this reduction does not work with SINKHORN because SINKHORN cannot compute sparse solutions.

Corollary 6.7.11 (Efficient projection to the transportation polytope)**.** *Let $Q = R + S \in (\mathbb{R}^n_{\geqslant 0})^{\otimes k}$, where R has constant rank and S is polynomially sparse. Suppose that $R_{\max}$ and $S_{\max}$ are $O(1)$. Given R in factored form, S through its non-zero entries, measures $\mu_1, \dots, \mu_k \in \Delta_n$, and accuracy $\varepsilon > 0$, we can compute in $\operatorname{poly}(n, k, 1/\varepsilon)$ time a feasible $P \in \mathcal{M}(\mu_1, \dots, \mu_k)$ that has ε-suboptimal cost for the projection problem* (6.26)*. This solution P is a sparse tensor output through its $\operatorname{poly}(n, k, 1/\varepsilon)$ non-zero entries.*

Proof. We apply the Frank-Wolfe algorithm (a.k.a., Conditional Gradient Descent) to solve (6.26), specifically using approximate LP solutions for the descent direction as in [120, Algorithm 2]. By the known convergence guarantee of this algorithm [120, Theorem 1.1], if each LP is solved to $\varepsilon' = O(\varepsilon)$ accuracy, then $T = O(1/\varepsilon)$ Frank-Wolfe iterations suffice to obtain an ε-suboptimal solution to (6.26).

The crux, therefore, is to show that each Frank-Wolfe iteration can be computed efficiently, and that the final solution is sparse. Initialize $P^{(0)}$ to be an arbitrary vertex of $\mathcal{M}(\mu_1, \dots, \mu_k)$. Then $P^{(0)}$ is feasible and is polynomially sparse

(see §6.2.1). Let $P^{(t)} \in (\mathbb{R}^n_{\geqslant 0})^{\otimes k}$ denote the t-th Frank-Wolfe iterate. Performing the next iteration requires two computations:

1. Approximately solve the following LP to ε' accuracy:

$$D^{(t)} \leftarrow \min_{P \in \mathcal{M}(\mu_1,\ldots,\mu_k)} \langle P, P^{(t)} - Q \rangle. \tag{6.27}$$

2. Update $P^{(t+1)} \leftarrow (1-\gamma_t)P^{(t)} + \gamma_t D^{(t)}$, where $\gamma_t = 2/(t+2)$ is the current stepsize.

For the first iteration $t = 0$, note that the LP (6.27) is an MOT problem with cost $C^{(0)} = P^{(0)} - Q = P^{(0)} - R - S$ which decomposes into a polynomially sparse tensor $P^{(0)} - S$ plus a constant-rank tensor $-R$. Therefore the algorithm in Corollary 6.7.5 can solve the LP (6.27) to $\varepsilon' = O(\varepsilon)$ additive accuracy in $\mathrm{poly}(n, k, 1/\varepsilon)$ time, and it outputs a solution $D^{(0)}$ that is $\mathrm{poly}(n, k, 1/\varepsilon)$ sparse. It follows that $P^{(1)}$ can be computed in $\mathrm{poly}(n, k, 1/\varepsilon)$ time and moreover is $\mathrm{poly}(n, k, 1/\varepsilon)$ sparse since it is a convex combination of the similarly sparse tensors $P^{(0)}$ and $D^{(0)}$. By repeating this argument identically for $T = O(1/\varepsilon)$ iterations, it follows that each iteration takes $\mathrm{poly}(n, k, 1/\varepsilon)$ time, and that each iterate $P^{(t)}$ is $\mathrm{poly}(n, k, 1/\varepsilon)$ sparse. □

■ 6.8 Discussion

In this chapter, we investigated what structure enables MOT—an LP with n^k variables—to be solved in $\mathrm{poly}(n, k)$ time. We developed a unified algorithmic framework for MOT by characterizing what "structure" is required to solve MOT in polynomial time by different algorithms in terms of simple variants of the dual feasibility oracle. On one hand, this enabled us to show that `ELLIPSOID` and `MWU` solve MOT in polynomial time whenever any algorithm can, whereas `SINKHORN` requires strictly more structure. And on the other hand, this made the design of polynomial-time algorithms for MOT much simpler, as we illustrated on three general classes of MOT cost structures.

Our results suggest several natural directions for future research. One exciting direction is to identify further tractable classes of MOT cost structures beyond the three studied in this chapter, since this may enable new applications of MOT. Our results help guide this search because they make it significantly easier to identify if an MOT problem is polynomial-time solvable (see §6.1.1.4).

Another important direction is practicality. While the focus of this chapter is to characterize when MOT problems can be solved in polynomial time, in

practice there is of course a difference between small and large polynomial runtimes. It is therefore a question of practical significance to improve our "proof of concept" polynomial-time algorithms by designing algorithms with smaller polynomial runtimes. Our theoretical results help guide this search for practical algorithms because they make it significantly easier to identify if an MOT problem is polynomial-time solvable in the first place.

In order to develop more practical algorithms, recall that, roughly speaking, our approach for designing MOT algorithms consisted of three parts:

- An "outer loop" algorithm such as `ELLIPSOID`, `MWU`, or `SINKHORN` that solves MOT in polynomial time conditionally on a polynomial-time implementation of a certain bottleneck oracle.
- An "intermediate" algorithm that reduces this bottleneck oracle to polynomial calls of a variant of the dual feasibility oracle.
- An "inner loop" algorithm that solves the relevant variant of the dual feasibility oracle for the structured MOT problem at hand.

Obtaining a smaller polynomial runtime for any of these three parts immediately implies smaller polynomial runtimes for the overall MOT algorithm. Another approach is to design altogether different algorithms that avoid the polynomial blow-up of the runtime that arises from composing these three parts. Understanding how to solve an MOT problem more "directly" in this way is an interesting question.

Chapter 7

Case study: resolving the computational complexity of Wasserstein barycenters

This chapter highlights the application of our general MOT framework to one particularly popular instance of MOT: the computation of Wasserstein barycenters (a.k.a., Optimal Transport barycenters). Wasserstein barycenters provide a geometrically-meaningful notion of average between probability distributions, and over the past decade have emerged as a central tool in data science for manipulating and interpreting complicated geometric data. However, despite considerable attention, the most fundamental question about the complexity of Wasserstein barycenters remained open: can they be computed in polynomial time?

In this chapter, we resolve this problem. Specifically, we establish that Wasserstein barycenters are NP-hard to compute in general, but can be computed in polynomial time in any fixed dimension (e.g., as in physical, imaging, or graphics applications). This uncovers a "curse of dimensionality" for Wasserstein barycenter computation which does not occur for Optimal Transport computation.

7.1 Introduction

Wasserstein barycenters provide a natural approach for averaging probability distributions in a way that respects their geometry. In words, Wasserstein barycenters are the Riemannian centers of mass (a.k.a. Fréchet means) with respect to the Optimal Transport distance [3]. More precisely, given probability distributions $\mu_1, \ldots, \mu_k$ over $\mathbb{R}^d$ and non-negative weights $\lambda_1, \ldots, \lambda_k$ summing to 1, the corresponding Wasserstein barycenters are the probability distributions ν over $\mathbb{R}^d$ that

minimize

$$\min_{\nu} \sum_{i=1}^{k} \lambda_i \mathcal{W}^2(\mu_i, \nu). \tag{7.1}$$

Here, $\mathcal{W}$ denotes the 2-Wasserstein distance (a.k.a. the standard Optimal Transport distance) between probability distributions [224], which we recall is defined as

$$\mathcal{W}(\mu, \nu) = \left(\inf_{\pi \in \mathcal{M}(\mu,\nu)} \mathbb{E}_{(X,Y)\sim\pi} \|X - Y\|_2^2 \right)^{1/2},$$

where $\mathcal{M}(\mu, \nu)$ is the set of joint distributions with first marginal μ and second marginal ν.

Wasserstein barycenters have received considerable research attention over the past decade due to their elegant mathematical properties (see, e.g., [3]) and many data-science applications (see, e.g., the surveys [172, 178]). For example, illustrative applications include improving Bayesian learning by averaging posterior distributions [208], improving sensors by averaging their measurements [86], interpolating between shapes by averaging them (viewed as point clouds in Euclidean space) [206], clustering documents (viewed as distributions over word embeddings) [234, 235], multilevel clustering of datasets [114, 115], and unsupervised representation learning in natural language processing [202]. Note that some of these applications are in low-dimensional settings (e.g., graphics, imaging, and physical applications), while others are in high-dimensional settings (e.g., natural language processing, machine learning, and statistics applications).

Open problem: computing barycenters in polynomial time. A key issue that determines how useful Wasserstein barycenters are in applications is whether they can be computed efficiently. Note that in most computational applications, each measure μ_i is a discrete distribution: it is a "point cloud" over data points. This motivates the following fundamental question, which has remained open despite considerable research attention (see the previous work section).

Are Wasserstein barycenters of discrete distributions computable in polynomial time?

That is, can the optimization problem (7.1) be solved in time that is polynomial in the number of distributions k, the dimension d, the maximum support size n of the input distributions μ_i, and the bit complexity $\log U$ of each entry in the input measures and weights? This constitutes a running time that is polynomial in the input size since each discrete measure is naturally described as a list of at most n point locations and the corresponding probability masses.

A highly related open problem is whether Wasserstein barycenters can be computed to high accuracy, i.e., whether an ε-additively approximate solution for (7.1) can be computed in time that is polynomial in the input size and $\log(1/\varepsilon)$.

All previous algorithms that compute arbitrarily close approximations have runtimes either 1) of the form $\text{poly}(n^{\Omega(k)}, d)$ which is exponential in the number of marginals k, or 2) of the form $\text{poly}(n, k, 1/\varepsilon^d)$ which is not only exponential in the dimension d, but moreover even for constant dimension like $d = 3$, only enables solving to a few digits of precision due to the $1/\varepsilon^d$ runtime dependence. See the previous work section for details.

The current state of affairs leaves open pressing fundamental questions about the computation of Wasserstien barycenters. Are polynomial runtimes possible in high dimension? In low dimension, are high-precision solutions possible? Is the lack of such a results a failure of previous algorithmic techniques or a fundamental impossibility?

The purpose of this chapter is to resolve all of these questions. Specifically, we establish that in general Wasserstein barycenters are NP-hard to compute, even approximately; however, they can be computed exactly in polynomial time in any fixed dimension (e.g., as in physical, imaging, or graphics applications). This uncovers a "curse of dimensionality" for Wasserstein barycenter computation which does not occur for Optimal Transport computation.

We elaborate on both results below.

7.1.1 Contribution 1: Wasserstein barycenters are NP-hard to compute, even approximately

The first main result of this chapter establishes that in general, it is NP-hard to compute Wasserstein barycenters. This explains why—despite a rapidly growing literature (see the previous work section)—there has been a lack of progress towards developing algorithms that provably compute optimal Wasserstein barycenters in polynomial time.

Theorem 7.1.1 (NP-hardness)**.** *Assuming* $\mathsf{P} \neq \mathsf{NP}$*, there is no algorithm that, given distributions* $\mu_1, \ldots, \mu_k$ *and uniform weights* $\lambda_1, \ldots, \lambda_k = 1/k$*, computes the value of the Wasserstein barycenter problem* (7.1) *in* $\text{poly}(n, k, d, \log U)$ *time.*

Moreover, this hardness extends even to computation of *approximate* Wasserstein barycenters. This extension requires the slightly stronger yet standard complexity-theoretic assumption $\mathsf{NP} \not\subset \mathsf{BPP}$, which in words is the statement that NP-hard problems do not admit polynomial-time randomized algorithms; see §5.2 for a formal definition. This result is formally stated as follows. Below, let R be an upper bound on the squared diameter of the supports of the measures μ_i; any

running time must depend on R and the accuracy ε through the scale-invariant ratio R/ε.

Theorem 7.1.2 (Inapproximability)**.** *Assuming* NP $\not\subset$ BPP*, there is no randomized algorithm that, given distributions $\mu_1, \ldots, \mu_k$ and uniform weights $\lambda_1, \ldots, \lambda_k = 1/k$, computes the value of the Wasserstein barycenter problem* (7.1) *to ε additive accuracy with probability at least* $2/3$ *in* $\mathrm{poly}(n, k, d, \log U, R/\varepsilon)$ *time.*

We make three remarks about these results. First, since Theorems 7.1.1 and 7.1.2 establish hardness for (approximately) computing the optimal *value* of the barycenter problem, as an immediate corollary they preclude finding an (approximately) optimal *solution*. Specifically, since $\mathcal{W}(\nu, \mu_i)$ is computable in polynomial time (e.g., via linear programming [196]) whenever ν has polynomial-size support, these results imply that: unless P = NP, there is no polynomial-time algorithm for computing a barycenter ν with polynomial-size support.[1]

Second, these hardness results hold even in seemingly simple settings. For example, our results are written for the case where all weights $\lambda_1 = \cdots = \lambda_k = 1/k$ are uniform. Our construction also sets all measures $\mu_1, \ldots, \mu_k$ to be supported on n points each with all support points having $\{0, 1\}$-valued coordinates, and can be readily extended to the case where $\mu_1, \ldots, \mu_k$ are uniform distributions; for details see the paper [14] upon which this the chapter is partially based.

Third, these hardness results capture a robust phenomenon as they extend to other important notions of averaging in Wasserstein space that are studied in the literature, see, e.g., [55, 74, 99, 143]. Specifically, for any fixed $p \in [1, \infty)$ and $q \in [1, \infty]$, the "generalized Wasserstein barycenter problem" is similarly computationally hard, where one replaces the 2-Wasserstein distance with the p-Wasserstein distance, and also replaces the ℓ_2 ground metric by any ℓ_q ground metric. For details on this generalized problem, its robustness properties, its applications, and the proof of this extension, we refer the reader to the paper [14] upon which this the chapter is partially based.

■ 7.1.2 Contribution 2: Polynomial-time algorithm in fixed dimension

In sharp contrast to the above intractability result that Wasserstein barycenters are NP-hard in general, our second result shows that in any fixed dimension d, they are polynomial-time computable. Specifically, we give the first algorithm that, in any fixed dimension d, solves the Wasserstein barycenter problem exactly or to high precision in polynomial time. The formal statement of these results

[1] Note that there always *exists* a barycenter with support of size $O(nk)$ [22]. Our result shows that if P $\neq$ NP, then such a barycenter cannot be efficiently computed.

is as follows. For simplicity of notation, throughout d is a constant; the running time for fixed d is $(nk)^d$ times a polynomial in the input size.

Theorem 7.1.3 (Computing high-precision barycenters in fixed dimension)**.** *There is an algorithm that, given k distributions each supported on n atoms in the ball of squared radius R in $\mathbb{R}^d$, a weight vector λ, and an accuracy $\varepsilon > 0$, computes an ε-additively approximate Wasserstein barycenter in $\mathrm{poly}(n, k, \log(R/\varepsilon))$ time. Moreover, this barycenter has support size at most $nk - k + 1$.*

Theorem 7.1.4 (Computing exact barycenters in fixed dimension)**.** *If the weight vector and distributions are represented with $\log U$ bits of precision, then an exact barycenter can be found in $\mathrm{poly}(n, k, \log U)$ time. Moreover, this barycenter has support size at most $nk - k + 1$.*

In contrast, the fastest previous algorithms in this setting have runtimes of the form $\mathrm{poly}(n, k, 1/\varepsilon^d)$, which even in fixed dimension d, depend polynomially on $1/\varepsilon$. This means that in practice they can only solve to a few digits of precision. See the prior work section for details. In many applications, including nearly all of those mentioned above, Wasserstein barycenters are used as a subroutine in a larger pipeline to solve downstream data science tasks. Thus, high-precision algorithms are important for downstream performance and to avoid error propagation, especially in applications which require multiple barycenter computations. Although the focus of this chapter is theoretical, we also provide preliminary numerical experiments in §7.4.3 demonstrating that a slight variant of our algorithm can provide high-precision solutions at previously intractable problem sizes.

In addition to its polynomial runtime, our algorithm has two additional properties that may be useful in downstream applications. First, the outputted barycenter ν has small support of $O(nk)$ size, which is much smaller than the *a priori* n^k bound on the support size. In particular, the support size of ν is at most the maximal sparsity of any vertex of the transportation polytope between $\mu_1, \ldots, \mu_k$—which is at most $nk - k + 1$. Note that Theorem 7.1.4 is not at odds with the NP-hardness of finding the sparsest barycenter [46]: indeed, our algorithm outputs a solution that albeit sparse is not necessarily the sparsest. Second, as a by-product, our algorithm also produces sparse solutions to the Optimal Transport problems $\mathcal{W}(\mu_i, \nu)$ that are non-mass-splitting maps from ν to μ_i. Among other benefits, this enables easy visualization and interpretability of the results—in comparison to entropic-regularization based approaches which produce "blurry" dense maps.

We also mention that the techniques we develop here extend to solving several related problems involving generalized Wasserstein barycenters. In particular, this gives the first polynomial-time algorithms for computing geometric medians with

respect to the 1-Wasserstein distance (a.k.a. Earth Mover's distance) over any of the popular ground metrics ℓ_1, ℓ_2, or ℓ_∞. For details, see the paper [11] upon which this chapter is partially based.

7.1.3 Related work

The many applications of Wasserstein barycenters have motivated an extensive literature that approaches this problem from both the algorithmic and hardness sides. Here we contextualize our results with the literature.

7.1.3.1 Algorithms for the Wasserstein barycenter problem

Many algorithms have been proposed. However, all of them have running time which scales exponentially in at least one of the input parameters, and/or do not provably compute arbitrarily close approximations, described below.

Algorithms with exponential dependence in d. A popular approach is to use "fixed-support approximations"; that is, assume that the barycenter is supported on a guessed set $S \subset \mathbb{R}^d$ of points, and then optimize over the corresponding weights, see, e.g., [32, 55, 74, 121, 136, 143, 206, 209] among many others. The point of this fixed-support approximation is that it reduces the barycenter problem to a polynomial-size LP—which can then be solved efficiently using out-of-the-box LP solvers or specially-tailored approaches such as entropic regularization—if the set S has polynomial size. However, this "if" is the key issue: obtaining a barycenter that is ε-additively approximate for the objective (7.1) requires taking S to be an ε-cover of the space. In particular, this means that all fixed-support methods require $\Omega((R/\varepsilon)^d)$ time. Such running times have two issues. First is the exponential scaling in the dimension d. Second is that they only compute to "low precision" ε due to the $1/\varepsilon$ dependence. While not fixed-support approaches, the Frank-Wolfe algorithm of [147] and the Functional Gradient Descent algorithm of [199] also suffer from the same two issues.

In contrast, our proposed algorithm in Theorem 7.1.3 has $\mathrm{poly}(n, k, \log(R/\varepsilon))$ runtime in fixed dimension—and critically, this has polylogarithmic dependence on R/ε.[2] In practice, this means that our algorithm can often solve up to machine precision, whereas fixed-support algorithms can only solve up to a few digits of precision—see §7.4.3 for numerical experiments.

Algorithms with exponential dependence in k. A well-known approach that avoids exponential dependence on the dimension d is to reformulate the Wasserstein

[2]This means that our algorithm solves the barycenter problem exactly in polynomial time, whereas previous algorithms require *pseudo-polynomial* time. This is because solving the barycenter problem exactly requires ε to be exponentially small in the bit-complexity of the input.

barycenter as a linear program (LP) and then solve it. However, this LP has n^k variables (see, e.g., [22, 32]), so applying a standard LP solver out-of-the-box requires $\Omega(n^k)$ time which is exponential in k.

2-approximation. [45] proposes the following algorithm: fix the support of ν to be the union of the supports of the input measures μ_i, and optimize the corresponding nk weights via an LP solver. [45] shows that this yields a multiplicative 2-approximation to the optimal barycenter problem (7.1) in $\mathrm{poly}(n, k, d, \log U)$ time, and that this approximation factor is tight (i.e., there exist inputs for which this algorithm yields objective exactly twice the optimal). This is the polynomial-time algorithm with the best provable approximation guarantees we are aware of for the barycenter problem in high dimensions. In fact, our Theorem 7.1.2 implies that this polynomial-time algorithm is nearly optimal in the sense that this multiplicative 2-approximation factor is unimprovable to a $(1+\varepsilon)$-approximation under standard complexity theory assumptions.

■ 7.1.3.2 Hardness of the sparsest Wasserstein barycenter

Perhaps the most related NP-hardness result is that finding the sparsest[3] Wasserstein barycenter is NP-hard, even in the setting of $k = 3$ uniform measures in dimension $d = 2$ [46]. The key difference from the hardness results in this chapter is that the results of [46] apply to the problem of finding the *sparsest* barycenter, and do not imply NP-hardness of finding a barycenter with sparsity that is polynomial in the input size, which is typically the goal in applications. For example, for the setting of $k = 3$ measures, while the result of [46] shows NP-hardness of finding a barycenter with sparsity n, a barycenter with sparsity $O(n)$ can be found in $\mathrm{poly}(n, \log U)$ time by using off-the-shelf LP solvers on the MOT formulation of the Wasserstein barycenter problem [22, 32]. Similarly, for any fixed dimension $d \geqslant 2$, while the result of [46] shows NP-hardness of finding a barycenter with sparsity n, a barycenter with sparsity $O(nk)$ can be found in $\mathrm{poly}(n, k, \log U)$ time for arbitrary k by our result in Theorem 7.1.4.

■ 7.1.3.3 Other related work

Algorithms based on entropic regularization. The influential paper [73] popularized the use of entropic regularization for large-scale Optimal Transport computa-

[3] In [46, Theorem 3], the NP-hardness is stated for the problem of finding a Wasserstein barycenter with sparsity at most some input integer N. This is polynomial-time equivalent to the problem of finding the barycenter with smallest sparsity. Indeed an answer to the latter problem is an answer to the former, and an algorithm for the former problem gives an answer to the latter by running the algorithm on all $N \leqslant nk - k + 1$ (since there always exists a barycenter with sparsity $nk - k + 1$ [22]).

tion, see Chapter 2. The use of entropic regularization to compute Wasserstein barycenters was first proposed in [74], which inspired a long line of work, see, e.g., [32, 121, 136, 143, 147, 199, 206]. Intuitively, the idea is to regularize the resulting LP by adding δ times an entropy cost, for δ small. This makes the LP strongly convex and easier to optimize. Previous work has sought to design barycenter algorithms by judiciously choosing δ and designing specialized algorithms for the resulting δ-regularized barycenter problem. An immediate corollary of Theorem 7.1.2 is that entropic regularization does not help for computing barycenters in high dimensions: under standard complexity assumptions, there is no efficient algorithm for the Wasserstein barycenter problem regardless of whether one uses entropic regularization.

Continuous distributions. While this chapter and much of the literature focuses on computing Wasserstein barycenters of discrete distributions, there is also a growing line of work on computing barycenters of continous distributions. This continuous setting has several additional computational challenges, such as how to even represent μ_i and ν concisely, and how to compute the Wasserstein distance between them efficiently. Due to these computational issues, the literature on barycenters of continuous distributions typically restricts to Gaussians, in which case specialized algorithms can be designed; see, e.g., [21, 64]. In stark contrast to the discrete setting studied in this chapter, there is no curse of dimensionality for Wasserstein barycenter computation in the setting of Gaussians [20].

7.1.4 Organization

This chapter is organized as follows. §7.2 recalls relevant preliminaries. In §7.3 we prove the hardness results for general Wasserstein barycenters (Theorems 7.1.1 and 7.1.2). In §7.4 we prove the algorithmic results for low-dimensional settings (Theorems 7.1.3 and 7.1.4). In §7.5 we conclude with a discussion of future research directions that are motivated by our results.

This chapter is based on the papers [11, 14], although we omit several results for the sake of brevity. Notable omissions include extensions of the presented results to generalized Wasserstein barycenters (i.e., for p-Wasserstein distance and ℓ_q ground metric, for general $p, q \neq 2$) and details on how the Ellipsoid algorithm can be applied to MOT.

7.2 Preliminaries

The section is organized as follows. In §7.2.1, we recall the well-known reformulation of the Wasserstein barycenter problem as an MOT problem. Then, for the

convenience of the reader, in §7.2.2 we recall the background about MOT from Chapters 5 and 6 that is relevant for the development in this chapter. In §7.2.3 and §7.2.4, we recall relevant preliminaries from computational geometry and computational complexity, respectively.

Notation. The atoms in the support of distribution μ_i are denoted by $x_{i,1}, \ldots, x_{i,n} \in \mathbb{R}^d$. We abuse notation slightly by writing μ_i to denote this discrete distribution as well as the vector of probability masses in the simplex $\Delta_n = \{p \in \mathbb{R}^n_{\geqslant 0} : \sum_{i=1}^n p_i = 1\}$ over the n atoms $\{x_{i,j}\}_{j=1}^n$ in any fixed ordering. The set $\{1, \ldots, n\}$ is denoted by $[n]$. The k-fold tensor product space $\mathbb{R}^n \otimes \cdots \otimes \mathbb{R}^n$ is denoted by $(\mathbb{R}^n)^{\otimes k}$, and similarly for $(\mathbb{R}^n_{\geqslant 0})^{\otimes k}$. For shorthand, we often denote a tuple $(j_1, \ldots, j_k) \in [n]^k$ by $\vec{j}$. The i-th marginal, $i \in [k]$, of a tensor $P \in (\mathbb{R}^n)^{\otimes k}$ is denoted by the vector $m_i(P) \in \mathbb{R}^n$, and has entries $[m_i(P)]_\ell := \sum_{\vec{j} \in [n]^k : j_i = \ell} P_{\vec{j}}$. The transportation polytope between $\mu_1, \ldots, \mu_k$ is the set of joint distributions with one-dimensional marginal distributions $\mu_1, \ldots, \mu_k$, and is identified with the set $\mathcal{M}(\mu_1, \ldots, \mu_k) := \{P \in (\mathbb{R}^n_{\geqslant 0})^{\otimes k} : m_i(P) = \mu_i, \forall i \in [k]\}$.

■ 7.2.1 MOT Formulation of Wasserstein barycenters

For both our intractability and algorithmic results, we make use of the well-known fact [3, 22, 32, 53] that the Wasserstein barycenter problem has an equivalent formulation as a certain exponential-size linear program, namely the Multimarginal Optimal Transport (MOT) problem

$$\min_{P \in \mathcal{M}(\mu_1, \ldots, \mu_k)} \langle P, C \rangle \tag{MOT}$$

for the specific cost tensor $C \in (\mathbb{R}^n)^{\otimes k}$ that has entries

$$C_{\vec{j}} = \min_{y \in \mathbb{R}^d} \sum_{i=1}^k \lambda_i \|x_{i,j_i} - y\|^2, \tag{7.2}$$

or equivalently, $C_{\vec{j}} = \sum_{i=1}^k \lambda_i \|x_{i,j_i} - \sum_{\ell=1}^k \lambda_\ell x_{\ell,j_\ell}\|^2$ by optimality of $y = \sum_{\ell=1}^k \lambda_\ell x_{\ell,j_\ell}$.

This equivalence is formally stated as follows.

Proposition 7.2.1 (MOT formulation)**.** *The value of the Wasserstein barycenter problem* (7.1) *for measures $\mu_1, \ldots, \mu_k$ is equal to the value of the MOT problem with marginals $\mu_1, \ldots, \mu_k$ and cost tensor $C \in (\mathbb{R}^n)^{\otimes k}$ given by* (7.2)*.*

In fact, more is true about this connection: not only are the optimal values of these two problems equal (Proposition 7.2.1), but also their solutions are in "correspondence" in the sense that, given a solution to either one of these

two problems, one can construct a solution to the other problem with the same objective value. While our intractability results only require the above connection between optimal values, our algorithmic results require the following direction of this correspondence between optimal solutions.

Lemma 7.2.2. *If $P \in \mathcal{M}(\mu_1, \ldots, \mu_k)$ is an optimal solution to* (MOT)*, then the pushforward of P under the map $(X_1, \ldots, X_k) \mapsto \sum_{i=1}^k \lambda_i X_i$ is an optimal barycenter ν. Furthermore, the support size of ν is at most the support size of P, and also the coupling $(\sum_{i=1}^k \lambda_i X_i, X_j)$ is a non-mass-splitting map that solves the Optimal Transport problem from ν to μ_j.*

Notice that applying this pushforward map $(X_1, \ldots, X_k) \mapsto \sum_{i=1}^k \lambda_i X_i$ in order to compute ν from P requires only $O(skd)$ arithmetic operations, where s denotes the support size of P. In particular, this takes polynomial time if s is of polynomial size. Therefore in order to efficiently compute a sparse Wasserstein barycenter, it suffices to efficiently compute a sparse solution P of (MOT). To this end, note that the solution P is guaranteed to be sparse if it is a vertex solution. Indeed, since (MOT) is a standard-form LP whose constraints have rank at most $nk - k + 1$, each vertex solution has at most $nk - k + 1$ non-zero entries.

Lemma 7.2.3. *If P is a vertex of the transportation polytope $\mathcal{M}(\mu_1, \ldots, \mu_k)$, then P has at most $nk - k + 1$ non-zero entries.*

■ 7.2.2 Separation oracle for MOT formulation of Wasserstein barycenters

An obvious obstacle for computing any solution—let alone a sparse solution—of the MOT formulation (MOT) is that it has n^k exponentially many variables. See Chapters 5 and 6 for in-depth discussions of when MOT problems can be solved in $\mathrm{poly}(n, k)$ time and related work to this end.

For the convenience of the reader, we recall here the preliminaries about MOT that are necessary for this chapter. First of all, whether an MOT problem can be solved in $\mathrm{poly}(n, k)$ time is intimately related to the "structure" of its cost tensor C. However, the cost tensor (7.2) which arises in the Wasserstein barycenter problem does not fall under any of the classes of structured cost tensors studied in Chapters 5 and 6 (or any other previous work for that matter), and thus the computational complexity of this problem does not follow from previous work. What the known results about MOT do show is that the (approximate) computation of an optimal value/solution is polynomial-time equivalent to the (approximate) computation of the "MIN" oracle studied in Chapters 5 and 6.

For the particular cost tensor (7.2) that arises in the Wasserstein barycenter problem, the $\mathsf{MIN}_C(p)$ problem is: given points $\{x_{i,j}\}_{i\in[k],j\in[n]} \subset \mathbb{R}^d$, scalings

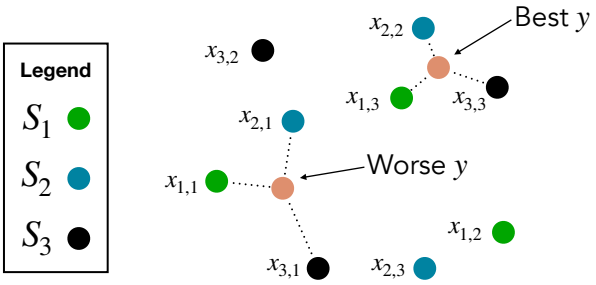


Figure 7.1: Illustration of the CHEAPEST-HUB problem in the case of $k = 3$ sets, each with $n = 3$ points. The points in set S_i are denoted by $\{x_{i,1}, x_{i,2}, x_{i,3}\}$ and displayed in the same color. The CHEAPEST-HUB problem is to choose one point per set to minimize the average distance to the closest "hub" $y \in \mathbb{R}^d$. Top: the points $\{x_{1,3}, x_{2,2}, x_{3,3}\}$ corresponding to tuple $(j_1, j_2, j_3) = (3, 2, 3)$ yield the best hub y. Bottom: the points $\{x_{1,1}, x_{2,1}, x_{3,1}\}$ corresponding to tuple $(j_1, j_2, j_3) = (1, 1, 1)$ yield a suboptimal hub.

$\lambda \in \Delta_n$, and weights $p = (p_1, \ldots, p_k) \in \mathbb{R}^{n \times k}$, compute

$$\min_{\vec{j} \in [n]^k} \left(\min_{y \in \mathbb{R}^d} \lambda_i \sum_{i=1}^{k} \|x_{i,j_i} - y\|^2 \right) - \sum_{i=1}^{k} [p_i]_{j_i}.$$

For our hardness results, we show that in high dimension it is hard to compute MIN_C even in the special case of zero weights $p = 0$ and uniform scalings $\lambda_i = 1/k$. In this particular case, we call the $\mathsf{MIN}_C(p)$ problem CHEAPEST-HUB because of its geometric interpretation: given k sets $S_i \subset \mathbb{R}^d$, each consisting of n points $S_i = \{x_{i,j}\}_{j \in [n]}$, find one point from each set so as to minimize the average distance to the closest "hub" $y \in \mathbb{R}^d$. See Figure 7.1 for an illustration.

Definition 7.2.4 (CHEAPEST-HUB)**.** *Given points $\{x_{i,j}\}_{i \in [k], j \in [n]} \subset \mathbb{R}^d$ as input, the CHEAPEST-HUB problem is to compute*

$$\min_{\vec{j} \in [n]^k} \min_{y \in \mathbb{R}^d} \sum_{i=1}^{k} \|x_{i,j_i} - y\|^2.$$

7.2.3 Relevant preliminaries from computational geometry

A key ingredient in our barycenter algorithm is power diagrams. Here we introduce these objects and some basic facts about their complexity. Although d is a fixed constant in our final algorithmic result, we state the explicit dependence on the

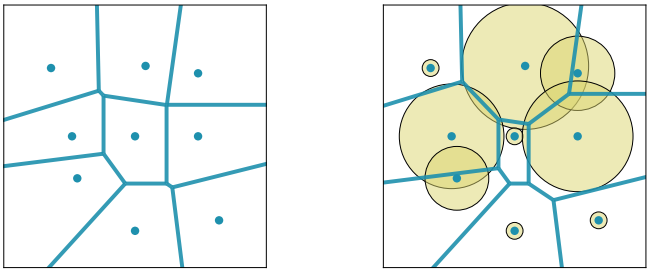


Figure 7.2: Two power diagrams on the same $n = 9$ points with varying weights w. Left: zero weights (this is a Voronoi diagram). Right: the weight of a point is indicated by the size of the ball around it. Increasing a point's weight increases the size of its cell.

dimension d in these power diagram complexity bounds to highlight how and where our algorithm incurs exponential runtime dependence in d.

Definition 7.2.5. *The* power diagram *on the spheres $S(z_1, r_1), \ldots, S(z_n, r_n)$ with centers $z_j \in \mathbb{R}^d$ and radii $r_j \geqslant 0$ is the cell complex whose cells $E_1, \ldots, E_n$ are given by*

$$E_j = \{y \in \mathbb{R}^d \,:\, \|z_j - y\|^2 - r_j^2 < \|z_{j'} - y\|^2 - r_{j'}^2,\ \forall j' \neq j\}.$$

A power diagram essentially partitions $\mathbb{R}^d$ in the sense that its cells are disjoint and their closures cover $\mathbb{R}^d$. See Figure 7.2 for an illustration. Following are two relevant classical facts. The first essentially shows that a power diagram is defined by a small hyperplane arrangement which can moreover be computed efficiently.

Lemma 7.2.6. (Theorems 1 and 7 of [25], using convex hull algorithm of [60]) *A power diagram on n spheres in $\mathbb{R}^d$ has $O(n)$ affine facets of dimension $d-1$. Moreover these facets can be computed in $O((n \log n + n^{\lceil d/2 \rceil}) \cdot \operatorname{polylog} U)$ time, where $\log U$ is the number of bits of precision.*

The second is about hyperplane arrangements. In the sequel this lets us bound the complexity of the "intersection" of multiple power diagrams (defined in §7.4.1).

Lemma 7.2.7 (Theorem 3.3 of [84])**.** *The cell complex formed by an arrangement of N hyperplanes in $\mathbb{R}^d$, represented up to $\log U$ bits of precision, can be computed in $N^d \cdot \operatorname{polylog}(N, U)$ time.*

■ 7.2.4 Relevant preliminaries from computational complexity

For a graph $G = (V, E)$ and vertices $v_1, \ldots, v_k \in V$, denote the number of edges in the induced subgraph of G with these vertices by $|E(v_1, \ldots, v_k)| =$

$\sum_{i<i' \in [k]} \mathbb{1}[(v_i, v_{i'}) \in E]$. In the sequel, it is convenient to consider this quantity $|E(v_1, \ldots, v_k)|$ in the general case where the vertices are not necessarily distinct; the definition extends as written to this case, and counts the number of edges with multiplicity. We make use of the well-known fact that the CLIQUE decision problem is NP-hard [129]. Recall that a k-clique in an undirected graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ of size $|S| = k$ such that $|E(S)| = \binom{k}{2}$. The CLIQUE decision problem is: given an undirected graph $G = (V, E)$ and an integer $k > 0$, decide whether G contains a k-clique. For technical reasons, it is convenient to use the fact that this CLIQUE problem is NP-hard even in certain special cases; for a proof see the Appendix of [14].

Proposition 7.2.8 (Hardness of CLIQUE)**.** *Assuming* P $\neq$ NP*, there does not exist an algorithm that, given an integer $k > 0$ and a graph G on n vertices, decides whether G contains a clique of size k in* $\mathrm{poly}(n, k)$ *time. This is unchanged if G is assumed regular and k is even.*

■ 7.3 Intractability in high dimension

Here we establish the hardness of the Wasserstein barycenter problem for exact and approximate computation by proving Theorems 7.1.1 and 7.1.2, respectively. The section is organized as follows. In §7.4.1 we prove intractability of the CHEAPEST-HUB problem, which as described above is a special case of the separation oracle corresponding to the dual MOT formulation of the Wasserstein barycenter poblem. This proves Theorems 7.1.1 and 7.1.2 by the general results about MOT in Chapter 5, as detailed in §7.4.2.

For simplicity, throughout §7.3 we ignore discussion of bit complexity since the constructed "hard" instances of Wasserstein barycenters are such that all support points $x_{i,j} \in \mathbb{R}^d$ have $\{-1, 0, 1\}$-valued entries, and thus clearly have polynomial bit complexity. The optimal value of the Wasserstein barycenter problem (7.1) is denoted by $\mathrm{OPT}_{\mathrm{BARY}}$. We show that the claimed hardness results hold even in the special case where the weights $\lambda_1 = \cdots = \lambda_k = 1/k$ are uniform, and thus henceforth specialize solely to this case.

■ 7.3.1 Intractability of separation oracle

Here we show that the CHEAPEST-HUB problem (see Definition 7.2.4) is NP-hard to compute, even approximately.

Lemma 7.3.1 (Inapproximability of CHEAPEST-HUB)**.** *Given vectors $\{x_{i,j}\}_{i \in [k], j \in [n]} \subseteq \{0, 1\}^d$, it is* NP*-hard to compute the value of CHEAPEST-HUB to additive error $0.99/k$.*

The motivation behind our proof is the geometric interpretation of the CHEAPEST-HUB problem, which recall from §7.2.2 is: given k sets $S_i \subset \mathbb{R}^d$, each consisting of n points $S_i = \{x_{i,j}\}_{j\in[n]}$, find one point from each set so as to minimize the average distance to the closest "hub" $y \in \mathbb{R}^d$. On an intuitive level, this question of finding k points which are close somewhat resembles the k-CLIQUE problem, which is the task of finding a set of k vertices in a graph such that all pairs of vertices are close in the sense of being adjacent. Motivated by this intuitive observation, we show a reduction from k-CLIQUE to CHEAPEST-HUB.

A key part of this reduction is figuring out how to appropriately embed the (combinatorial) adjacency properties of a graph G into a (geometric) point configuration in order to encode k-CLIQUE problem as an instance of CHEAPEST-HUB in sufficiently high dimension d. Briefly, we show that, given an n-vertex graph $G = (V, E)$, one can efficiently compute points $\{x_{i,j}\}_{i\in[k],j\in[n]} \subset \mathbb{R}^d$ such that the value of the corresponding CHEAPEST-HUB problem indicates whether G has a clique of size k. Our embedding ensures that G has a clique of size k if and only if there are k points $x_{1,j_1}, \ldots, x_{k,j_k}$ that are sufficiently close to each other. Roughly speaking, we achieve this by setting the nk points $\{x_{i,j}\}_{i\in[k],j\in[n]}$ to be an embedding of k copies of the vertex set V of the graph G, where adjacent vertices are embedded as close points in $\mathbb{R}^d$.

Concretely, this embedding is as follows; see Remark 7.3.3 below for an interpretation of it as the edge-indicator vector embedding of a certain augmented graph. For simplicity of exposition, we make no attempt to optimize d here (dimensionality reduction can be done, e.g., by simply applying the Johnson-Lindenstrauss lemma).

Lemma 7.3.2 (Embedding for CLIQUE)**.** *Given an n-vertex D-regular graph $G = (V, E)$ and an integer $k > 0$, there exists a function $\phi : [k] \times [n] \to \{0,1\}^{\binom{k}{2}n^2}$ satisfying the following.*

(i) ϕ can be evaluated in $\mathrm{poly}(n, k)$ time.

(ii) For all $i \neq i' \in [k]$ and $v, v' \in V$, it holds that $\langle \phi(i, v), \phi(i', v') \rangle = \mathbb{1}[(v, v') \in E]$.

(iii) For all $i \in [k]$ and $v \in V$, it holds that $\|\phi(i, v)\|_2^2 = D(k - 1)$.

Proof. Define the embedding ϕ as follows. Index the $\binom{k}{2}n^2$ coordinates by tuples (ℓ, u, ℓ', u') for indices $\ell < \ell' \in [k]$ and $u, u' \in [n] \cong V$. Set

$$[\phi(i, v)]_{(\ell,u,\ell',u')} = \mathbb{1}[(u, u') \in E] \cdot \mathbb{1}[(\ell, u) = (i, v) \text{ or } (\ell', u') = (i, v)]. \tag{7.3}$$

Property (i) clearly holds since each entry of ϕ is computable in $\mathrm{poly}(n, k)$ time.

To show property (ii), note that the vectors $\phi(i,v)$ and $\phi(i',v')$ have disjoint support if $(v,v') \notin E$, and otherwise share exactly one non-zero entry at the coordinate $(\ell,u,\ell',u') = (i,v,i',v')$. Thus $\langle \phi(i,v), \phi(i',v') \rangle = \mathbb{1}[(v,v') \in E]$.

To show property (iii), note that the squared norm $\|\phi(i,v)\|_2^2$ is equal to the number of non-zeros in the embedded vector $\phi(i,v)$ since it is an indicator vector. Since G is D-regular, counting the number of non-zero entries in $\phi(i,v)$ shows that $\|\phi(i,v)\|_2^2 = D(k-1)$. □

Remark 7.3.3 (Interpretation of embedding via tensor-product graph)**.** *The embedding $\phi : [k] \times [n] \to \mathbb{R}^d$ is the edge-indicator embedding of the graph $\tilde{G}$ which is the tensor product of the complete graph on k vertices and G. That is, $\tilde{G}$ is k-partite and has vertex set $\tilde{V}$ equal to k independent copies of V. A vertex in $\tilde{V}$ can be indexed by a tuple (i,v), where $i \in [k]$ denotes the copy index and $v \in V$ denotes the corresponding vertex in the original graph. Two vertices (i,v) and (i',v') are adjacent in $\tilde{G}$ if and only if $i \neq i'$ and $(v,v') \in E$; that is, if and only if these two vertices in $\tilde{V}$ are from different copies of V and are such that their underlying vertex indices are adjacent in the original graph G.*

Proof of Lemma 7.3.1. We reduce CLIQUE to approximately solving CHEAPEST-HUB. Given an n-vertex D-regular graph $G = (V,E)$ and an integer $k > 0$, let ϕ be the corresponding embedding in Lemma 7.3.2. Set $x_{i,j} = \phi(i,j) \in \{0,1\}^d$ for each $i \in [k]$ and $j \in [n] \cong V$, where $d = \binom{k}{2} n^2$.

Recall that the CHEAPEST-HUB for the input points $\{x_{i,j}\}_{i \in [k], j \in [n]} \subset \mathbb{R}^d$ is the problem of computing the value

$$F^* := \min_{(j_1,\ldots,j_k) \in [n]^k} \min_{y \in \mathbb{R}^d} \sum_{i=1}^{k} \|x_{i,j_i} - y\|_2^2, \tag{7.4}$$

see Definition 7.2.4. This objective simplifies for the particular choice of input points. Indeed,

$$\begin{aligned} \min_{y \in \mathbb{R}^d} \sum_{i=1}^{k} \|x_{i,j_i} - y\|_2^2 &= \left(1 - \frac{1}{k}\right) \sum_{i=1}^{k} \|x_{i,j_i}\|_2^2 - \frac{2}{k} \sum_{i<i' \in [k]} \langle x_{i,j_i}, x_{i',j_{i'}} \rangle \\ &= D(k-1)^2 - \frac{2}{k} \sum_{i<i' \in [k]} \mathbb{1}[(j_i, j_{i'}) \in E] \\ &= M - \frac{2}{k} |E(j_1,\ldots,j_k)|. \end{aligned} \tag{7.5}$$

Above, the first step is by plugging in the closed-form solution for $y = \frac{1}{k} \sum_{i=1}^{k} x_{i,j_i}$. The second step is by using the key properties (ii) and (iii) of the embedding ϕ

in Lemma 7.3.2. The third step is by defining $M := D(k-1)^2$ and recalling that we write $|E(j_1, \ldots, j_k)| = \sum_{i<i' \in [k]} \mathbb{1}[(j_i, j_{i'}) \in E]$ to denote the number of edges in the induced subgraph of G with vertices $j_1, \ldots, j_k$ where we count the edges with multiplicity if $j_1, \ldots, j_k$ are not distinct (see §7.2.4).

Therefore by combining (7.4) and (7.5), we conclude that the value F^* of the CHEAPEST-HUB problem for the particular chosen input points $\{x_{i,j}\}_{i \in [k], j \in [n]} \subset \mathbb{R}^d$ is equal to

$$F^* = M - \frac{2}{k} \cdot \max_{(j_1, \ldots, j_k) \in [n]^k} |E(j_1, \ldots, j_k)| = M - \frac{2}{k} \cdot \max_{S \in V^k} |E(S)|. \tag{7.6}$$

Note that in this final equation, the optimization is over a multiset S of k vertices in V that are not necessarily distinct. This is a multiset rather than a set because CHEAPEST-HUB optimizes over tuples $(j_1, \ldots, j_k) \in [n]^k$, and such a tuple does not necessarily consist of distinct indices; this is also why we defined $|E(S)|$ to count edges with multiplicity.

Using (7.6), we now argue that the value F^* of CHEAPEST-HUB varies significantly depending on whether G contains a clique of size k. Specifically, on one hand

$$F^* = M - k + 1, \qquad \text{if } G \text{ contains a clique of size } k\ , \tag{7.7}$$

because using this clique as the set S and plugging into (7.6) yields objective value $M - \frac{2}{k} \cdot \binom{k}{2} = M - k + 1$. On the other hand,

$$F^* \geqslant M - k + 1 - \frac{2}{k}, \qquad \text{if } G \text{ does not contain a clique of size } k\ , \tag{7.8}$$

because then $\max_{S \in V^k} |E(S)| \leqslant \binom{k}{2} - 1$, whereby from (7.6) we conclude that $F^* \geqslant M - \frac{2}{k}(\binom{k}{2} - 1) = M - k + 1 - \frac{2}{k}$. It therefore follows from (7.7) and (7.8) that computing CHEAPEST-HUB to any additive error less than $\frac{1}{k}$ enables one to decide whether G contains a clique of size k, which is an NP-hard problem [129]. □

■ 7.3.2 Putting the pieces together

Here we make precise how Lemma 7.3.1 implies Theorems 7.1.1 and 7.1.2.

Proof of Theorem 7.1.1. If there is a $\mathrm{poly}(n, k, d)$-time algorithm for computing $\mathrm{OPT}_{\mathrm{BARY}}$, then by Proposition 7.2.1 there is a $\mathrm{poly}(n, k, d)$-time algorithm for computing MOT_C with the cost tensor C given by (7.2), thus by Theorem 5.3.2 there is a $\mathrm{poly}(n, k, d)$-time algorithm for CHEAPEST-HUB. Assuming P ≠ NP, this contradicts the NP-hardness of CHEAPEST-HUB in Lemma 7.3.1. □

Proof of Theorem 7.1.2. If there is a $\mathrm{poly}(n, k, d, R/\varepsilon)$-time randomized algorithm for ε-approximating $\mathrm{OPT}_{\mathrm{BARY}}$ with probability 2/3, then by Proposition 7.2.1 there is a $\mathrm{poly}(n, k, d, R/\varepsilon)$-time randomized algorithm for ε-approximating MOT_C with probability 2/3. By a standard boosting argument—namely repeating this algorithm $\log 1/\delta$ times, taking the median, and applying a Chernoff bound—this implies a $\mathrm{poly}(n, k, d, R/\varepsilon, \log 1/\delta)$-time randomized algorithm for ε-approximating MOT_C with probability $1 - \delta$. Therefore, by setting δ sufficiently high, we conclude by Theorem 5.3.3 and a union bound that there exists a $\mathrm{poly}(n, k, d, R/\varepsilon)$-time randomized algorithm for $\varepsilon(nk)^\alpha$-approximating CHEAPEST-HUB with probability of success at least 0.51. This can be boosted to 2/3 probability of success by another standard boosting argument, proving that CHEAPEST-HUB lies in BPP. However, assuming $\mathsf{NP} \not\subset \mathsf{BPP}$, this contradicts the NP-hardness in Lemma 7.3.1. □

7.4 Polynomial-time algorithm in fixed dimension

In this section we prove Theorems 7.1.3 and 7.1.4, describe the algorithm in these theorems, and provide preliminary numerical simulations demonstrating that this algorithm enables computing exact solutions at previously intractable sizes.

The section is organized as follows. In §7.4.1 we design a polynomial-time algorithm for the separation oracle corresponding to the dual MOT formulation of the low-dimensional Wasserstein barycenter problem. This proves Theorems 7.1.3 and 7.1.4 aside from checking bit-complexity details, which is done formally in §7.4.2. Finally §7.4.3 provides preliminary numerical simulations.

For simplicity, throughout §7.4 we assume without loss of generality that each λ_i is strictly positive, since otherwise μ_i does not affect the barycenter (see (7.1)).

7.4.1 Efficient separation oracle

Here we design a polynomial-time algorithm for the separation oracle corresponding to the dual MOT formulation of the low-dimensional Wasserstein barycenter problem. This is formally stated as follows.

Proposition 7.4.1. *If the cost C is given by* (7.2)*, then the oracle $\mathsf{MIN}_C(p)$ can be implemented in* $\mathrm{poly}(n, k, \log U)$ *time, where* $\log U$ *is the number of bits of precision needed to represent the points $x_{i,j} \in \mathbb{R}^d$, weights $\lambda_i \in \mathbb{R}^k_{>0}$ and potentials $p \in \mathbb{R}^{n\times k}$.*

Recall from §7.2.2 that this MIN_C oracle requires computing the value of

$$\min_{\vec{j}\in[n]^k} \min_{y\in\mathbb{R}^d} g(\vec{j}, y) \tag{7.9}$$

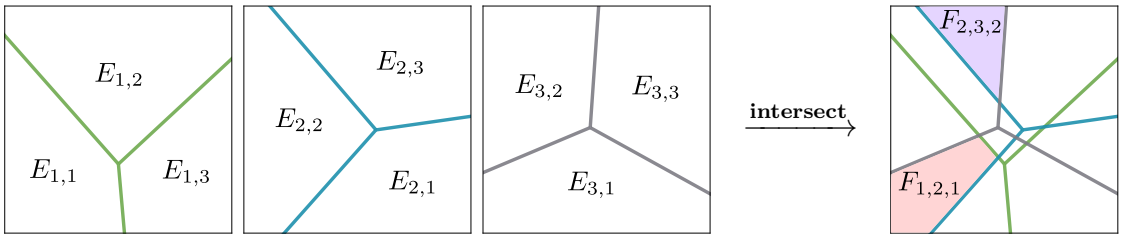


Figure 7.3: Illustrates $k = 3$ power diagrams $\{\{E_{i,j}\}_{j\in[n]}\}_{i\in[k]}$ each with $n = 3$ cells, and their intersection $\{F_{\vec{j}}\}_{\vec{j}\in[n]^k}$. For instance, the red cell in the intersected diagram is $F_{1,2,1} = E_{1,1} \cap E_{2,2} \cap E_{3,1}$, and the purple cell is $F_{2,3,2} = E_{1,2} \cap E_{2,3} \cap E_{3,2}$. Note that the intersected diagram has only 13 non-empty cells, which is less than $n^k = 27$ (c.f., Lemma 7.4.3).

where

$$g(\vec{j}, y) := \sum_{i=1}^{k} \lambda_i(\|x_{i,j_i} - y\|^2 - [w_i]_{j_i}),$$

and w_i denotes p_i/λ_i. At a high level, our approach is to swap the order of minimization, optimize over $y \in \mathbb{R}^d$, and then (easily) recover an optimal tuple from this optimal y. The difficulty is in the optimization over $y \in \mathbb{R}^d$. The key to performing this efficiently is partitioning the space $\mathbb{R}^d$ into a "cell complex" such that (i) the optimization over y in each cell is easy, and (ii) there are only polynomially many cells. Operationally, this allows us to reduce the separation oracle optimization (7.9) to optimizing over only a polynomially sized set of candidate tuples in $[n]^k$—one for each cell—which we moreover show can be efficiently identified and enumerated.

To formalize this, we make the following key definitions. Define for each $i \in [k]$ and $j \in [n]$ the set

$$E_{i,j} = \{y \in \mathbb{R}^d : \|x_{i,j} - y\|^2 - [w_i]_j < \|x_{i,j'} - y\|^2 - [w_i]_{j'}, \forall j' \neq j\}, \tag{7.10}$$

and define for each tuple $\vec{j} \in [n]^k$ the set

$$F_{\vec{j}} = \bigcap_{i=1}^{k} E_{i,j_i}. \tag{7.11}$$

Geometrically, for each $i \in [k]$, the cells $\{E_{i,j}\}_{j\in[n]}$ form a power diagram (see §7.2.3) on the spheres $S(x_{i,1}, r_{i,1}), \ldots, S(x_{i,n}, r_{i,n})$, where the j-th sphere is centered at point $x_{i,j}$ and has radius $r_{i,j} := \sqrt{[w_i]_j - \min_{j'}[w_i]_{j'}} \geqslant 0$. Each power diagram essentially partitions $\mathbb{R}^d$ in the sense that its constituent cells are disjoint

and their closures cover $\mathbb{R}^d$; see Figure 7.2 for an illustration. The cell complex $\{F_{\vec{j}}\}_{\vec{j}\in[n]^k}$ is the intersection of these k power diagrams and essentially partitions $\mathbb{R}^d$ in the analogous way; see Figure 7.3 for an illustration.

The heart of our algorithm lies in the following two lemmas. The first lemma shows that the optimization (7.9) over the exponentially many tuples $j \in [n]^k$ may be restricted to just those whose corresponding cell $F_{\vec{j}}$ is non-empty, i.e., we may restrict to the tuples in

$$T := \{\vec{j} \in [n]^k \ : \ F_{\vec{j}} \neq \emptyset\}. \tag{7.12}$$

The second lemma shows that this candidate set T contains only polynomially many tuples and moreover can be efficiently enumerated. Briefly, the first lemma exploits the fact that the optimization over $y \in \mathbb{R}^d$ is equivalent to optimizing over the cells in $F_{\vec{j}}$, and the second lemma exploits complexity bounds for the intersections of power diagrams. Together, these lemmas are sufficient to efficiently solve the separation oracle because for any fixed $\vec{j}$, the value $\min_{y\in\mathbb{R}^d} g(\vec{j}, y)$ can be efficiently computed in closed-form (as shown below in (7.13)).

Lemma 7.4.2. *The optimization over $\vec{j} \in [n]^k$ in the separation oracle problem* (7.9) *can be equivalently restricted to $\vec{j} \in T$. That is,*

$$\min_{\vec{j}\in[n]^k} \min_{y\in\mathbb{R}^d} g(\vec{j}, y) = \min_{\vec{j}\in T} \min_{y\in\mathbb{R}^d} g(\vec{j}, y).$$

Proof. The inequality "$\leqslant$" is obvious; we show the other inequality "$\geqslant$". By swapping the order of minimization and using the fact that $\{F_{\vec{j}}\}_{\vec{j}\in T}$ cover $\mathbb{R}^d$ modulo closure, we have

$$\min_{\vec{l}\in[n]^k} \min_{y\in\mathbb{R}^d} g(\vec{l}, y) = \min_{y\in\mathbb{R}^d} \min_{\vec{l}\in[n]^k} g(\vec{l}, y) = \min_{\vec{j}\in T} \min_{y\in\overline{F_{\vec{j}}}} \min_{\vec{l}\in[n]^k} g(\vec{l}, y).$$

We claim that the inner minimization over $\vec{l}$ is explicit: $\vec{l} = \vec{j}$. Indeed, by separability of g in the coordinates of $\vec{l}$ and non-negativity of λ_i, for each $i \in [n]$ the optimal ℓ_i is a solution of $\arg\min_{\ell_i\in[n]} \|x_{i,\ell_i} - y\|^2 - [w_i]_{\ell_i}$; and j_i is a solution of this by definition of E_{i,j_i} (see (7.10)) and the fact that $\overline{E_{i,j_i}}$ contains y (by definition of $F_{\vec{j}}$, see (7.11)). Therefore

$$\min_{\vec{j}\in T} \min_{y\in\overline{F_{\vec{j}}}} \min_{\vec{l}\in[n]^k} g(\vec{l}, y) = \min_{\vec{j}\in T} \min_{y\in\overline{F_{\vec{j}}}} g(\vec{j}, y).$$

Now by enlarging the optimization region, we have the simple bound

$$\min_{\vec{j}\in T} \min_{y\in\overline{F_{\vec{j}}}} g(\vec{j}, y) \geqslant \min_{\vec{j}\in T} \min_{y\in\mathbb{R}^d} g(\vec{j}, y).$$

Combining the above three displays completes the proof. □

Lemma 7.4.3. *For any fixed dimension d, the set T can be enumerated in* $\mathrm{poly}(n, k, \log U)$ *time.*

Proof. By Lemma 7.2.6, the $O(nk)$ total facets for the k power diagrams $\{\{E_{i,j}\}_{i\in[n]}\}_{j\in[k]}$ can be computed in $\mathrm{poly}(n, k, \log U)$ time. For each facet, compute the $(d-1)$-dimensional hyperplane it lies in. The cell complex $\mathcal{H}$ formed by these hyperplanes is a subcomplex of the cell complex formed by intersecting the power diagrams. By Lemma 7.2.7, we can enumerate the cells in $\mathcal{H}$ in $\mathrm{poly}(n, k, \log U)$ time. For each cell in $\mathcal{H}$, the corresponding tuple $\vec{j} \in [n]^k$ is computable in $O(nk \cdot \mathrm{polylog}\, U)$ time by computing the k coordinates of the tuple separately. Since each non-empty cell $F_{\vec{j}}$ contains at least one cell in $\mathcal{H}$, this process enumerates all tuples in T. □

We now conclude the desired efficient algorithm for the separation oracle.

Proof of Proposition 7.4.1. By Lemma 7.4.2, it suffices to compute the optimal value of $\min_{\vec{j}\in T} \min_{y\in\mathbb{R}^d} g(\vec{j}, y)$. Since $\sum_{i=1}^k \lambda_i x_{i,j_i} \in \arg\min_{y\in\mathbb{R}^d} g(\vec{j}, y)$, it therefore suffices to solve

$$\min_{\vec{j}\in T} \sum_{i=1}^k \lambda_i \|x_{i,j_i}\|^2 - \|\sum_{i=1}^k \lambda_i x_{i,j_i}\|^2 - \sum_{i=1}^k \lambda_i [w_i]_{j_i}. \tag{7.13}$$

Peform this by enumerating the set T using the algorithm in Lemma 7.4.3. □

■ 7.4.2 Putting the pieces together

Proof of Theorem 7.1.4. Assume each x_{i,j_i} and λ_i is written to $\log U$ bits of precision. Since each entry of the cost tensor (7.2) requires only $O(\log k + \log U)$ bits of precision, and since the parameter $w \in \mathbb{R}^{n\times k}$ in each MIN_C query made by the algorithm in Theorem 6.4.1 requires only $\mathrm{poly}(n, k, \log U)$ bits of precision, it follows that the algorithm in Theorem 6.4.1 combined with the MIN_C oracle implementation in Proposition 7.4.1 computes a vertex solution P^* for (MOT) in $\mathrm{poly}(n, k, \log U)$ time. By Lemma 7.2.3, P^* has at most $nk - k + 1$ non-zero entries. Thus we can recover from P^* an optimal barycenter ν with support size at most $nk - k + 1$ in time $\mathrm{poly}(n, k, \log U)$ by the reduction in Lemma 7.2.2. □

Proof of Proof of Theorem 7.1.3. By rounding both the weights λ_i and the coordinates of the atoms $x_{i,j} \in \mathbb{R}^d$ to $\mathrm{poly}(\varepsilon/(Rkd))$ additive accuracy, it can be ensured that each of these numbers requires only $O(\log(Rkd/\varepsilon))$ bits of precision and also that the objective function $\nu \mapsto \sum_{i=1}^k \lambda_i \mathcal{W}(\mu_i, \nu)$ for the barycenter optimization (7.1) is preserved pointwise to ε additive accuracy. This follows from a straightforward calculation and the fact (immediate from the definition of Optimal Transport and an application of Hölder's inequality) that if the squared Euclidean

distance between each atom of μ_i and each atom of ν is preserved up to ε' additive accuracy, then the squared 2-Wasserstein distance $\mathcal{W}(\mu_i, \nu)$ is preserved up to ε' additive accuracy. Now solve the barycenter problem for the rounded weights and atoms exactly using Theorem 7.1.4. □

7.4.3 Numerical implementation

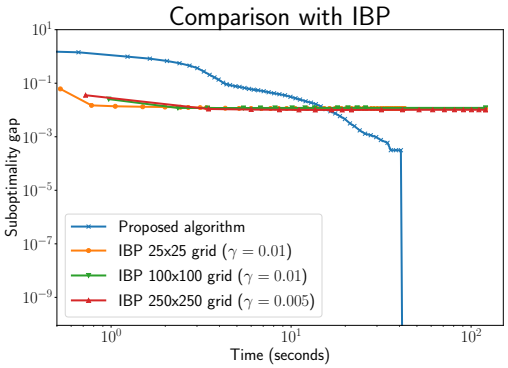
While the focus of this section is theoretical, here we briefly mention that a slight variant of our algorithm can provide high-precision solutions at previously intractable problem sizes. To demonstrate this, we implement our algorithm for dimension $d = 2$ in Python. The only difference between our numerical implementation and the theoretical algorithm described above is that we use a standard Column Generation method for the "outer loop" in step 1 rather than the Ellipsoid algorithm due to its good practical performance; for details see §6.4.1.3. Code and further implementation details are provided on Github.[4]

Computing Exact Solutions at Previously Intractable Scales. Figure 7.4 demonstrates that our algorithm solves the barycenter problem (7.1) to machine precision on an instance with $k = 10$ uniform distributions each on $n = 20$ points randomly drawn from $[-1, 1]^2 \subset \mathbb{R}^2$. In contrast, existing popular barycenter algorithms which use the fixed-support assumption can converge faster but only to lower-precision approximations. This is because the $\Theta(1/\varepsilon^d)$ gridsize that they require for ε-additive approximation results in a large-scale LP which is prohibitive even for relatively low precision ε; see the previous work section for details. Note also that a standard LP solver requires optimizing over $n^k = 20^{10} \approx 10^{13}$ variables for the LP formulation (MOT) and thus is clearly infeasible at this scale.

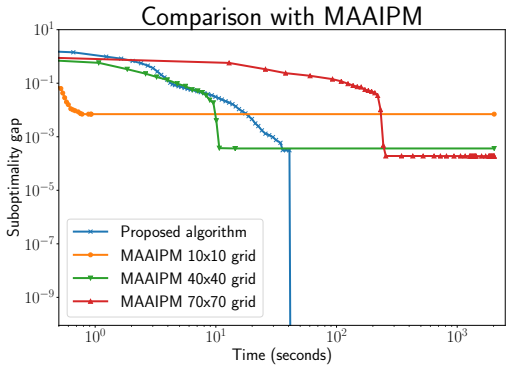
Sharper Visualizations. Here we demonstrate that the high-precision solutions computed by our algorithm yield significantly sharper visualizations than the low-precision solutions that were previously computable. Specifically, here we compare our barycenter algorithm against state-of-the-art methods on a standard benchmark dataset of images of nested ellipses [74, 121]. This dataset consists of $k = 10$ images, each of size 60×60. Five of these images are shown in Figure 7.5. Figure 7.6 contains a visual comparison of the exact barycenter computed by our algorithm and the approximate barycenters produced by the most competitive algorithms tested in the recent paper [121]. All are fixed-support algorithms except for the algorithm of [147].

Of the compared algorithms, MAAIPM gives the most accurate barycenter approximation. It uses a 60×60 grid fixed-support assumption. Although MAAIPM solves this fixed-support problem exactly, the support of an optimal barycenter

[4] `https://github.com/eboix/high_precision_barycenters`



(a) Comparison with the Iterated Bregman Projection (IBP) algorithm of [206] using their implementation `https://github.com/gpeyre/2015-SIGGRAPH-convolutional-ot`.



(b) Comparison with the Matrix-based Adaptive Alternating Interior-Point Method (MAAIPM) of [92] using their implementation `https://gitlab.com/ZXiong/wasserstein-barycenter`.

Figure 7.4: Comparison with state-of-the-art algorithms. The y-axis is the suboptimality for the barycenter optimization (7.1); note that while standard LP solvers cannot be run at this scale, our algorithm yields an exact solution (certified by our separation oracle) which enables plotting this suboptimality. Both compared algorithms require a fixed-support assumption and are run on uniform grids of increasing sizes. IBP has an additional parameter: the entropic regularization γ, which significantly impacts the algorithm's accuracy and numerical stability, see [178, 206]. We provide a generous comparison here for IBP by (i) fine-tuning γ for it (we binary search for the most accurate γ; note that their code does not always converge for γ small due to numerical instability); and (ii) exactly computing the Wasserstein distances $\mathcal{W}(\mu_i, \nu)$ to IBP's current barycenter ν in the barycenter objective (7.1) using [79], which is more accurate than IBP's approximation (this is slow for large grids but is not counted in IBP's timing). Our algorithm finds an exact barycenter after $\sim$50 seconds. All experiments are run on a standard 2014 Lenovo Yoga 720-13IKB laptop.

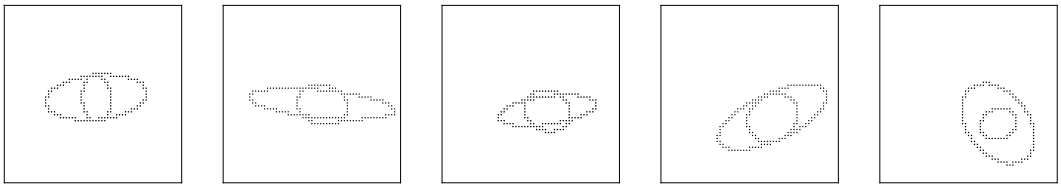


Figure 7.5: Five sample images from the nested ellipses dataset in [121].

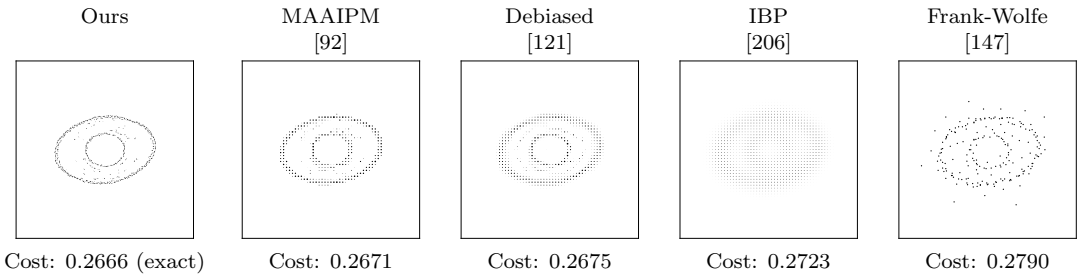


Figure 7.6: Comparison of barycenter algorithms on a standard benchmark dataset of ellipse images. Each barycenter atom is plotted as a disk with area proportional to its probability mass. All compared methods are run with the code, parameter choices, and dataset of [121].

does not lie on a 60×60 grid, and thus MAAIPM only computes an approximate barycenter. A natural approach is to run MAAIPM on a finer grid discretization, i.e., finer than 60×60. However, this does not work, since MAAIPM does not scale to much larger grid sizes (see also Figure 7.4b).

The other two fixed-support algorithms are based on entropic regularization: debiased Sinkhorn barycenters [121] and IBP [206]. These use entropic regularization parameter $\gamma = 0.002$ and the same 60×60 fixed-support approximation as MAAIPM. Again, these methods produce suboptimal barycenters. While these methods scale to larger grid sizes than MAAIPM, this results in qualitatively similar and blurry visualizations as in this 60×60 case due to the entropic regularization.

The final compared algorithm is the free-support algorithm of [147], which is based on the Frank-Wolfe algorithm. Although this method does not make a fixed-support assumption, it still returns an approximate solution due to the approximate nature of the Frank-Wolfe algorithm.

■ 7.5 Discussion

Over the past decade, Wasserstein barycenters have become central to diverse applications in data science. However, fundamental questions about their computational complexity remained open—in particular, it was previously unknown whether they can be computed in polynomial time. This chapter addressed this issue by giving two results that, together, resolve the computational complexity of Wasserstein barycenters. On one hand, we showed that under standard complexity-theoretic assumptions, it is impossible to compute arbitrarily close approximations for the Wasserstein barycenter problem in polynomial time. And on the other hand, we gave the first algorithm that, in any fixed dimension d,

solves the barycenter problem exactly or to high precision in polynomial time.

Our results motivate several interesting research directions.

Future directions for high-dimensional computation. Our intractability results motivate the question: what properties of distributions enable efficient computation of Wasserstein barycenters? A first candidate could be to require all distributions to be uniform discrete distributions, but unfortunately, this does not help from a computational complexity perspective; for details see the paper [14] upon which this chapter is partially based.

Nevertheless, there is a growing body of work that shows that other assumptions do help, both in theory and practice. For example, polynomial-time computation of Wasserstein barycenters is possible for certain parametric families of high-dimensional distributions such as Gaussian distributions, or more generally location-scatter families [20]. There is also recent work that shows promising empirical results for computing barycenters of high-dimensional distributions if they are supported on low-dimensional manifolds or are represented by a convolutional neural network generative model [68]. Other assumptions that might be interesting to investigate are if the input distributions μ_i are drawn from some generative process, or if the points in their supports lie in structured geometric configurations. Further understanding what commonly arising properties of distributions ensure efficient computation would have immediate impact on the many data-science applications of Wasserstein barycenters.

Future directions for low-dimensional computation. While Theorem 7.1.4 answers the polynomial-time computability of low-dimensional Wasserstein barycenters from a *theoretical* perspective, from a *practical* perspective it is still a hard and interesting problem to compute high-precision barycenters for large-scale inputs. Indeed, our current implementation is not efficient beyond moderate-scale inputs; and while existing algorithms such as IBP scale to larger inputs, they have limited accuracy. Moreover, all existing algorithms pay for the curse of dimensionality in one way or another. We emphasize that our implementation does not contain further optimizations or heuristics; it is an interesting direction for future work to investigate potential such options including pruning cutting planes, warm starts, and specially tailored algorithms for the power diagram intersections in §7.4.1 (e.g., in $\mathbb{R}^2$ or $\mathbb{R}^3$, settings which commonly arise in image processing and computer graphics applications).

References

[1] Duncan Adamson, Argyrios Deligkas, Vladimir V Gusev, and Igor Potapov. On the hardness of energy minimisation for crystal structure prediction. In *International Conference on Current Trends in Theory and Practice of Informatics*, pages 587–596. Springer, 2020.

[2] Shipra Agrawal, Yichuan Ding, Amin Saberi, and Yinyu Ye. Price of correlations in stochastic optimization. *Operations Research*, 60(1):150–162, 2012.

[3] Martial Agueh and Guillaume Carlier. Barycenters in the Wasserstein space. *SIAM Journal on Mathematical Analysis*, 43(2):904–924, 2011.

[4] Amir Ali Ahmadi and Pablo A Parrilo. Joint spectral radius of rank one matrices and the maximum cycle mean problem. In *Conference on Decision and Control*, pages 731–733. IEEE, 2012.

[5] Ravindra K Ahuja and James B Orlin. Inverse optimization. *Operations Research*, 49(5): 771–783, 2001.

[6] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. Network flows. 1988.

[7] Yasunori Akagi, Yusuke Tanaka, Tomoharu Iwata, Takeshi Kurashima, and Hiroyuki Toda. Probabilistic optimal transport based on collective graphical models. *Preprint at arXiv:2006.08866*, 2020.

[8] Zeyuan Allen-Zhu, Zheng Qu, Peter Richtárik, and Yang Yuan. Even faster accelerated coordinate descent using non-uniform sampling. In *International Conference on Machine Learning*, pages 1110–1119, 2016.

[9] Zeyuan Allen-Zhu, Yuanzhi Li, Rafael Oliveira, and Avi Wigderson. Much faster algorithms for matrix scaling. In *Symposium on the Foundations of Computer Science*. IEEE, 2017.

[10] Jason Altschuler, Francis Bach, Alessandro Rudi, and Jonathan Niles-Weed. Massively scalable Sinkhorn distances via the Nyström method. In *Conference on Neural Information Processing Systems*, pages 4429–4439, 2019.

[11] Jason M Altschuler and Enric Boix-Adserà. Wasserstein barycenters can be computed in polynomial time in fixed dimension. *Journal of Machine Learning Research*, 22:44–1, 2021.

[12] Jason M Altschuler and Enric Boix-Adserà. Hardness results for Multimarginal Optimal Transport problems. *Discrete Optimization*, 42:100669, 2021.

[13] Jason M Altschuler and Enric Boix-Adserà. Polynomial-time algorithms for Multimarginal Optimal Transport problems with structure. *Mathematical Programming*, pages 1–72, 2022.

[14] Jason M Altschuler and Enric Boix-Adserà. Wasserstein barycenters are NP-hard to compute. *SIAM Journal on Mathematics of Data Science*, 4(1):179–203, 2022.

[15] Jason M Altschuler and Pablo A Parrilo. Lyapunov exponent of rank-one matrices: Ergodic formula and inapproximability of the optimal distribution. *SIAM Journal on Control and Optimization*, 58(1):510–528, 2020.

[16] Jason M Altschuler and Pablo A Parrilo. Approximating Min-Mean-Cycle for low-diameter graphs in near-optimal time and memory. *Preprint at arXiv:2004.03114 v1*, 2020.

[17] Jason M Altschuler and Pablo A Parrilo. Near-linear convergence of the Random Osborne algorithm for Matrix Balancing. *Mathematical Programming*, pages 1–35, 2022.

[18] Jason M Altschuler and Pablo A Parrilo. Approximating Min-Mean-Cycle for low-diameter graphs in near-optimal time and memory. *SIAM Journal on Optimization*, 32(3):1791–1816, 2022.

[19] Jason M Altschuler, Jonathan Weed, and Philippe Rigollet. Near-linear time approximation algorithms for optimal transport via Sinkhorn iteration. In *Conference on Neural Information Processing Systems*, 2017.

[20] Jason M Altschuler, Sinho Chewi, Patrik Gerber, and Austin J Stromme. Averaging on the Bures-Wasserstein manifold: dimension-free convergence of gradient descent. In *Conference on Neural Information Processing Systems*, 2021.

[21] Pedro C Álvarez-Esteban, E Del Barrio, JA Cuesta-Albertos, and C Matrán. A fixed-point approach to barycenters in Wasserstein space. *Journal of Mathematical Analysis and Applications*, 441(2):744–762, 2016.

[22] Ethan Anderes, Steffen Borgwardt, and Jacob Miller. Discrete Wasserstein barycenters: Optimal transport for discrete data. *Mathematical Methods of Operations Research*, 84 (2):389–409, 2016.

[23] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999. ISBN 0-89871-447-8 (paperback).

[24] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pages 214–223, 2017.

[25] Franz Aurenhammer. Power diagrams: properties, algorithms and applications. *SIAM Journal on Computing*, 16(1):78–96, 1987.

[26] Kyriakos Axiotis, Aleksander Madry, and Adrian Vladu. Circulation control for faster minimum cost flow in unit-capacity graphs. In *Symposium on Foundations of Computer Science*, pages 93–104. IEEE, 2020.

[27] Francis Bach. Learning with submodular functions: a convex optimization perspective. *Foundations and Trends in Machine Learning*, 6(2-3):145–373, 2013.

[28] Michael O Ball. Computational complexity of network reliability analysis: An overview. *IEEE Transactions on Reliability*, 35(3):230–239, 1986.

[29] Michael O Ball, Charles J Colbourn, and J Scott Provan. Network reliability. *Handbooks in Operations Research and Management Science*, 7:673–762, 1995.

[30] RB Bapat, David P Stanford, and P van den Driessche. The eigenproblem in max algebra. Technical report, 1993.

[31] Leonid Barenboim, Michael Elkin, and Fabian Kuhn. Distributed (Δ+1)-coloring in linear (in Δ) time. *SIAM Journal on Computing*, 43(1):72–95, 2014.

[32] Jean-David Benamou, Guillaume Carlier, Marco Cuturi, Luca Nenna, and Gabriel Peyré. Iterative Bregman projections for regularized transportation problems. *SIAM Journal on Scientific Computing*, 37(2):A1111–A1138, 2015.

[33] Jean-David Benamou, Guillaume Carlier, and Luca Nenna. A numerical method to solve multi-marginal optimal transport problems with Coulomb cost. In *Splitting Methods in Communication, Imaging, Science, and Engineering*, pages 577–601. Springer, 2016.

[34] Jean-David Benamou, Guillaume Carlier, Simone Di Marino, and Luca Nenna. An entropy minimization approach to second-order variational mean-field games. *Mathematical Models and Methods in Applied Sciences*, 29(08):1553–1583, 2019.

[35] Jean-David Benamou, Guillaume Carlier, and Luca Nenna. Generalized incompressible flows, multi-marginal transport and Sinkhorn algorithm. *Numerische Mathematik*, 142(1): 33–54, 2019.

[36] Dimitri P Bertsekas and John N Tsitsiklis. *Parallel and distributed computation: numerical methods*, volume 23. Prentice Hall Englewood Cliffs, NJ, 1989.

[37] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena Scientific Belmont, MA, 1997.

[38] Jérémie Bigot, Raúl Gouet, Thierry Klein, and Alfredo López. Geodesic PCA in the Wasserstein space by convex PCA. In *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*, volume 53, pages 1–26. Institut Henri Poincaré, 2017.

[39] Jose Blanchet, Arun Jambulapati, Carson Kent, and Aaron Sidford. Towards optimal running times for optimal transport. *Preprint at arXiv:1810.07717*, 2018.

[40] Mathieu Blondel, Vivien Seguy, and Antoine Rolet. Smooth and sparse optimal transport. In *International Conference on Artificial Intelligence and Statistics*, pages 880–889. PMLR, 2018.

[41] Hans L Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

[42] Hans L Bodlaender. Treewidth: Structure and algorithms. In *International Colloquium on Structural Information and Communication Complexity*, pages 11–25. Springer, 2007.

[43] Béla Bollobás. *Random graphs*. Number 73. Cambridge University Press, 2001.

[44] Nicolas Bonneel, Michiel Van De Panne, Sylvain Paris, and Wolfgang Heidrich. Displacement interpolation using Lagrangian mass transport. In *ACM Transactions on Graphics*, volume 30, page 158. ACM, 2011.

[45] Steffen Borgwardt. An LP-based, strongly-polynomial 2-approximation algorithm for sparse Wasserstein barycenters. *Operational Research*, pages 1–41, 2020.

[46] Steffen Borgwardt and Stephan Patterson. On the computational complexity of finding a sparse Wasserstein barycenter. *Journal of Combinatorial Optimization*, 41(3):736–761, 2021.

[47] Yann Brenier. The least action principle and the related concept of generalized flows for incompressible perfect fluids. *Journal of the American Mathematical Society*, 2(2): 225–255, 1989.

[48] Yann Brenier. The dual least action problem for an ideal, incompressible fluid. *Archive for Rational Mechanics and Analysis*, 122(4):323–351, 1993.

[49] Yann Brenier. Minimal geodesics on groups of volume-preserving maps and generalized solutions of the Euler equations. *Communications on Pure and Applied Mathematics*, 52 (4):411–452, 1999.

[50] Yann Brenier. Generalized solutions and hydrostatic approximation of the Euler equations. *Physica D: Nonlinear Phenomena*, 237(14-17):1982–1988, 2008.

[51] Sébastien Bubeck et al. Convex optimization: Algorithms and complexity. *Foundations and Trends in Machine Learning*, 8(3-4):231–357, 2015.

[52] Giuseppe Buttazzo, Luigi De Pascale, and Paola Gori-Giorgi. Optimal-transport formulation of electronic density-functional theory. *Physical Review A*, 85(6):062502, 2012.

[53] Guillaume Carlier and Ivar Ekeland. Matching for teams. *Economic Theory*, 42(2):397–418, 2010.

[54] Guillaume Carlier and Bruno Nazaret. Optimal transportation for the determinant. *ESAIM: Control, Optimisation and Calculus of Variations*, 14(4):678–698, 2008.

[55] Guillaume Carlier, Adam Oberman, and Edouard Oudet. Numerical methods for matching for teams and Wasserstein barycenters. *ESAIM: Mathematical Modelling and Numerical Analysis*, 49(6):1621–1642, 2015.

[56] Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge University Press, 2006.

[57] Deeparnab Chakrabarty and Sanjeev Khanna. Better and simpler error analysis of the Sinkhorn-Knopp algorithm for matrix scaling. In *Symposium on Simplicity in Algorithms*, 2018.

[58] Nitin Chandrachoodan, Shuvra S Bhattacharyya, and KJ Ray Liu. Adaptive negative cycle detection in dynamic graphs. In *International Symposium on Circuits and Systems*, volume 5, pages 163–166. IEEE, 2001.

[59] Krishnendu Chatterjee, Monika Henzinger, Sebastian Krinninger, Veronika Loitzenbauer, and Michael A Raskin. Approximating the minimum cycle mean. *Theoretical Computer Science*, 547:104–116, 2014.

[60] Bernard Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, 10(4):377–409, 1993.

[61] Louis Chen, Will Ma, Karthik Natarajan, David Simchi-Levi, and Zhenzhen Yan. Distributionally robust linear and discrete optimization with marginals. *Operations Research*, 2022.

[62] Tzu-Yi Chen. Balancing sparse matrices for computing eigenvalues. Master's thesis, UC Berkeley, 5 1998.

[63] Tzu-Yi Chen and James W Demmel. Balancing sparse matrices for computing eigenvalues. *Linear Algebra and its Applications*, 309(1-3):261–287, 2000.

[64] Sinho Chewi, Tyler Maunu, Philippe Rigollet, and Austin J Stromme. Gradient descent algorithms for Bures-Wasserstein barycenters. In *Conference on Learning Theory*, pages 1276–1304, 2020.

[65] Pierre-André Chiappori, Robert J McCann, and Lars P Nesheim. Hedonic price equilibria, stable matching, and optimal transport: equivalence, topology, and uniqueness. *Economic Theory*, 42(2):317–354, 2010.

[66] Jean Cochet-Terrasson, Guy Cohen, Stéphane Gaubert, Michael McGettrick, and Jean-Pierre Quadrat. Numerical computation of spectral elements in max-plus algebra. In *Conference on System Structure and Control*, 1998.

[67] Michael B Cohen, Aleksander Madry, Dimitris Tsipras, and Adrian Vladu. Matrix scaling and balancing via box constrained Newton's method and interior point methods. In *Symposium on the Foundations of Computer Science*, pages 902–913. IEEE, 2017.

[68] Samuel Cohen, Michael Arbel, and Marc Peter Deisenroth. Estimating barycenters of measures in high dimensions. *Preprint at arXiv:2007.07105*, 2020.

[69] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, 2009.

[70] Codina Cotar, Gero Friesecke, and Claudia Klüppelberg. Density functional theory and optimal transportation with Coulomb cost. *Communications on Pure and Applied Mathematics*, 66(4):548–599, 2013.

[71] N Courty, R Flamary, D Tuia, and A Rakotomamonjy. Optimal transport for domain adaptation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1, 2016.

[72] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[73] Marco Cuturi. Sinkhorn distances: lightspeed computation of optimal transport. *Conference on Neural Information Processing Systems*, 26:2292–2300, 2013.

[74] Marco Cuturi and Arnaud Doucet. Fast computation of Wasserstein barycenters. In *International Conference on Machine Learning*, pages 685–693, 2014.

[75] Ali Dasdan. Experimental analysis of the fastest optimum cycle ratio and mean algorithms. *Transactions on Design Automation of Electronic Systems*, 9(4):385–418, 2004.

[76] Ali Dasdan, Sandy S Irani, and Rajesh K Gupta. Efficient algorithms for optimum cycle mean and optimum cost to time ratio problems. In *Design Automation Conference*, pages 37–42. IEEE, 1999.

[77] Jesús A De Loera and Edward D Kim. Combinatorics and geometry of transportation polytopes: an update. *Discrete Geometry and Algebraic Combinatorics*, 625:37–76, 2014.

[78] Michel Marie Deza and Elena Deza. Encyclopedia of distances. pages 1–583. Springer, 2009.

[79] Balázs Dezső, Alpár Jüttner, and Péter Kovács. LEMON–an open source C++ graph template library. *Electronic Notes in Theoretical Computer Science*, 264(5):23–45, 2011.

[80] Simone Di Marino, Augusto Gerolin, and Luca Nenna. Optimal transportation theory with repulsive costs. *Topological optimization and optimal transport*, 17:204–256, 2017.

[81] Rick Durrett. *Probability: theory and examples*. Cambridge University Press, 2010.

[82] Martin Dyer, Leslie Ann Goldberg, Catherine Greenhill, and Mark Jerrum. On the relative complexity of approximate counting problems. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 108–119. Springer, 2000.

[83] B Curtis Eaves, Alan J Hoffman, Uriel G Rothblum, and Hans Schneider. Line-sum-symmetric scalings of square nonnegative matrices. *Mathematical Programming*, pages 124–141, 1985.

[84] Herbert Edelsbrunner, Joseph O'Rourke, and Raimund Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM Journal on Computing*, 15(2):341–363, 1986.

[85] Tommy Elfving. On some methods for entropy maximization and matrix scaling. *Linear Algebra and its Applications*, 34:321–339, 1980.

[86] Filip Elvander, Isabel Haasler, Andreas Jakobsson, and Johan Karlsson. Multi-marginal optimal transport using partial information with applications in robust localization and sensor fusion. *Signal Processing*, 171:107474, 2020.

[87] Jon Feldman, Ryan O'Donnell, and Rocco A Servedio. Learning mixtures of product distributions over discrete domains. *SIAM Journal on Computing*, 37(5):1536–1564, 2008.

[88] Rémi Flamary, Marco Cuturi, Nicolas Courty, and Alain Rakotomamonjy. Wasserstein discriminant analysis. *Machine Learning*, 107(12):1923–1945, 2018.

[89] Shmuel Friedland. Tensor optimal transport, distance between sets of measures and tensor scaling. *Preprint at arXiv:2005.00945*, 2020.

[90] Michael R Garey and David S Johnson. The complexity of near-optimal graph coloring. *Journal of the ACM*, 23(1):43–49, 1976.

[91] Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. 1979.

[92] Dongdong Ge, Haoyue Wang, Zikai Xiong, and Yinyu Ye. Interior-point methods strike back: Solving the Wasserstein barycenter problem. In *Conference on Neural Information Processing Systems*, pages 6891–6902, 2019.

[93] Aude Genevay, Marco Cuturi, Gabriel Peyré, and Francis Bach. Stochastic optimization for large-scale optimal transport. *Conference on Neural Information Processing Systems*, 29, 2016.

[94] Loukas Georgiadis, Andrew V Goldberg, Robert E Tarjan, and Renato F Werneck. An experimental study of minimum mean cycle algorithms. In *Workshop on Algorithm Engineering and Experiments*, pages 1–13. SIAM, 2009.

[95] Ilya Gertsbakh and Yoseph Shpungin. *Network reliability and resilience*. Springer Science & Business Media, 2011.

[96] Andrew V Goldberg and Robert E Tarjan. Finding minimum-cost circulations by canceling negative cycles. *Journal of the ACM*, 36(4):873–886, 1989.

[97] Leslie Ann Goldberg and Mark Jerrum. The complexity of ferromagnetic Ising with local fields. *Combinatorics, Probability and Computing*, 16(1):43–61, 2007.

[98] Vincent Goulet, Christophe Dutang, Martin Maechler, David Firth, Marina Shapira, Michael Stadelmann, et al. expm: Matrix exponential. *R package version 0.99-0*, 2013.

[99] Alexandre Gramfort, Gabriel Peyré, and Marco Cuturi. Fast optimal transport averaging of neuroimaging data. In *International Conference on Information Processing in Medical Imaging*, pages 261–272. Springer, 2015.

[100] Kristen Grauman and Trevor Darrell. Fast contour matching using approximate earth mover's distance. In *Conference on Computer Vision and Pattern Recognition*, volume 1, pages 220–227. IEEE, 2004.

[101] Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[102] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2. Springer Science & Business Media, 2012.

[103] Jeremy Gunawardena. Cycle times and fixed points of min-max functions. In *Conference on Analysis and Optimization of Systems*, pages 266–272. Springer, 1994.

[104] Heng Guo and Mark Jerrum. A polynomial-time approximation algorithm for all-terminal network reliability. *SIAM Journal on Computing*, 48(3):964–978, 2019.

[105] Leonid Gurvits and Peter N Yianilos. The deflation-inflation method for certain semidefinite programming and maximum determinant completion problems. Technical report, NECI, 1998.

[106] Dan Gusfield and Leonard Pitt. A bounded approximation for the minimum cost 2-SAT problem. *Algorithmica*, 8(1-6):103–117, 1992.

[107] Isabel Haasler, Axel Ringh, Yongxin Chen, and Johan Karlsson. Estimating ensemble flows on a hidden Markov chain. In *Conference on Decision and Control*, pages 1331–1338. IEEE, 2019.

[108] Isabel Haasler, Axel Ringh, Yongxin Chen, and Johan Karlsson. Multi-marginal optimal transport with a tree-structured cost and the Schrödinger bridge problem. *SIAM Journal on Control and Optimization*, 59(4):2428–2453, 2021.

[109] Isabel Haasler, Axel Ringh, Yongxin Chen, and Johan Karlsson. Scalable computation of dynamic flow problems via multi-marginal graph-structured optimal transport. *Preprint at arXiv:2106.14485*, 2021.

[110] Isabel Haasler, Rahul Singh, Qinsheng Zhang, Johan Karlsson, and Yongxin Chen. Multi-marginal optimal transport and probabilistic graphical models. *IEEE Transactions on Information Theory*, 2021.

[111] Willem K Haneveld. Robustness against dependence in PERT: An application of duality and distributions with known marginals. In *Stochastic Programming*, pages 153–182. Springer, 1986.

[112] Mark Hartmann and James B Orlin. Finding minimum cost to time ratio cycles with small integral transit times. *Networks*, 23(6):567–574, 1993.

[113] Nicholas J Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM Journal on Matrix Analysis and Applications*, 26(4):1179–1193, 2005.

[114] Nhat Ho, Xuan Long Nguyen, Mikhail Yurochkin, Hung Hai Bui, Viet Huynh, and Dinh Phung. Multilevel clustering via Wasserstein means. In *International Conference on Machine Learning*, pages 1501–1509, 2017.

[115] Nhat Ho, Viet Huynh, Dinh Phung, and Michael Jordan. Probabilistic multilevel clustering via composite transportation distance. In *International Conference on Artificial Intelligence and Statistics*, pages 3149–3157, 2019.

[116] Daniel Hoske, Dimitar Lukarski, Henning Meyerhenke, and Michael Wegner. Engineering a combinatorial Laplacian solver: Lessons learned. *Algorithms*, 9(4):72, 2016.

[117] Ronald A Howard. *Dynamic programming and Markov processes*. John Wiley, 1960.

[118] Martin Idel. A review of matrix scaling and Sinkhorn's normal form for matrices and positive maps. *Preprint at arXiv:1609.06349*, 2016.

[119] Piotr Indyk and Nitin Thaper. Fast image retrieval via embeddings. In *International Workshop on Statistical and Computational Theories of Vision*, volume 2, page 5, 2003.

[120] Martin Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In *International Conference on Machine Learning*, pages 427–435, 2013.

[121] Hicham Janati, Marco Cuturi, and Alexandre Gramfort. Debiased Sinkhorn barycenters. In *International Conference on Machine Learning*, pages 4692–4701, 2020.

[122] Wittawat Jitkrittum, Zoltán Szabó, Kacper P Chwialkowski, and Arthur Gretton. Interpretable distribution features with maximum testing power. In *Conference on Neural Information Processing Systems*, pages 181–189, 2016.

[123] Anatoli Juditsky, Philippe Rigollet, and Alexandre B Tsybakov. Learning by mirror averaging. *The Annals of Statistics*, 36(5):2183–2206, 2008.

[124] Bahman Kalantari and Leonid Khachiyan. On the rate of convergence of deterministic and randomized RAS matrix scaling algorithms. *Operations Research Letters*, 14(5):237–244, 1993.

[125] Bahman Kalantari and Leonid Khachiyan. On the complexity of nonnegative-matrix scaling. *Linear Algebra and its Applications*, 240:87–103, 1996.

[126] Bahman Kalantari, Leonid Khachiyan, and A Shokoufandeh. On the complexity of matrix balancing. *SIAM Journal on Matrix Analysis and Applications*, 18(2):450–463, 1997.

[127] Bahman Kalantari, Isabella Lari, Federica Ricca, and Bruno Simeone. On the complexity of general matrix scaling and entropy minimization via the RAS algorithm. *Mathematical Programming*, 112(2):371–401, 2008.

[128] David R Karger. A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. *SIAM Review*, 43(3):499–522, 2001.

[129] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

[130] Richard M Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23(3):309–311, 1978.

[131] Richard M Karp and James B Orlin. Parametric shortest path algorithms with an application to cyclic staffing. *Discrete Applied Mathematics*, 3(1):37–45, 1981.

[132] Leonid G Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

[133] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

[134] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT Press, 2009.

[135] Vladik Kreinovich and Scott Ferson. Computing best-possible bounds for the distribution of a sum of several variables is NP-hard. *International Journal of Approximate Reasoning*, 41(3):331–342, 2006.

[136] Alexey Kroshnin, Nazarii Tupitsa, Darina Dvinskikh, Pavel Dvurechensky, Alexander Gasnikov, and Cesar Uribe. On the complexity of approximating Wasserstein barycenters. In *International Conference on Machine Learning*, pages 3530–3540, 2019.

[137] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956.

[138] Eugene L Lawler. Optimal cycles in doubly weighted directed linear graphs. In *International Symposium on the Theory of Graphs*, pages 209–232, 1966.

[139] Ching-Pei Lee and Stephen J Wright. Random permutations fix a worst case for cyclic coordinate descent. *IMA Journal of Numerical Analysis*, 39(3):1246–1275, 2019.

[140] Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in $O(\sqrt{\text{rank}})$ iterations and faster algorithms for maximum flow. In *Symposium on Foundations of Computer Science*, pages 424–433. IEEE, 2014.

[141] Yin Tat Lee, Aaron Sidford, and Santosh S Vempala. Efficient convex optimization with membership oracles. In *Conference On Learning Theory*, pages 1292–1294. PMLR, 2018.

[142] Christian Léonard. A survey of the Schr̈odinger problem and some of its connections with optimal transport. *Discrete and Continuous Dynamical Systems*, 34(4):1533–1574, 2014.

[143] Tianyi Lin, Nhat Ho, Xi Chen, Marco Cuturi, and Michael Jordan. Fixed-support Wasserstein barycenters: Computational hardness and fast algorithm. *Conference on Neural Information Processing Systems*, 33:5368–5380, 2020.

[144] Tianyi Lin, Nhat Ho, Marco Cuturi, and Michael I Jordan. On the complexity of approximating multimarginal optimal transport. *Journal of Machine Learning Research*, 23(65): 1–43, 2022.

[145] Nathan Linial, Alex Samorodnitsky, and Avi Wigderson. A deterministic strongly polynomial algorithm for matrix scaling and approximate permanents. In *Symposium on Theory of Computing*, pages 644–652, 1998.

[146] James R Lloyd and Zoubin Ghahramani. Statistical model criticism using kernel two sample tests. *Advances in Neural Information Processing Systems*, 28, 2015.

[147] Giulia Luise, Saverio Salzo, Massimiliano Pontil, and Carlo Ciliberto. Sinkhorn barycenters with free support via Frank-Wolfe algorithm. In *Conference on Neural Information Processing Systems*, pages 9322–9333, 2019.

[148] Van Sy Mai and Abdella Battou. Asynchronous distributed matrix balancing and application to suppressing epidemic. In *American Control Conference*, pages 2177–2182. IEEE, 2019.

[149] GD Makarov. Estimates for the distribution function of a sum of two random variables when the marginal distributions are fixed. *Theory of Probability and its Applications*, 26 (4):803–806, 1982.

[150] MathWorks. balance: diagonal scaling to improve eigenvalue accuracy. *https://www.mathworks.com/help/matlab/ref/balance.html*, .

[151] MathWorks. eig: eigenvalues and eigenvectors. *https://www.mathworks.com/help/matlab/ref/eig.html*, .

[152] Isaac Meilijson and Arthur Nádas. Convex majorization with an application to the length of critical paths. *Journal of Applied Probability*, 16(3):671–677, 1979.

[153] Vinit Kumar Mishra, Karthik Natarajan, Dhanesh Padmanabhan, Chung-Piaw Teo, and Xiaobo Li. On theoretical and empirical aspects of marginal distribution choice models. *Management Science*, 60(6):1511–1531, 2014.

[154] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: randomization and probabilistic techniques in algorithms and data analysis*. Cambridge University Press, 2017.

[155] Gaspard Monge. Mémoire sur la théorie des déblais et des remblais. *Histoire de l'Académie Royale des Sciences de Paris*, 1781.

[156] Edward F Moore and Claude E Shannon. Reliable circuits using less reliable relays. *Journal of the Franklin Institute*, 262(3):191–208, 1956.

[157] Jonas W Mueller and Tommi Jaakkola. Principal differences analysis: Interpretable characterization of differences between distributions. *Advances in Neural Information Processing Systems*, 28, 2015.

[158] Boris Muzellec, Richard Nock, Giorgio Patrini, and Frank Nielsen. Tsallis regularized optimal transport and ecological inference. In *AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[159] Arthur Nadas. Probabilistic PERT. *IBM Journal of Research and Development*, 23(3): 339–347, 1979.

[160] Karthik Natarajan, Miao Song, and Chung-Piaw Teo. Persistency model and its applications in choice modeling. *Management Science*, 55(3):453–469, 2009.

[161] Arkadi Nemirovski and Uriel Rothblum. On complexity of matrix scaling. *Linear Algebra and its Applications*, 302:435–460, 1999.

[162] Luca Nenna. *Numerical methods for multi-marginal optimal transportation*. PhD thesis, PSL Research University, 2016.

[163] Yurii Nesterov and Sebastian U Stich. Efficiency of the accelerated coordinate descent method on structured optimization problems. *SIAM Journal on Optimization*, 27(1): 110–123, 2017.

[164] James B Orlin. The complexity of dynamic languages and dynamic optimization problems. In *Symposium on the Theory of Computing*, pages 218–227. ACM, 1981.

[165] James B Orlin and Ravindra K Ahuja. New scaling algorithms for the assignment and minimum mean cycle problems. *Mathematical Programming*, 54(1-3):41–56, 1992.

[166] EE Osborne. On pre-conditioning of matrices. *Journal of the ACM*, 7(4):338–345, 1960.

[167] Rafail Ostrovsky, Yuval Rabani, and Arman Yousefi. Matrix balancing in l_p norms: bounding the convergence rate of Osborne's iteration. In *Symposium on Discrete Algorithms*, pages 154–169. SIAM, 2017.

[168] Rafail Ostrovsky, Yuval Rabani, and Arman Yousefi. Strictly balancing matrices in polynomial time using Osborne's iteration. In *International Colloquium on Automata, Languages and Programming*, 2018.

[169] Adam Ouorou and Philippe Mahey. A minimum mean cycle cancelling method for non-linear multicommodity flow problems. *European Journal of Operational Research*, 121(3): 532–548, 2000.

[170] Divya Padmanabhan, Selin Damla Ahipasaoglu, Arjun Ramachandra, and Karthik Natarajan. Extremal probability bounds in combinatorial optimization. *Preprint at arXiv:2109.01591*, 2021.

[171] Victor M Panaretos and Yoav Zemel. Amplitude and phase variation of point processes. *The Annals of Statistics*, 44(2):771–812, 2016.

[172] Victor M Panaretos and Yoav Zemel. Statistical aspects of Wasserstein distances. *Annual Review of Statistics and its Application*, 6:405–431, 2019.

[173] Christos H Papadimitriou. The largest subdeterminant of a matrix. *Bulletin of the Greek Mathematical Society*, 25(25):95–105, 1984.

[174] Christos H Papadimitriou and Tim Roughgarden. Computing correlated equilibria in multi-player games. *Journal of the ACM*, 55(3):1–29, 2008.

[175] Beresford N Parlett and Christian Reinsch. Balancing a matrix for calculation of eigenvalues and eigenvectors. *Numerische Mathematik*, 13(4):293–304, 1969.

[176] Brendan Pass. Multi-marginal optimal transport: theory and applications. *ESAIM: Mathematical Modelling and Numerical Analysis*, 49(6):1771–1790, 2015.

[177] Ofir Pele and Michael Werman. Fast and robust Earth Mover's Distances. In *International Conference on Computer Vision*, pages 460–467. IEEE, 2009.

[178] Gabriel Peyré and Marco Cuturi. Computational optimal transport. *Foundations and Trends in Machine Learning*, 11(5-6):355–607, 2019.

[179] François Pitié, Anil C Kokaram, and Rozenn Dahyot. Automated colour grading using colour distribution transfer. *Computer Vision and Image Understanding*, 107(1-2):123–137, 2007.

[180] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge University Press, 2007.

[181] J Scott Provan and Michael O Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal on Computing*, 12(4):777–788, 1983.

[182] Kent Quanrud. Approximating optimal transport with linear programs. In *Symposium on Simplicity in Algorithms*, 2018.

[183] Julien Rabin, Gabriel Peyré, Julie Delon, and Marc Bernot. Wasserstein barycenter and its application to texture mixing. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pages 435–446. Springer, 2011.

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

[185] James Renegar. A polynomial-time algorithm, based on Newton's method, for linear programming. *Mathematical Programming*, 40(1):59–93, 1988.

[186] Philippe Rigollet and Alexandre Tsybakov. Exponential screening and optimal rates of sparse estimation. *The Annals of Statistics*, 39(2):731–771, 2011.

[187] Philippe Rigollet and Alexandre B Tsybakov. Sparse estimation by exponential weighting. *Statistical Science*, 27(4):558–575, 2012.

[188] Uriel G Rothblum, Hans Schneider, and Michael H Schneider. Scaling matrices to prescribed row and column maxima. *SIAM Journal on Matrix Analysis and Applications*, 15 (1):1–14, 1994.

[189] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.

[190] Ludger Rüschendorf. Random variables with maximum sums. *Advances in Applied Probability*, pages 623–632, 1982.

[191] Ludger Rüschendorf and Ludger Uckelmann. On the n-coupling problem. *Journal of Multivariate Analysis*, 81(2):242–258, 2002.

[192] Roman Sandler and Michael Lindenbaum. Nonnegative matrix factorization with earth mover's distance metric for image analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1590–1602, 2011.

[193] Piotr Sankowski. Shortest paths in matrix multiplication time. In *European Symposium on Algorithms*, pages 770–778. Springer, 2005.

[194] Hans Schneider and Michael H Schneider. Max-balancing weighted directed graphs and matrix scaling. *Mathematics of Operations Research*, 16(1):208–222, 1991.

[195] Michael H Schneider and Stavros A Zenios. A comparative study of algorithms for matrix balancing. *Operations Research*, 38(3):439–455, 1990.

[196] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

[197] Erwin Schrödinger. Über die Umkehrung der Naturgesetze. *Angewandte Chemie*, 44(30): 636–636, 1931.

[198] Leonard J Schulman and Alistair Sinclair. Analysis of a classical matrix preconditioning algorithm. *Journal of the ACM*, 64(2):9, 2017.

[199] Zebang Shen, Zhenfu Wang, Alejandro Ribeiro, and Hamed Hassani. Sinkhorn barycenter via functional gradient descent. *Conference on Neural Information Processing Systems*, 33, 2020.

[200] Sameer Shirdhonkar and David W Jacobs. Approximate earth mover's distance in linear time. In *Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008.

[201] Rahul Singh, Isabel Haasler, Qinsheng Zhang, Johan Karlsson, and Yongxin Chen. Incremental inference of collective graphical models. *IEEE Control Systems Letters*, 2020.

[202] Sidak Pal Singh, Andreas Hug, Aymeric Dieuleveut, and Martin Jaggi. Context mover's distance & barycenters: Optimal transport of contexts for building representations. In *International Conference on Artificial Intelligence and Statistics*, pages 3437–3449, 2020.

[203] Richard Sinkhorn. Diagonal equivalence to matrices with prescribed row and column sums. *The American Mathematical Monthly*, 74(4):402–405, 1967.

[204] Maurice Sion. On general minimax theorems. *Pacific Journal of Mathematics*, 8(1): 171–176, 1958.

[205] Brian T Smith, James M. Boyle, BS Garbow, Y Ikebe, VC Klema, and CB Moler. *Matrix eigensystem routines - EISPACK guide*, volume 6. Springer, 2013.

[206] Justin Solomon, Fernando De Goes, Gabriel Peyré, Marco Cuturi, Adrian Butscher, Andy Nguyen, Tao Du, and Leonidas Guibas. Convolutional Wasserstein distances: Efficient optimal transportation on geometric domains. *ACM Transactions on Graphics*, 34(4): 1–11, 2015.

[207] Daniel A Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Symposium on Theory of computing*, pages 81–90, 2004.

[208] Sanvesh Srivastava, Cheng Li, and David B Dunson. Scalable Bayes via barycenter in Wasserstein space. *The Journal of Machine Learning Research*, 19(1):312–346, 2018.

[209] Matthew Staib, Sebastian Claici, Justin M Solomon, and Stefanie Jegelka. Parallel streaming Wasserstein barycenters. In *Conference on Neural Information Processing Systems*, pages 2647–2658, 2017.

[210] Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *Journal of the ACM*, 44(4):585–591, 1997.

[211] Marco Di Summa, Friedrich Eisenbrand, Yuri Faenza, and Carsten Moldenhauer. On largest volume simplices and sub-determinants. In *Symposium on Discrete Algorithms*, pages 315–323. SIAM, 2014.

[212] Ruoyu Sun and Yinyu Ye. Worst-case complexity of cyclic coordinate descent: $O(n^2)$ gap with randomized version. *Mathematical Programming*, pages 1–34, 2019.

[213] Ruoyu Sun, Zhi-Quan Luo, and Yinyu Ye. On the efficiency of random permutation for ADMM and coordinate descent. *Mathematics of Operations Research*, 45(1):233–271, 2020.

[214] Gábor J Székely and Maria L Rizzo. Testing for equal distributions in high dimension. *InterStat*, 5(16.10):1249–1272, 2004.

[215] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[216] Yee W Teh and Max Welling. The unified propagation and scaling algorithm. In *Conference on Neural Information Processing Systems*, pages 953–960, 2002.

[217] John A Tomlin. A new paradigm for ranking pages on the world wide web. In *International Conference on World Wide Web*, pages 350–355, 2003.

[218] Lloyd N Trefethen. *Approximation theory and approximation practice*, volume 164. SIAM, 2019.

[219] Alexandre B. Tsybakov. *Introduction to Nonparametric Estimation*. Springer Series in Statistics. Springer, 2009.

[220] Nazarii Tupitsa, Pavel Dvurechensky, Alexander Gasnikov, and César A Uribe. Multimarginal optimal transport by accelerated alternating minimization. In *Conference on Decision and Control*, pages 6132–6137. IEEE, 2020.

[221] Leslie G Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[222] Jan van den Brand, Yin-Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. *Preprint at arXiv:2009.01802*, 2020.

[223] Jan van den Brand, Yin-Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In *Symposium on Foundations of Computer Science*, pages 919–930. IEEE, 2020.

[224] Cédric Villani. *Topics in optimal transportation*. Number 58. American Mathematical Society, 2003.

[225] Cédric Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

[226] Martin J Wainwright and Michael I Jordan. Variational inference in graphical models: The view from the marginal polytope. In *Allerton Conference on Communication, Control, and Computing*, volume 41, pages 961–971, 2003.

[227] Martin J Wainwright and Michael Irwin Jordan. *Graphical models, exponential families, and variational inference*. Now Publishers Inc, 2008.

[228] Robert C Ward. Numerical computation of the matrix exponential with accuracy estimate. *SIAM Journal on Numerical Analysis*, 14(4):600–610, 1977.

[229] Gideon Weiss. Stochastic bounds on distributions of optimal value functions with applications to PERT, network flows and reliability. *Operations Research*, 34(4):595–605, 1986.

[230] Michael Werman, Shmuel Peleg, and Azriel Rosenfeld. A distance metric for multidimensional histograms. *Computer Vision, Graphics, and Image Processing*, 32(3):328–336, 1985.

[231] Virginia Vassilevska Williams. Multiplying matrices in $O(n^{2.373})$ time. *Available at* `http://theory.stanford.edu/~virgi/matrixmult-f.pdf`, 2014.

[232] Alan Geoffrey Wilson. The use of entropy maximising models, in the theory of trip distribution, mode split and route split. *Journal of Transport Economics and Policy*, pages 108–126, 1969.

[233] Stephen J Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1): 3–34, 2015.

[234] Jianbo Ye, Yanran Li, Zhaohui Wu, James Z Wang, Wenjie Li, and Jia Li. Determining gains acquired from word embedding quantitatively using discrete distribution clustering. In *Proceedings of the Association for Computational Linguistics*, pages 1847–1856, 2017.

[235] Jianbo Ye, Panruo Wu, James Z Wang, and Jia Li. Fast discrete distribution clustering using Wasserstein barycenter with sparse support. *IEEE Transactions on Signal Processing*, 65(9):2317–2332, 2017.

[236] VA Yemelichev, Michail M Kovalev, MK Dravtsov, and G Lawden. *Polytopes, graphs and optimisation*. Cambridge University Press, 1984.

[237] Neal E Young. Sequential and parallel algorithms for mixed packing and covering. In *Symposium on Foundations of Computer Science*, pages 538–546. IEEE, 2001.

[238] Neal E Young, Robert E Tarjan, and James B Orlin. Faster parametric shortest path and minimum-balance algorithms. *Networks*, 21(2):205–221, 1991.

[239] David B Yudin and Arkadi S Nemirovskii. Informational complexity and efficient methods for the solution of convex extremal problems. *Matekon*, 13(2):22–45, 1976.

[240] Raphael Yuster and Uri Zwick. Answering distance queries in directed graphs using fast matrix multiplication. In *Symposium on Foundations of Computer Science*, pages 389–396. IEEE, 2005.

[241] Eitan Zemel. Polynomial algorithms for estimating network reliability. *Networks*, 12(4): 439–452, 1982.

[242] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Symposium on the Theory of Computing*, pages 681–690. ACM, 2006.

[243] Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1-2):343–359, 1996.