Algorithms for the Geometric Transportation Problem

Patrick Emami

August 16, 2018

Abstract

Discrete optimal transport has rich geometric interpretations that can be exploited to develop fast algorithms for a variety of problems. In this survey, I present the recent literature on network flow algorithms that use computational geometry to find exact and approximate transportation plans in subcubic running times. Additionally, I introduce the theory of entropy-regularized discrete optimal transport to highlight how the transportation problem is being used in fields such as machine learning. An outline for a streaming version of a fast, planar transportation algorithm is provided in the conclusions.

Contents

1 Introduction

The optimal transport problem, originally proposed by Monge in 1781 [1], asks how to most efficiently move a supply of an object (e.g., a pile of dirt) to another location (e.g., a hole). In our society, optimal transportation problems concerning people, commodities, and information are solved at a global scale on a daily basis. Indeed, it is practical applications that have driven much of the theoretical progress in this field. During World War II, Hitchcock, Koopmans, and Kantorovich made significant progress on the problem by formalizing it as a linear program. The contributions of Villani [2] has helped to popularize

optimal transport, especially since he won the Fields medal in 2010. Optimal transport has been used for many applications, and has recently become useful in many areas of computer vison/graphics and machine learning. In machine learning, the transportation distance, or Earth Mover's Distance (EMD), offers a powerful, geometrically-motivated metric for probability measures. However, most of the interesting uses for the transportation problem in this area are either high-dimensional or require running with millions or billions of data points. Hence, even though many polynomial-time algorithms exist for solving linear programs, there is still ongoing research on developing faster algorithms for the transportation problem that scale in the era of big data. Improvements in our capability to efficiently solve linear programs usually results in faster algorithms for transportation, matching, assignment, and network flow problems (see, e.g., [3]). I refer the reader to §4 of a recent survey on various geometric optimization algorithms by Agarwal et. al. [4] for another in-depth review of the transportation problem.

In the remainder of this section, I establish the setting for the geometric transportation problem. Then, the next section will present various algorithms for this problem based on network flow and computational geometry. This will be followed by a brief motivation of the entropy-regularized version of the discrete transportation problem. I conclude with a discussion on future research directions.

We define sets $A, B \subset \mathbb{R}^d$, $|A| = |B| = n$, and let $a \in A$ have demand $d_a \in \mathbb{Z}^+$ and $b \in B$ have supply $s_b \in \mathbb{Z}^+$. I assume that $\sum_{a \in A} d_a = \sum_{b \in B} s_b = U$ and that a "ground distance" $d(\cdot, \cdot)$ has been specified. Let P be the $n \times n$ transportation matrix, also known as the transportation plan, whose entries $p_{ba} \geq 0$ contain the counts of how much of each supply s_b is sent to satisfy demand d_a. Each row sums to some fixed r_b, and each column sums to some fixed c_a; the sum of the r_b's and the sum of the c_a's is U. The *Hitchcock-Koopmans* transportation problem, which seeks a minimum cost transportation plan, can be written as the following linear program:

$$\begin{aligned} C(P) = \min_P & \sum_{a \in A} \sum_{b \in B} p_{ba} d(a, b), \\ \text{s.t.} \quad & \sum_{a \in A} p_{ba} = r_b, \\ & \sum_{b \in B} p_{ba} = c_a, \\ & p_{ba} \geq 0. \end{aligned} \tag{1}$$

where $C(P)$ is the optimal transportation distance, or EMD. Throughout most of this paper, I use terminology from network flow algorithms, so at this point it will be helpful to highlight the connection between geometric transportation and network flow. The key relationship is between the transportation plan and the flow through the network from sources $b \in B$ to sinks $a \in A$. Equation 1 can be re-written by replacing p_{ba} with f_{ba}, where f_{ba} is the flow from b to a.

Then the problem becomes one of finding the minimum cost flow. The flows are *uncapacitated*, and the amount of flow originating from each source is given by s_b and the amount that each sink accepts is d_a. In the network flow formulation of the transportation problem, it is typically assumed that the bipartite graph formed by A and B is complete. While the two problems have many similarities, one of the difficulties that the transportation problem has that the matching problem does not is that flow can enter or exit a node by multiple edges. I discuss various approaches for solving the geometric transportation problem that make use of matching or network flow algorithms as subroutines.

Before I begin discussing the computational complexity of algorithms for the transportation problem, it is useful to discuss some baselines for comparison. In general, the transportation problem can be solved with generic algorithms such as the Hungarian method in $O(|V||E|)$ time; for complete bipartite graphs, this is equivalent to $O(n^3)$. Other generic approaches can achieve slightly faster bounds by exploiting characteristics of the inputs, e.g., Gabow and Tarjan's well-known scaling algorithm [3] that can achieve $O((\min\{\sqrt{U}, n\}n^2 + U \log U) \log nN)$, where N is the magnitude of the largest supply or demand of any node. Major breakthroughs in improving the efficiency of min-cost flow algorithms generally correspond to direct improvements to these algorithms. Orlin's algorithm [5] solves uncapacitated min-cost flow in $O(n \log n(n^2 + n \log n))$ and is occasionally used as a subroutine in algorithms for the transportation problem. Recently, Lee and Sidford proposed a novel interior-point method for solving linear programs that resulted in a $\tilde{O}(n^{2.5})$ algorithm for min-cost flow [6].

2 Geometric Algorithms

In this section, I present a variety of algorithms for solving the geometric transportation problem based on network flow. The algorithms I consider make use of the geometric aspects of the problem to gain extra efficiency. I divide this section into two parts; the first part is concerned with exact algorithms, and the second part focuses on approximation algorithms.

2.1 Exact Geometric Algorithms

The first exact algorithm I consider extends a fast geometric algorithm for weighted bipartite matching from Vaidya [7] to the transportation setting. The assumptions in [8] about the setting are that $A, B \subset \mathbb{R}^2$ and the row and column sums r_b, c_a of any valid transportation matrix are nonnegative integers. The main contribution of this paper is that, when the ground distance is the l_1, l_2, or l_∞ norm, the algorithm runs in $O(n^{2.5} \log n \log N)$.

The algorithm presented in [8] uses the *dual* of the linear program formulation from Eq. 1. A primal-dual algorithm for the transportation problem associates a dual variable with each node and a nonnegative slack value with each edge [3, 8]. Typically, primal-dual algorithms use a scaling of supplies and demands. At each scale, a max-flow subproblem is solved that requires finding $O(n)$ augmenting

paths. Next, I summarize how [8] uses data structures from computational geometry to find each augmenting path in $O(n^{1.5} \log n)$ time, instead of the naive $O(n^3)$ run time.

Without going into much detail on the algorithm for finding a max-flow in the primal-dual method, I highlight that it involves identifying a set of admissible edges used to define a residual graph. The admissible edges are edges that satisfy certain constraints, which are defined as functions of the dual variables and the ground distance. Augmenting paths in this residual graph are used to find a max-flow. Naively, this can be accomplished in $O(n^2 A)$ time, where A is the final flow amount—A is typically $\geq n$.

In each max-flow subproblem, the quantity

$$\delta = \min_{a \in A, b \in B} \{d(a, b) - \alpha_a - \beta_b\}, \tag{2}$$

is used to identify admissible edges, where α_a is the dual variable associated with node a and β_b is the dual variable associated with node b. When the ground distance is the l_2 norm, they suggest to use a weighted Voronoi diagram (WVD) [9], and when the distance is the l_1 or l_∞, a range tree (RT) can be used. These data structures allow for δ to be found without having to check all edge costs. Simply, a WVD is a Voronoi diagram with a weight associated with each point in the plane. If all of the weights are equal, the resulting WVD is equivalent to the standard Voronoi diagram. The WVD can be used to compute Eq. 2 as follows. Note that for a set of points Q, the WVD divides the plane into $|Q|$ regions. The Voronoi cells are

$$\text{Vor}(q) = \{x \in \mathbb{R}^2 : \texttt{nearest}[x, Q] = q\},$$

where I define $\texttt{nearest}[z, Q], z \in \mathbb{R}^2$ as the point q^* such that

$$d(z, q^*) - w(q^*) = \min_{q \in Q} \{d(z, q) - w(q)\}.$$

Then I can solve Eq. 2 by answering `nearest` queries for points $a \in A$, e.g., $\texttt{nearest}[a, B]$. The WVD can be constructed in $O(|Q| \log |Q|)$ time, and preprocessed in $O(|Q| \log |Q|)$ additional time to answer individual `nearest` queries in $O(\log |Q'|)$ time, where Q' is the size of the point set being queried, e.g., $|B|$.

The WVD can be modified for use with the l_1 norm as the ground distance, but the authors suggest the use of a RT as a simpler alternative. The RT partitions a point set Q into $O(\log |Q|)$ intervals along the x-axis. The subset of points contained within each interval $L = [x_1, x_2]$ is $Q_L = \{q \in Q : q_x \in L\}$. For each such $q \in Q_L$, $\texttt{nearest}[(x_1, q_y), Q_L]$ and $\texttt{nearest}[(x_2, q_y), Q_L]$ are stored, where q_y is the y-coordinate of q projected onto the right-hand vertical line bounding the current interval. Define

$$Q(q_y, \infty) := \{q' \in Q : q'_y \geq q_y\}$$

and

$$Q(-\infty, q_y) := \{q' \in Q : q'_y \leq q_y\}.$$

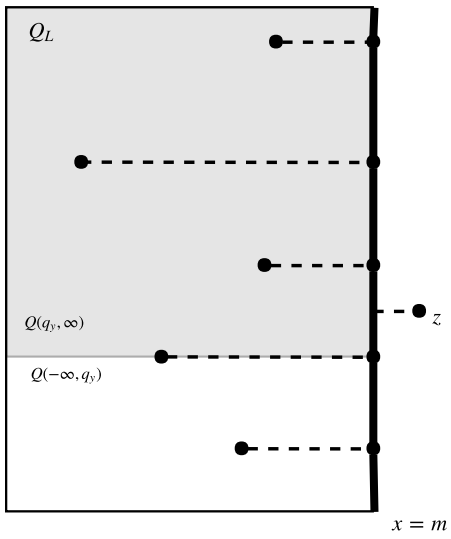

Figure 1: One split of the point set Q over which the range tree is defined recursively.

Since the distance is the l_1 norm, `nearest`$[z, Q_L]$ of a point z not in Q_L is answered by the closer of `nearest`$[(m, q_y), Q(q_y, \infty)]$ and `nearest`$[(m, q_y^{'}), Q(-\infty, q_y^{'})]$, where m is the x-coordinate of either boundary of L such that z lies on the opposite side of the points in L, and $q_y, q_y^{'}$ are the y-coordinates of the two closest points to z such that $q_y \geq z \geq q_y^{'}$ (see Figure 1). Hence, the `nearest` query can be answered by traversing only $O(\log |Q|)$ nodes of a tree, each level of which represents one of the partitions. It can be constructed in $O(|Q| \log |Q|)$ time, and queries can be answered in $O(\log |Q|)$ time. The extension to the l_∞ metric follows by rotating the plane of reference through 45°; each l_1 distance is a multiple of the l_∞ distance in the original coordinate system [8]

Given these data structures for finding admissible edges in the residual graph of the max-flow subproblems, [8] proceeds to derive an algorithm that achieves the desired $O(n^{2.5} \log n \log N)$ time bound for finding the optimal transportation. Varadarajan [10] shows that the approach of [7] for geometric bipartite matching, which was used by [8] for the $O(n^{2.5} \log n \log N)$ algorithm just described, can be improved by nearly a factor of n using geometric divide-and-conquer. They achieve a $O(n^{1.5} \log^5 n)$ time bound for min-cost perfect matching in the plane. A key component of their method involves finding a subset of $\tilde{O}(n)$ candidate edges at each phase of the primal-dual method, generated with the *semi-separated decomposition*, a relaxation of the *well-separated pairwise decomposition* (WSPD) [11]. Agarwal et. al. [12] improved the data structures of [7] for general L_p norms using the concept of *vertical decompositions* of arrangements, which requires $O(n^\epsilon)$, $\epsilon > 0$, update time to maintain a set of closest pairs of weighted points, but only incurs $O(\log n)$ query time. Recently, [13] extended this result

by considering *"vertical" shallow cuttings* and introducing a dynamic planar Voronoi diagram that has polylogarithmic update time.

A recent paper from Agarwal et. al. [14] claim that they have derived a strongly polynomial algorithm for the planar transportation problem that runs in $O(n^2 \text{polylog} n)$ time, but details on the algorithm are missing from the paper, perhaps to be included in the journal version (which is not yet available). Finally, I point out that the state-of-the-art min-cost flow algorithm by Lee and Sidford [6] that was mentioned earlier in §1 implies that in the general setting, i.e. supply and demand is integral and total demand is U, an optimal transportation can be found in $\tilde{O}(n^{2.5})$.

2.2 Approximate Geometric Algorithms

In certain applications, it may be acceptable to trade off accuracy of the optimal transportation cost for speed. One immediate potential use case is for dynamically computing transportation plans for logistics companies with global business operations. Due to the scale of their transactions, it is likely that $(1+\epsilon)$-approximate near-linear time algorithms are valuable tools for efficient and rapid decision-making. In certain situations, it is only necessary to approximately compute the transportation distance. Fast $(1+\epsilon)$ approximation algorithms for this task can achieve near-linear time bounds [15, 16, 17].

We attempt to restrict our focus to approximation algorithms for the geometric transportation problem; for a recent survey on approximation algorithms for geometric matching, see §4 of [4].

The first $(1+\epsilon)$ approximation algorithm for the geometric transportation problem found in the literature is from Sharathkumar and Agarwal [18]. For the standard geometric setting and any ground distance, they propose a $O((n\sqrt{U}\log^2 n + U\log U)\Phi(n)\log(U/\epsilon))$ ϵ-approximation algorithm. Here, $\Phi(n)$ is the query and update time of a dynamic weighted nearest neighbor data structure under the provided ground distance. Basically, this algorithm modifies the multi-scale Gabow-Tarjan transportation algorithm [3] (recall, it computes an optimal transportation in $O((\min\{\sqrt{U}, n\}n^2 + U\log U)\log nN)$); similar to Gabow-Tarjan and the algorithms from the previous section, they solve the dual of the LP. Any dynamic weighted nearest neighbor data structure with update and query time $\Phi(n)$ can be employed to efficiently compute a value defined similarly to δ from Eq. 2. The approximation parameter ϵ is used to control the number of iterations (note that after each iteration, a feasible solution is established); the algorithm is allowed to terminate once it has achieved the appropriate level of ϵ-closeness to the optimal assignment.

Next, I describe two algorithms from a recent paper by Agarwal et. al. [14]. The first is a randomized ϵ-approximation algorithm that finds the optimal transportation in $O(n^{1+\epsilon})$ expected time whose expected cost is $O(\log(1/\epsilon))\mu(\tau^*)$ if the spread of $A \cup B$ is bounded by some polynomial. This algorithm uses randomly-shifted grids to recursively decompose the problem into a set of easier subproblems that are solved by Orlin's algorithm [5]. The process of creating and solving the subproblems introduces an amount error into the solution that

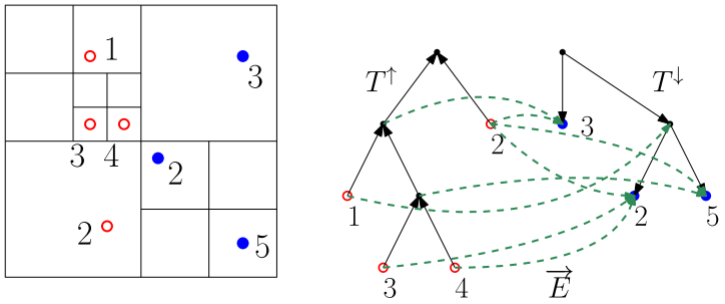

Figure 2: (Left) The point set is clustered using a compressed quadtree. (Right) The compressed quadtree is split into two trees $T^{\uparrow}$ and $T^{\downarrow}$. Pairwise distances $\overrightarrow{E}$ between the nodes of the two quadtrees are approximated with a WSPD. Figure reproduced from [14].

they bound to achieve the aforementioned expected approximate cost. For an efficient implementation in the planar case, the authors suggest using 2D dynamic orthogonal range searching data structures to store the two point sets. The randomly-shifted grid is representable with a quadtree.

The second algorithm described in [14] is a $(1+\epsilon)$ approximation algorithm that finds a transportation plan with cost $(1+\epsilon)\mu(\tau^*)$ in time $O(n^{1.5}\epsilon^{-d}\,\text{polylog}(U)\,\text{polylog}(n))$. The algorithm constructs an efficient representation of the pairwise distances between the point sets, and then makes use of Lee and Sidford's min-cost flow algorithm on the resulting flow graph. The efficient pairwise distance representation is constructed by forming a hierarchical clustering of $A \cup B$ with a compressed quadtree, with the input points as leaves. Two copies of the quadtree are made, called the up-tree and down-tree. Edges are added between the up-tree and down-tree by constructing a WSPD [11] over the two hierarchical representations of the point set (Figure 2). The WSPD forms a directed acyclic graph with a number of edges of size linear in the number of input points. The distance between any two points in the input is approximated by a unique path through the WSPD. Using Lee and Sidford's algorithm, a min-cost flow is then computed over the graph formed by the up-tree, down-tree, and the WSPD; the approximate transportation map can be recovered from this in linear time. In the planar case, this incurs a run time of $O(n^{1.5}\epsilon^{-2}\,\text{polylog}(U)\,\text{polylog}(n))$.

3 Entropic Regularization

The exact algorithms I described in the previous section become inefficient when n gets large (e.g., millions or billions of data points) or the dimension of the ambient space is large. In future study, it will be interesting to benchmark the approximate algorithms on large-scale problems of interest to the machine

learning and computer vision/graphics communities. In this section, I describe a significant result made in the machine learning community by Cuturi [19] for tackling large-scale discrete transportation problems.

The key insight is that the LP from Eq. 1 can be smoothed, enabling the use of the famous Sinkhorn matrix scaling algorithm [20]. I can rewrite the objective in Eq. 1 with matrix notation as

$$C(P) = \min_{P \in B(r,c)} \langle P, D \rangle, \tag{3}$$

where P is the $n \times n$ transportation matrix, D is the $n \times n$ positive semi-definite distance matrix, $\langle \cdot, \cdot \rangle$ is the Frobenius inner product, and $B(r, c)$ is the *transportation polytope* of r and c. The transportation polytope is the set of all real, positive $n \times n$ matrices whose rows sum to r and whose columns sum to c. Note that I am making a simple assumption that all rows and columns sum to the same integral value. In [19], they adapt a probabilistic interpretation of $B(r, c)$ as a joint probability table for two multinomial random variables X and Y taking values in $[1, n]$ with distribution r and c, respectively.

The key insight is to restrict the search over transport plans in Eq. 3 to the set of transportation matrices that satisfy an *entropic* constraint; i.e., the joint probability table $P \in B(r, c)$ satisfies an inequality in terms of the Kullback-Leibler information divergence (KL-divergence) between P and the marginals X and Y. In [19], they show that if the constraint on the KL-divergence is large enough, then the transportation distance found by solving the smoothed version of Eq. 3 is equivalent to the original transportation distance. Furthermore, the smoothed version, provided below,

$$P^{\lambda} = \arg\min_{P \in B(r,c)} \langle P, M \rangle - \frac{1}{\lambda} h(P) \tag{4}$$

can be solved with a fast, linear-time algorithm first introduced by Sinkhorn [20]. Above, λ is a Lagrange multiplier for the entropic constraint $h(P)$. This entropic regularization term enforces a simple structure on the regularized transport P^{λ}, which is what allows the use of Sinkhorn's iterative matrix scaling algorithm. Theoretically, as $\lambda \to \infty$, convergence to the optimal transport is guaranteed. However, in practice one must decide on a λ_{max} at which the algorithm must terminate, as the limitations of representing numbers with floating point in memory will be reached. More details on this topic can be found in [21].

4 Conclusion

In this section, I highlight a few research directions before concluding. It was previously mentioned that the geometric approximation algorithms discussed in §2.2 could be benchmarked on the big data and high-dimensional problems common in certain applied areas. Most likely, the amount of error that can be tolerated in the solution will vary depending on the problem. Another

research problem to consider is a streaming version of one of the network flow algorithms that can update a planar transportation plan when a small number of points are removed from the point set and a small number of points are simultaneously added at each time step. This application often arises in practice, e.g., in data association for multi-object tracking, or when considering adding and removing factories and warehouses over time. One would hope to be able to achieve a subquadratic time bound in the number of input points after the update, or perhaps quadratic only in the number of added and removed points. Observe, however, that there are some subtleties that make this challenging; it is easy to contrive of a planar point set $A \cup B$ such that inserting a new point $a' \in A$ would drastically change the minimum weight matching solution. This suggests that the worst case time bound could be a function of all of the input points. Consequentially, the approximation algorithm from [14], where they use a hierarchical clustering of the point set, seems appropriate. It should be fairly robust to small perturbations since it already approximates the pairwise distances by clustering nearby points.

References

[1] Gaspard Monge. Mémoire sur la théorie des déblais et des remblais. *Histoire de l'Académie Royale des Sciences de Paris*, 1781.

[2] Cédric Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

[3] Harold N Gabow and Robert E Tarjan. Faster scaling algorithms for generating graph-matching problems. *SIAM Journal on Computing*, 18(5):1013–1036, 1991.

[4] Pankaj K Agarwal, Esther Ezra, and Kyle Fox. Geometric optimization revisited.

[5] James B Orlin. A faster strongly polynomial minimum cost flow algorithm. *Acm Stoc*, 41(2):377–387, 1988.

[6] Yin Tat Lee and Aaron Sidford. Path finding ii: An\˜ o (m sqrt (n)) algorithm for the minimum cost flow problem. *arXiv preprint arXiv:1312.6713*, 2013.

[7] Pravin M Vaidya. Geometry helps in matching. *SIAM Journal on Computing*, 18(6):1201–1225, 1989.

[8] D. S. Atkinson and P M Vaidya. Using geometry to solve the transportation problem in the plane. *Algorithmica*, 13(5):442–461, 1995.

[9] Steven Fortune. A sweepline algorithm for voronoi diagrams. *Algorithmica*, 2(1-4):153, 1987.

[10] Kasturi R Varadarajan. A divide-and-conquer algorithm for min-cost perfect matching in the plane. In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, pages 320–329. IEEE, 1998.

[11] Paul B Callahan and S. Rao Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM*, 42(1):67–90, 1995.

[12] Pankaj K Agarwal, Alon Efrat, and Micha Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM Journal on Computing*, 29(3):912–953, 2000.

[13] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar voronoi diagrams for general distance functions and their algorithmic applications. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2495–2504. SIAM, 2017.

[14] Pankaj K. Agarwal, Kyle Fox, Debmalya Panigrahi, Kasturi R. Varadarajan, and Allen Xiao. Faster algorithms for the geometric transportation problem. In *Proceedings of the 33rd International Symposium on Computational Geometry (SoCG 2017)*, 2017.

[15] Piotr Indyk. A near linear time constant factor approximation for euclidean bichromatic matching (cost). In *SODA*, volume 7, pages 39–42, 2007.

[16] Sergio Cabello, Panos Giannopoulos, Christian Knauer, and Günter Rote. Matching point sets with respect to the earth mover's distance. *Computational Geometry*, 39(2):118–133, 2008.

[17] Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 574–583. ACM, 2014.

[18] R Sharathkumar and Pankaj K Agarwal. Algorithms for the transportation problem in geometric settings. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 306–317. Society for Industrial and Applied Mathematics, 2012.

[19] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in neural information processing systems*, pages 2292–2300, 2013.

[20] Richard Sinkhorn. Diagonal equivalence to matrices with prescribed row and column sums. *The American Mathematical Monthly*, 74(4):402–405, 1967.

[21] Gabriel Peyré, Marco Cuturi, et al. Computational optimal transport. Technical report, 2017.