A Study of Performances of Optimal Transport

Yihe Dong
MSR Redmond
yihe.dong@microsoft.com

Yu Gao
Georgia Tech*
ygao380@gatech.edu

Richard Peng
Goergia Tech*
rpeng@cc.gatech.edu

Ilya Razenshteyn
MSR Redmond
ilya.razenshteyn@microsoft.com

Saurabh Sawlani
Georgia Tech
sawlani@gatech.edu

Abstract

We investigate the problem of efficiently computing optimal transport (OT) distances, which is equivalent to the node-capacitated minimum cost maximum flow problem in a bipartite graph. We compare runtimes in computing OT distances on data from several domains, such as synthetic data of geometric shapes, embeddings of tokens in documents, and pixels in images. We show that in practice, combinatorial methods such as Network Simplex and augmenting path based algorithms can consistently outperform numerical matrix-scaling based methods such as Sinkhorn [Cut13] and Greenkhorn [AWR17], even in low accuracy regimes, with up to orders of magnitude speedups.

1 Introduction

The optimal transport (OT) problem can be defined formally as follows: given a supply distribution $r \in \mathbb{R}^n$ and a demand distribution $c \in \mathbb{R}^m$, find the optimal flow $X \in \mathbb{R}^{n\times m}$ that minimizes the overall transportation cost:

$$\min_X \langle C, X\rangle, \text{ s.t. } X\mathbf{1}_r = r \text{ and } X^T\mathbf{1}_c = c, \tag{1}$$

where C denotes the cost matrix, X the flow matrix, and $\mathbf{1}_r \in \mathbb{R}^m$ and $\mathbf{1}_c \in \mathbb{R}^n$ the all ones vectors. The Hitchcock-Koopmans transportation problem is a special case of the above – where r and c consist of only positive integers. Further, the case when r and c are all-ones vectors is known as the assignment problem.

*part of this work was done while at MSR Redmond

The OT problem has recently received much attention as an algorithmic subroutine [Cut13, BCC$^+$15, GCPB16, AWR17, DGK18, BJKS18, LHJ19, Qua19]. At first glance, Equation 1 can simply be solved exactly using an efficient linear program solver [LS14] in about $n^{2.5}$ time. However, to our knowledge, there have yet to be publically available packages that implement these more numerically driven methods. Numerical techniques based on entropic regularization [Cut13, BCC$^+$15, GCPB16, AWR17] have been shown to be scalable, both in terms of practice and their asymptotic runtimes. In particular, [Cut13] and [AWR17] are near-linear time additive approximation algorithms for the optimal transport problem.

On the other hand, many combinatorial techniques also give exact algorithms for the OT problem. The Kuhn-Munkres (aka Hungarian) algorithm [Kuh55, Kuh56, Mun57] solves the assignment problem exactly in $O(n^3)$ time. For the optimal transport problem, Gabow and Tarjan [GT91] gave an $O(n^{2.5}\log(nN))$ time algorithm, involving scaling costs to integers and solving the transportation problem. Here, N refers to the largest element in the cost matrix. Using the technique of scaling to integer demands and supplies, min-cost flow algorithms such as Network Simplex [AMO93] also provide exact algorithms for the optimal transport problem in $O(n^3 \log n \log(nN))$ time. Recently, Lahn *et. al.* [LMR19] modified the Gabow-Tarjan algorithm to give a faster approximation algorithm.

The computation of optimal transport distances has recently gained momentum across multiple problem domains, such as natural language processing [KSKW15] and image retrieval [RTG00, Cut13, SL11]. Additionally, computing optimal transport distances between probability distributions has direct applicability to tasks involving unsupervised learning [ACB17, BGKL17], semi-supervised learning [SRGB14], clustering [HNY$^+$17], and several other applications [KPT$^+$17]. With this in mind, it is essential to develop a good understanding of the efficiency and effectiveness of the various methods across different types and sizes of data. As far as we are aware, there have not been comprehensive studies that rigorously cross-examine the accuracies and performances of the classical and more recent methods to solve the OT problem. We seek to fill that void in this study, for a variety of relevant datasets.

In this paper, we compare various exact and approximate OT algorithms ranging from combinatorial methods to numerical ones based on matrix rescaling. We perform comparisions in BLAS-optimized verisons of C++ when possible, but also include the original MATLAB versions of the codes when relevant.

We also observe numerical instabilities of the floating-point based implementations of matrix rescaling techniques, and measure their iteration counts using simulations that track the logarithms of the values, and implicitly performs the rescalings in a numerically stable manner.

2 Optimal Transport Algorithms

In this section, we outline the efficient OT algorithms evaluated in our experiments.

2.1 Network Simplex

Network simplex [AMO93] is a specialized version of the simplex algorithm that solves the min-cost max-flow problem on graphs. Formulating the min-cost max-flow problem as a linear program, each edge corresponds to a variable. Similar to the set of basic variables in the original simplex algorithm, the network simplex algorithm maintains a spanning tree of edges.

The following observation explains why it is sufficient to maintain a spanning tree. Consider some flow on a graph. We say an edge is nonbasic if the amount of flow along it is not equal to 0 nor its capacity. If there exists an undirected cycle formed by nonbasic edges, we can send flow along the cycle clockwise or counterclockwise to decrease the cost until some edge becomes basic. Thus, we observe that in an optimal solution, the nonbasic edges always form a spanning tree (or forest).

The network simplex algorithm restricts its search amongst such spanning trees. In each step, the algorithm finds a basic edge (the entering edge) and the path in the spanning tree connecting the two endpoints of that edge. It then tries to push flow along the cycle formed by the edge and the path in a direction that decreases the total cost. The entering edge generally becomes nonbasic and is then included in the spanning tree, while another edge in the tree path (the leaving edge) becomes basic and leaves the tree. Like the simplex algorithm, network simplex can be applied to find an initial spanning tree.

2.2 Matrix Scaling

Matrix scaling methods are developed to derive approximate solutions to the optimal transport linear problem. Here we discuss Sinkhorn [Cut13] and Greenkhorn [AWR17].

2.2.1 Sinkhorn

Cuturi [Cut13] opened up a new line of work that uses matrix scaling to find approximate solutions to the optimal transport problem, by finding the Sinkhorn distance $d_C^\eta(r,c) := \langle X^\eta, C\rangle$, – a solution to OT problem with entropy regularization:

$$X^\eta = \text{argmin}_X \langle X, C\rangle - \frac{1}{\eta} h(X)$$

where $\eta > 0$, and the flow X is optimized over the feasible region where $X\mathbf{1}_r = r$ and $X^T\mathbf{1}_c = c$.

Adding the entropy term to the objective function enforces a structure to the solution X^η: $X^\eta = D_1 e^{-\eta C} D_2$, where $e^{-\eta C} \in \mathbb{R}^{n\times m}$ is the entrywise exponential of $-\eta C$, and $D_1 \in \mathbb{R}^{n\times n}$ and $D_2 \in \mathbb{R}^{m\times m}$ are diagonal matrices.

Therefore, X^η and hence the Sinkhorn distance d_k can be computed by iterative matrix scaling on the matrix $e^{-\eta C}$, whereby all rows and all columns are alternately scaled to minimize the residue, i.e. the difference between current row or column sums and the target row or column sums.

After the residue $\|r - X\mathbf{1}_r\|_1 + \left\|c - X^T\mathbf{1}_c\right\|_1$ drops below a preset threshold ϵ, the flow X is rounded so that the solution lies in the feasible region, i.e. $X\mathbf{1}_r = r$ and $X^T\mathbf{1}_c = c$. As decribed in a later section, this rounding does not necessasrily bring the solution closer to the optimum.

2.2.2 Greenkhorn

Greenkhorn [AWR17] is a greedy adaptation of Sinkhorn, instead of scaling all rows and all columns simultaneously, Greenkhorn scales one row or one column where the gain is the most, i.e. where the discrepancy between the current row or column sum and the target row or column sum is the largest. Greenkhorn brings the runtime to $O(n^2 \log(n)(N/\delta)^3)$.

[AWR17] shows that Greenkhorn converges with fewer row/column updates than Sinkhorn, one Greenkhorn update is more expensive than one Sinkhorn update, as Sinkhorn updates all rows or all columns simultaneously, whereas Greenkhorn needs to keep track of which row and which column to update next.

Rounding

Both Sinkhorn and Greenkhorn output not only an approximate objective value to the OT linear program, but also a feasible solution for the same. This is achieved by a supplementary *rounding* procedure [AWR17], which first scales all routed flows down to at most the demands/supplies, and then distributes the leftover demands proportional to the row- and column-wise flow errors. Since this routing of the leftover demands is not done optimally, this can cause a loss in the objective at the cost of obtaining a feasible solution.

Without rounding, since $\|r - X\mathbf{1}_r\|_1 + \left\|c - X^T\mathbf{1}_c\right\|_1 \leq \epsilon$, there exists an X' satisfying $X'\mathbf{1}_r = r$ and $X'^T\mathbf{1}_c = c$ such that $|\langle C, X'\rangle - \langle C, X\rangle| \leq \epsilon \|C\|_\infty$, which is negligible compared to the total transport cost in most datasets.

2.3 Combinatorial Optimal Transport

Combinatorial methods such as the Hungarian algorithm can be used to solve the min cost matching problem on a bipartite graph, the fastest implementation of which converges in $O(n(m + n\log n))$ time, where n and m are the number of vertices and edges, respectively. Gabow [Gab85] developed a scaling algorithm for min cost matching, where the costs are doubled $\log(N)$ times, here N is the largest magnitude of a cost. This algorithm converges in $O(mn^{3/4}\log(N)$ time.

Gabow and Tarjan [GT91] improved this result with a different scaling algorithm, wherein the costs are doubled $\log(nN)$ times instead of $\log(N)$ times, using an additional $\log(n)$ scalings to ensure that the last approximate optimum is exact. This scaling mechanism allows the algorithm to simulate both the Hungarian algorithm and the Hopcraft-Karp algorithm simultaneously, in the low-cost regime. The increased number of scalings improves the time to convergence to $O(m\sqrt{n}\log(nN))$.

More recently, Lahn *et. al.* [LMR19] adapted the Gabow-Tarjan scaling algorithm to bound the runtime for approximate solutions with additive errors by $O(n^2(N/\delta) + n(N/\delta)^2)$, where δ is the additive error. We benchmark Lahn *et. al.* on the datasets described below.

3 Experiments

3.1 Experimental Setup

All experiments are performed on an Intel (R) Core i7-2600 CPU @ 3.40GHz $\times$ 8 CPU, with Ubuntu 18.04.3 LTS.

In C++ implementations of Sinkhorn and Greenkhorn, we use the highly optimized OpenBLAS [XKS$^+$19] for Basic Linear Algebra Subprograms (BLAS).

For Greenkhorn, note that we keep track of the changing row and column sums per row/column update, instead of computing all the row and column sums per update to determine which row or column to update.

3.2 Datasets

We consider four settings to compute the OT problem: between integral points filling geometric regions in the shape of a disk and a square, between word embeddings of texts, between MNIST images, and between CIFAR images. We release an expanded version of these datasets for public benchmarking.

CircleSquare: CircleSquare describes a min cost matching problem between integral points. Given a pair of square region and circular region in $\mathbb{R}^2$ that share the same center and contain the same number of integral points, CircleSquare asks for a matching between the two sets of integral points such that the total Euclidean distance between matched points is minimum.

NLP: We create distributions from disjoint portions of the novel *The Count of Monte Cristo*. Each sample contains 900 lines of text, with varying lengths. The text is tokenized with AllenNLP [GGN$^+$18] and stopwords are removed. The resulting tokens are embedded into $\mathbb{R}^{100}$ with 100-dimensional GloVe [PSM14] word embeddings, creating a $N \times 100$ matrix, where N is the number of tokens in the distribution. The capacities are integral vectors of size the number of tokens, and the cost matrix contains pairwise Euclidean distances between the token embeddings in a pair of distributions.

MNIST: The MNIST [LC10] dataset contains black and white images of size 28×28 pixels, thus each image gives a capacity distribution of length-784 (28^2), where capacity values are determined by pixel intensities. Since this vector is a priori sparse, we keep only the nonzero coordinates for efficiency. To compute the optimal transport between two distributions, the cost matrix consists of the pairwise Euclidean distances between the (x, y) coordinates in each distribution. The supply and demand capacity vectors are appropriately normalized and rounded such that the supply and demand sum to the same.

CIFAR: The CIFAR[2] [Kri09] dataset contains color images of size 32×32 pixels. To construct a distribution from an image, we represent each pixel as a (x, y, r, g, b) tuple, scale each tuple such that the range of each coordinate is $[0, 1]$. Each image is thus represented as a 1024×5 matrix, and the cost matrix between two images is computed as the Euclidean distance matrix between the two 1024×5 representations. The capacities are integral vectors of size 1024, appropriately normalized and rounded such that the supply and demand capacities sum to the same.

[2]https://www.cs.toronto.edu/ kriz/cifar.html

Table 1 contains metadata about the various datasets that we test the algorithms on.

Dataset	n	m	supply/demand range	total demand	cost range
CS100	100	100	1 - 1	100	1365 - 1382653
CS900	900	900	1 - 1	900	445 - 1554383
CS2500	2500	2500	1 - 1	2500	0 - 1571480
CS4900	4900	4900	1 - 1	4900	8 - 1579922
mnist0	116	169	54 - 13818	999929	0 - 234
mnist1	165	172	144 - 9194	999961	0 - 241
mnist2	64	136	218 - 25833	999961	0 - 226
mnist3	193	168	68 - 8671	999933	0 - 209
mnist4	120	75	208 - 15741	999945	0 - 204
mnist5	82	137	72 - 18332	999950	0 - 219
mnist6	135	148	43 - 12037	999926	0 - 219
mnist7	129	134	88 - 12107	999948	0 - 262
mnist8	174	210	65 - 8297	999920	0 - 220
mnist9	176	106	95 - 15903	999942	0 - 233
NLP1	1389	1638	9 - 7270	81400	0 - 11577502
NLP2	1860	1614	10 - 7020	89030	0 - 10839515
NLP3	1556	1555	9 - 6760	80210	0 - 11512870
NLP4	1788	1757	10 - 6390	88190	0 - 11480738
NLP5	1705	1775	8 - 6729	81820	0 - 11401853
CIFAR1	1024	1024	1 - 1	1024	5 - 610
CIFAR2	1024	1024	1 - 1	1024	9 - 611
CIFAR3	1024	1024	1 - 1	1024	10 - 619
CIFAR4	1024	1024	1 - 1	1024	5 - 568
CIFAR5	1024	1024	1 - 1	1024	3 - 600
CIFAR6	1024	1024	1 - 1	1024	5 - 654

Table 1: Details of datasets used in our experiments.

4 Results

4.1 Algorithm configurations

In Table 2, we compare the efficiency of a variety of techniques used commonly to solve OT. The following is a brief description of the parameters used for each dataset in this comparison.

Lemon NS This is an exact solution using the Network Simplex algorithm. This is implemented in C++ by using the Network Simplex module from Lemon[3] [DJK11] as a blackbox which computes minimum cost flows.

Sinkhorn The approximation factor and runtime of the Sinkhorn algorithm depend on our choice of η (the regularization parameter) and ϵ (the maximum allowed leftover demand and supply).

We pick the smallest possible η for which the algorithm converges to a solution which is 110% of the optimal routing cost (computed by Network Simplex) while keeping $\epsilon = 1$. (Since the original distributions in the NLP datasets are not integral, we scaled up and rounded the demand/supply vectors. ϵ is adjusted to 300 for those datasets.) We document the values of η and ϵ in Table 2. We test our highly-optimized implementation of the algorithm using C++, as well as the MATLAB implementation[4] of Sinkhorn by [AWR17], On large sized graphs, it runs into precision issues on both MATLAB and C++ when using sufficiently large values of η, which are needed for accurate solutions (leading to very small exponentials). These graphs are marked with "-"s in their corresponding entries of Table 2.

[3]https://lemon.cs.elte.hu/trac/lemon

[4]https://github.com/JasonAltschuler/OptimalTransportNIPS17

Greenkhorn Similar to Sinkhorn, we fix $\epsilon = 1$ ($\epsilon = 300$ for NLP datasets) and pick the smallest possible regularization parameter η for which we obtain a 1.1-approximate solution. Since the regularization parameter η and threshold ϵ do not vary between the obtained approximation factor in both Sinkhorn and Greenkhorn, we do not document ithem for Greenkhorn separately. On a majority of mid-sized or larger graphs, around $\sim$1000 nodes in the supply and demand distributions **YD: can someone pls verify this as well?**, the algorithm runs into precision issues on MATLAB. Hence, we only show the runtimes for our implementation in C++. The graphs on which the C++ code runs into precision issues are marked with "-"s in their corresponding entries of Table 2.

CombOT This is the combinatorial dual-adjustment based algorithm for optimal transport from [LMR19]. Although a value δ is input as allowed error, this value is only a theoretical upper bound on the error. In practice, the error is much smaller, and the algorithm even gives exact answers for many datasets despite setting non-zero δ. In our experiments, we set δ to be the biggest possible number (equivalently, fastest possible runtime) which gives an answer close to exact (at most 0.5% error).

For testing, we use the MATLAB implementation[5] by the authors of [LMR19].

Graph	Lemon NS (exact)	Sinkhorn (1.1-approx)				Greenkhorn (1.1-approx)	CombOT (≈exact*)
		MATLAB	C++	η	ϵ		
CS100	0.002s	0.078s	0.359s	368.0	1	0.19s	0.009s
CS900	0.094s	6.7s	30.93s	2792	1	26.2s	1.72s
CS2500	1.418s	271s	1860s	5371	1	-	22.65s
CS4900	7.973s	-	-	-	-	-	270.5s
mnist0	0.003s	0.065s	0.739s	48.93	1	0.38s	0.18s
mnist1	0.002s	0.099s	1.371s	59.18	1	0.56s	0.21s
mnist2	0.001s	0.021s	0.016s	51.86	1	0.10s	0.07s
mnist3	0.002s	0.211s	2.656s	74.41	1	1.68s	0.23s
mnist4	0.001s	0.013s	0.012s	36.04	1	0.10s	0.07s
mnist5	0.001s	0.025s	0.194s	34.28	1	0.08s	0.11s
mnist6	0.002s	0.128s	1.378s	67.38	1	0.52s	0.09s
mnist7	0.001s	0.04s	0.382s	48.93	1	0.11s	0.11s
mnist8	0.003s	0.049s	0.503s	37.79	1	0.33s	0.41s
mnist9	0.002s	0.077s	1.02s	63.87	1	0.37s	0.11s
NLP1	0.471s	8.78s	24.12s	42.58	300	9.25s	2.07s
NLP2	0.604s	39.1s	28.76s	41.02	300	9.81s	3.03s
NLP3	0.465s	34.2s	30.66s	41.80	300	7.84s	2.46s
NLP4	0.583s	27.4s	24.03s	43.36	300	9.25s	3.27s
NLP5	0.597s	14.9s	25.62s	41.80	300	9.80s	2.76s
CIFAR1	0.172s	0.86s	16.81s	99.22	1	15.1s	2.62s
CIFAR2	0.143s	0.99s	18.75s	94.53	1	13.5s	2.31s
CIFAR3	0.16s	0.51s	11.03s	57.42	1	7.75s	3.25s
CIFAR4	0.173s	0.49s	8.865	85.16	1	7.38s	3.47s
CIFAR5	0.207s	0.36s	5.394s	73.83	1	4.50s	2.18s
CIFAR6	0.185s	0.26s	6.078s	61.33	1	5.66s	4.08s

Table 2: Comparison of runtimes between various methods for OT. *CompOT allows at most 0.5% error.

From Table 2, we see that Network Simplex consistently performs much faster than the rest of the algorithms on all datasets, many times achieving speedups of at least an order of magnitude. Particularly, it is faster than Greenkhorn because Greenkhorn computes the residue after each update while Sinkhorn does this after every n (or m) updates. Additionally, despite allowing a 10% margin of error to Sinkhorn and Greenkhorn, they are almost always slower than CombOT. The cells in the

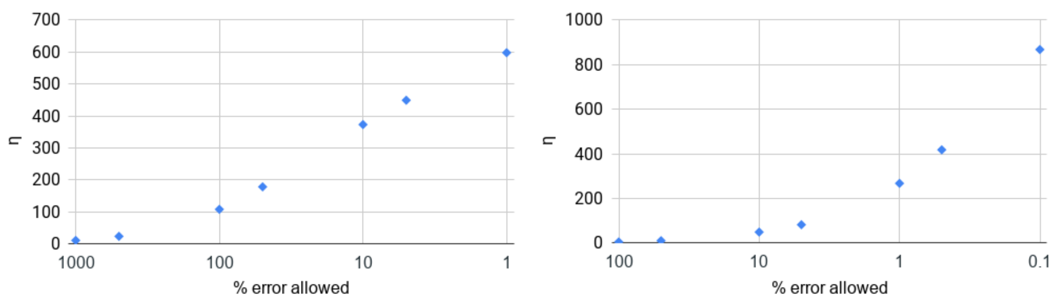
[5] https://github.com/nathaniellahn/CombinatorialOptimalTransport

table which are blank are either due to the algorithm taking too much time to run, or due to precision issues.

4.2 Effect of regularization in matrix scaling techniques

In both matrix scaling methods - Greenkhorn and Sinkhorn, the accuracy of the algorithm depends crucially on the choice of η. As we increase the value of η, the algorithm requires more iterations to converge and naturally also approaches a better approximation factor.

Figure 1 shows the relation between the accuracy of matrix scaling methods with increasing values of η. Note that this value of η is selected assuming that the maximum value of the cost matrix is 1. This allows for a fair comparison between different datasets. The matrices have been scaled appropriately to accommodate this.

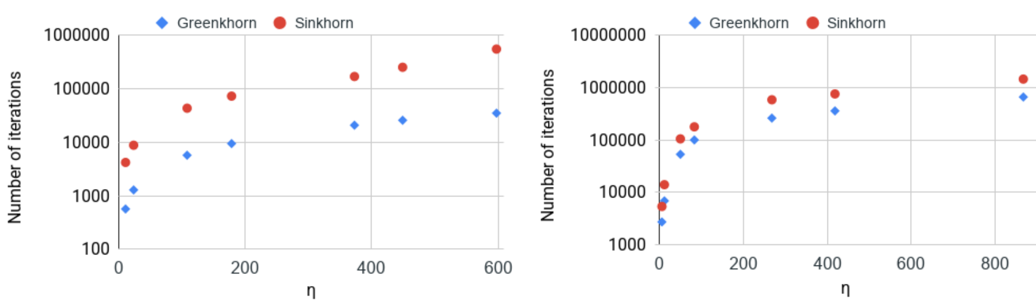


(a) CS100 (b) mnist0

Figure 1: Accuracy analysis of Sinkhorn and Greenkhorn

As can be seen from Figure 1, significantly high values of the regularization parameter η are required for the algorithms to reach an approximation arbitrarily close to optimal. This creates many issues in practice because higher values of η require higher float precision from the compiler (due to very small exponentials appearing).

Figure 2 shows the number of iterations needed for the algorithm to converge as a function of the desired approximation factor (or equivalently, η).



(a) CS100 (b) mnist0

Figure 2: Convergence analysis of Sinkhorn and Greenkhorn

We see that the growth in number of iterations is much higher than linear, and this can cause algorithms to be very slow when trying to get a solution with high accuracy.

5 Conclusion

An important takeaway from our findings is that, despite the design of asymptotically faster algorithms for optimal transport, comniatorial algorithms such as Network Simplex and the adaptation of Gabow and Tarjan [LMR19] are able to exploit the structure of the matrix and significantly outperform the newer matrix rescaling based algorithms. Recall that this fact is despite a 10% error allowance for the latter. This is not always an artifact of the algorithms themselves, but is also partly due to the former being highly tuned and perfected for practice over the years.

Secondly, while the claim from [AWR17] that Greenkhorn uses fewer iterations to reach the same objective as Sinkhorn is indeed true, Greenkhorn still performs significantly slower when evaluated via the actual time taken by the algorithm. The additional cost at each iteration of finding the best row/column proves to be rather high, and optimizing this would go a long way in obtaining runtimes better than Sinkhorn.

References

[ACB17] Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 214–223, 2017.

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows - theory, algorithms and applications*. Prentice Hall, 1993.

[AWR17] Jason Altschuler, Jonathan Weed, and Philippe Rigollet. Near-linear time approximation algorithms for optimal transport via sinkhorn iteration. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 1964–1974, 2017.

[BCC+15] Jean-David Benamou, Guillaume Carlier, Marco Cuturi, Luca Nenna, and Gabriel Peyré. Iterative bregman projections for regularized transportation problems. *SIAM J. Scientific Computing*, 37(2), 2015.

[BGKL17] J. Bigot, R. Gouet, T. Klein, and A. López. Geodesic PCA in the Wasserstein space by convex PCA. *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques, vol. 53, issue 1, pp. 1-26*, 53:1–26, February 2017.

[BJKS18] Jose H. Blanchet, Arun Jambulapati, Carson Kent, and Aaron Sidford. Towards optimal running times for optimal transport. *CoRR*, abs/1810.07717, 2018.

[Cut13] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 2292–2300, 2013.

[DGK18] Pavel E. Dvurechensky, Alexander Gasnikov, and Alexey Kroshnin. Computational optimal transport: Complexity by accelerated gradient descent is better than by sinkhorn's algorithm. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 1366–1375, 2018.

[DJK11] Balázs Dezs, Alpár Jüttner, and Péter Kovács. Lemon - an open source c++ graph template library. *Electron. Notes Theor. Comput. Sci.*, 264(5):23–45, July 2011.

[Gab85] Harold Gabow. Scaling algorithms for network problems. *Journal Of Computer AND System Sciences*, 31, 1985.

[GCPB16] Aude Genevay, Marco Cuturi, Gabriel Peyré, and Francis R. Bach. Stochastic optimization for large-scale optimal transport. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 3432–3440, 2016.

[GGN+18] Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew E. Peters, Michael Schmitz, and Luke Zettlemoyer. Allennlp: A deep semantic natural language processing platform. *CoRR*, abs/1803.07640, 2018.

[GT91] Harold N. Gabow and Robert E. Tarjan. Faster scaling algorithms for general graph matching problems. *J. ACM*, 38(4):815–853, October 1991.

[HNY+17] Nhat Ho, XuanLong Nguyen, Mikhail Yurochkin, Hung Hai Bui, Viet Huynh, and Dinh Q. Phung. Multilevel clustering via wasserstein means. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1501–1509, 2017.

[KPT+17] Soheil Kolouri, Se Rim Park, Matthew Thorpe, Dejan Slepcev, and Gustavo K. Rohde. Optimal mass transport: Signal processing and machine-learning applications. *IEEE Signal Process. Mag.*, 34(4):43–59, 2017.

[Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Canadian Institute for Advanced Research, 2009.

[KSKW15] Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, and Kilian Q. Weinberger. From word embeddings to document distances. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 957–966, 2015.

[Kuh55] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[Kuh56] H. W. Kuhn. Variants of the hungarian method for assignment problems. *Naval Research Logistics Quarterly*, 3(4):253–258, December 1956.

[LC10] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/, 2010.

[LHJ19] Tianyi Lin, Nhat Ho, and Michael I. Jordan. On efficient optimal transport: An analysis of greedy and accelerated mirror descent algorithms. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 3982–3991, 2019.

[LMR19] Nathaniel Lahn, Deepika Mulchandani, and Sharath Raghvendra. A graph theoretic additive approximation of optimal transport. *CoRR*, abs/1905.11830, 2019.

[LS14] Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in õ(vrank) iterations and faster algorithms for maximum flow. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 424–433, 2014.

[Mun57] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[Qua19] Kent Quanrud. Approximating optimal transport with linear programs. In *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019, January 8-9, 2019 - San Diego, CA, USA*, pages 6:1–6:9, 2019.

[RTG00] 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.

[SL11] Roman Sandler and Michael Lindenbaum. Nonnegative matrix factorization with earth mover's distance metric for image analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(8):1590–1602, 2011.

[SRGB14] Justin Solomon, Raif M. Rustamov, Leonidas J. Guibas, and Adrian Butscher. Wasserstein propagation for semi-supervised learning. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 306–314, 2014.

[XKS+19] Zhang Xianyi, Martin Kroeker, Werner Saar, Wang Qian, Zaheer Chothia, Chen Shaohu, and Luo Wen. Openblas: An optimized blas library. `https://www.openblas.net/`, 2019.