A deterministic near-linear time approximation scheme for geometric transportation

Emily Fox
Department of Computer Science
The University of Texas at Dallas
Richardson, Texas 75080
Email: emily.fox@utdallas.edu

Jiashuai Lu
Department of Computer Science
The University of Texas at Dallas
Richardson, Texas 75080
Email: jslu@utdallas.edu

***Abstract*—Given a set of points $P = (P^+ \sqcup P^-) \subset \mathbb{R}^d$ for some constant d and a supply function $\mu : P \to \mathbb{R}$ such that $\mu(p) > 0\ \forall p \in P^+$, $\mu(p) < 0\ \forall p \in P^-$, and $\sum_{p\in P} \mu(p) = 0$, the geometric transportation problem asks one to find a transportation map $\tau : P^+ \times P^- \to \mathbb{R}_{\geq 0}$ such that $\sum_{q\in P^-} \tau(p,q) = \mu(p)\ \forall p \in P^+$, $\sum_{p\in P^+} \tau(p,q) = -\mu(q)\ \forall q \in P^-$, and the weighted sum of Euclidean distances for the pairs $\sum_{(p,q)\in P^+\times P^-} \tau(p,q)\cdot||q-p||_2$ is minimized. We present the first deterministic algorithm that computes, in near-linear time, a transportation map whose cost is within a $(1+\varepsilon)$ factor of optimal. More precisely, our algorithm runs in $O(n\varepsilon^{-(d+2)} \log^5 n \log\log n)$ time for any constant $\varepsilon > 0$.**

While a randomized $n\varepsilon^{-O(d)} \log^{O(d)} n$ time algorithm for this problem was discovered in the last few years, all previously known deterministic $(1+\varepsilon)$-approximation algorithms run in $\Omega(n^{3/2})$ time. A similar situation existed for geometric bipartite matching, the special case of geometric transportation where all supplies are unit, until a deterministic $n\varepsilon^{-O(d)} \log^{O(d)} n$ time $(1+\varepsilon)$-approximation algorithm was presented at STOC 2022. Surprisingly, our result is not only a generalization of the bipartite matching one to arbitrary instances of geometric transportation, but it also reduces the running time for all previously known $(1+\varepsilon)$-approximation algorithms, randomized or deterministic, even for geometric bipartite matching. In particular, we give the first $(1+\varepsilon)$-approximate deterministic algorithm for geometric bipartite matching and the first $(1+\varepsilon)$-approximate deterministic or randomized algorithm for geometric transportation with no dependence on d in the exponent of the running time's polylog.

As an additional application of our main ideas, we also give the first randomized near-linear $O(\varepsilon^{-2} m \log^{O(1)} n)$ time $(1+\varepsilon)$-approximation algorithm for the uncapacitated minimum cost flow (transshipment) problem in undirected graphs with arbitrary *real* edge costs.

I. Introduction

Let $P \subset \mathbb{R}^d$ be a set of n points in d-dimensional Euclidean space, and let $\mu : P \to \mathbb{R}$ be a function assigning each point a ***supply*** such that $\sum_{p\in P} \mu(p) = 0$. Let $P^+ = \{p \in P \mid \mu(p) > 0\}$ and $P^- = \{p \in P \mid \mu(p) < 0\}$. A ***transportation map*** $\tau : P^+ \times P^- \to \mathbb{R}_{\geq 0}$ is a non-negative assignment to each ordered pair such that for all $p \in P^+$ we have $\sum_{q\in P^-} \tau(p,q) = \mu(p)$ and for all $q \in P^-$ we have $\sum_{p\in P^+} \tau(p,q) = -\mu(q)$. A common interpretation of this setting is to imagine each point $p \in P^+$ as a pile of earth of volume $\mu(p)$ and point $p \in P^-$ to be a hole of volume $-\mu(p)$. A transportation map describes a way to move all of the earth from the piles to the holes. Accordingly, the ***cost*** of our transportation map is its total *earth-distance* according to the Euclidean norm, $\textsc{cost}(\tau) := \sum_{(p,q)\in(P^+\times P^-)} \tau(p,q) \cdot ||q-p||_2$.[1] Our goal is to find a transportation map of minimum cost $\textsc{cost}^*(P,\mu)$, a task we refer to as the ***geometric transportation problem***. Due to the analogy relating the geometric transportation problem to moving piles of earth, the optimal cost $\textsc{cost}^*(P,\mu)$ is often called the ***earth mover's distance***. The earth mover's distance is a discrete version of the 1-*Wasserstein distance* between continuous probability distractions, and the continuous version of this problem has also been referred to as the *optimal transport* or *Monge-Kantorovich* problem. Along with being an interesting math problem in its own right, earth mover's distance has applications to various topics in computer science such as shape matching and graphics [8], [11], [14], [15], [21], [23], [28]–[30].

The geometric transportation problem can be viewed as a special case of the minimum cost flow problem restricted to an uncapacitated complete bipartite graph. Unfortunately, merely constructing an explicit representation of the appropriate graph takes $\Theta(n^2)$ time. The transportation map can then be found in strongly polynomial $O(n^3 \operatorname{polylog} n)$ time using a minimum cost flow algorithm of Orlin [20]. If supplies are integral, we can instead use an algorithm of Lee and Sidford [18] that will run in $O(n^{2.5} \operatorname{polylog}(n, U))$ time where $U = \sum_{p\in P} |\mu(p)|$ is the sum of the absolute values of the points' supplies. The only faster exact algorithm we are aware of is an implicit implementation of Orlin's algorithm by [2] that runs in $O(n^2 \operatorname{polylog} n)$ time and only when given points in the plane.

Agarwal and Raghvendra [24] described an $O(n\sqrt{U} \operatorname{polylog}(U, \varepsilon, n))$-time $(1+\varepsilon)$-*approximation* algorithm for the integral supply case. [5] described an $O(n^{1+o(1)})$ time algorithm that computes a $(1+\varepsilon)$-factor *estimate* of the earth mover's distance (without associated transportation map) where the $o(1)$ hides dependencies on ε. Later, [2] described

E. Fox is supported in part by NSF grant CCF-1942597. J. Lu is currently at Google. He was supported in part by NSF grant CCF-1942597.

[1] Our results apply to any ℓ_p-norm, but we stick with the ℓ_2-norm to simplify the presentation.

a randomized algorithm with an expected $O(\log^2(1/\varepsilon))$-approximation ratio that runs in $O(n^{1+\varepsilon})$ expected time. Lahn, Mulchandani, and Raghvendra [17] described an $O(n(C\delta)^2 \operatorname{polylog}(U,n))$-time algorithm that computes a transportation map of cost at most $\textsc{cost}^*(P,\mu)+\delta U$ where $C = \max_{p\in P}|\mu(p)|$ is the maximum over the supplies' absolute values. Finally, Khesin, Nikolov, and Paramonov [16] described a randomized $(1+\varepsilon)$-approximation algorithm with running time $n\varepsilon^{-O(d)}\log^{O(d)}\textsc{Sp}(P)\log n$ where $\textsc{Sp}(P)$ is the ***spread*** or ratio of largest to smallest distance between any pair of points in P. Via a straightforward reduction, one can use their algorithm to approximately solve the integer supply case in $n\varepsilon^{-O(d)}\log^{O(d)} U\log^2 n$ time [16]. Fox and Lu [13] subsequently extended their algorithm to run in time $n\varepsilon^{-O(d)}\log^{O(d)} n$, a bound which is independent of both the spread and supplies of P.

The above history of the geometric transportation problem neatly mirrors that of the ***geometric bipartite matching problem***, the special case of geometric transportation where all supplies are either 1 or -1. (Geometric bipartite matching also requires the output map to be $0,1$, but one can guarantee that is the case with near-linear additional overhead in running time; see Section IV.) Indeed, Raghvendra and Agarwal [22], [25] achieved the same $n\varepsilon^{-O(d)}\log^{O(d)} n$ running time only after a long line of work detailed in their paper. This running time was recently improved to $n(\varepsilon^{-O(d^3)}\log\log n+\varepsilon^{-O(d)}\log^4 n\log^5\log n)$, eliminating the dependence on d from the polylog's exponent [3].

One commonality held by many of the above results, including most notably the near-linear time approximation schemes for geometric bipartite matching and transportation [3], [13], [16], [22], is that these results are Monte Carlo *randomized* algorithms that are guaranteed to work in their reported time bounds but have a small probability of not finding a good solution. These four results in particular work by approximating distances between pairs of points using a sparse graph based on a randomly shifted quadtree. Agarwal and Raghvendra [4] were able to describe a few different *deterministic* approximation algorithms for geometric bipartite matching with varying tradeoffs between approximation ratio and running time. Still, no deterministic $(1+\varepsilon)$-approximation algorithm with running time $o(n^{3/2})$ was known, even for the geometric bipartite matching problem, for nearly a decade after the initial publication [25] of Raghvendra and Agarwal's $(1+\varepsilon)$-approximation algorithm.

At STOC 2022, [1] showed that randomness was not necessary for a fast approximation of geometric bipartite matching by describing a *deterministic* $(1+\varepsilon)$-approximation algorithm that runs in $n\varepsilon^{-O(d)}\log^{O(d)} n$ time. Instead of basing distances on a single randomly shifted quadtree, they use the concept of a *tree cover*, introduced by Awerbuch, Kutten, and Peleg [7]. A tree cover can be thought of as 2^d deterministicly shifted quadtrees combined in a way to guarantee distances are well-approximated. Through a great deal of effort, they are able to apply the main ideas behind Raghvendra and Agarwal's [22] algorithm to work in the more complicated setting of a tree cover as opposed to a single tree.

It is tempting to imagine this same work can be applied to the geometric transportation problem. Unfortunately, the approach originally taken by Raghvendra and Agarwal [22] and later [3] for geometric bipartite matching is very different from the one taken by [16] and [13] for geometric transportation. The former results iteratively add matching edges along carefully chosen augmenting paths that increase in length slowly enough that they can all be found in a small amount of time. The latter results instead build a sparse spanner graph which is entrusted to a minimum cost flow approximation framework of Sherman [26] (see also [6]). Derandomizing the latter results likely requires many ideas other than those used by [1], and they presented the design of a fast deterministic approximation scheme for geometric transportation as an open problem in their paper.

A. Our results

We describe a *deterministic* $(1+\varepsilon)$-approximation algorithm for the geometric transportation problem that runs in near-linear time. Specifically, for any $\varepsilon > 0$, our algorithm is guaranteed to compute a transportation map of cost at most $(1+\varepsilon)\cdot\textsc{cost}^*(P,\mu)$, and it has a worst-case running time of $O(n\varepsilon^{-(d+2)}\log^5 n\log\log n)$.

Our approximate transportation map also has the property that each value $\tau(p,q)$ is guaranteed to be an integer if all supplies $\mu(p)$ are integers. In the special case of all supplies being -1 and 1, this property implies each value $\tau(p,q)\in\{0,1\}$; those pairs (p,q) with $\tau(p,q)=1$ form a matching. In other words, τ is a $(1+\varepsilon)$-approximate solution to the geometric bipartite matching problem.

We consider our algorithm noteworthy for two main reasons.

- It derandomizes previous approximation schemes for the geometric transportation problem, extending the recent result of [1] beyond the more specialized geometric bipartite matching problem.
- It actually *improves upon* the running times of all previously known approximation schemes for geometric transportation and geometric bipartite matching, both randomized and deterministic (we are able to remove a $\log$ factor from the running time for the special case of geometric bipartite matching; see Section V). In particular, ours is the first $(1+\varepsilon)$-approximate deterministic algorithm for geometric bipartite matching and the first $(1+\varepsilon)$-approximate determinstic *or randomized* algorithm for geometric transportion where the exponent on the running time's $\operatorname{polylog} n$ factor is bounded by an absolute constant instead of a linear function of the dimension d.

Application: Approximating uncapacitated minimum cost flow: Recent work, including [6], [19], [26], has established a connection between the geometric transportation problem and the minimum cost flow problem in uncapacitated undirected graphs. The latter is often referred to as the ***transshipment*** problem. In adjacent papers of the same proceedings, [19] and [6] both claim randomized near-linear $O(m\operatorname{polylog} n)$

time $(1+\varepsilon)$-approximation algorithms for the latter problem. Unfortunately, both algorithms require the edge costs to have bounded spread or consist of small integers. While the running times are polylogarithmic in the spread/sum of edge costs, they can become arbitrarily high when the values are allowed to be arbitrary real numbers.

That said, [6] reduces finding a $(1+\varepsilon)$-approximation for minimum cost flow to finding an $O(\log^{O(1)} n)$-approximation for a bounded spread instance of geometric transportation in $O(\log^{O(1)} n)$-dimensional Euclidean space. The efficiency of their algorithm crucially relies on both the low spread and the fact that their dependency on the dimension is merely polynomial instead of exponential. By setting the desired approximation ratio for our deterministic geometric transportation algorithm to be sufficiently large, we are also able to remove exponential dependencies on dimension while being able to handle point sets of arbitrary spread. As a consequence, we give the first randomized near-linear time $(1+\varepsilon)$-approximation algorithm for uncapacitated minimum cost flow in undirected graphs with arbitrary real costs.

B. Technical overview

Similar to how the recent result of [1] for geometric bipartite matching uses many ideas originally described by Raghvendra and Agarwal [22], our deterministic transportation algorithm is closely tied to the randomized algorithms of [16] and [13]. We will briefly review their approach and then summarize the new ideas required for its derandomization.

Randomized algorithms: The randomized transportation algorithms begin by building a randomly shifted quadtree over P, a hierarchical collection of d-dimensional cubic cells where each cell contains at most 2^d equal sized children and cells containing exactly one point act as leaves. They then add a large number of Steiner vertices to the collection of input points P and use the tree structure to build a sparse spanner graph over P and the Steiner vertices. The number of Steiner vertices added to each cell of the quadtree is $\Theta((K/\varepsilon)^d)$, where K would be the excepted distortion between any pair of vertices if the graph was constructed as a simple tree containing one Steiner vertex per cell. In [16], $K = O(\log \textsc{Sp}(P))$, while in [13], $K = O(\log n)$, and these large numbers of Steiner vertices are essentially *the* reason why $O(d)$ appears in the exponents of the runtimes' polylogs. While there are additional edges, the spanner is essentially a rooted tree where every point and nearly every Steiner vertex has exactly one *parent* Steiner vertex. Distances between points of P are approximately maintained in the spanner graph, implying the value of an uncapacitated minimum cost flow with sources and sink P will serve as a good estimate for the earth mover's distance (the cost of the optimal transportation map).

Both algorithms use a framework of Sherman [26] to approximately compute the minimum cost flow within the spanner graph. Sherman's framework requires one to formulate the uncapacitated minimum cost flow problem as finding a flow vector f of minimum cost subject to linear constraints $Af = b$ where A is the vertex-edge incidence matrix of the graph and b is a supply vector not necessarily equal to the supplies of P. One repeatedly finds flows f of approximately optimal cost that approximately satisfy linear constraints where b may vary between rounds of the process. Applied naively, the number of rounds of this process is polynomial in the *condition number* of A which can be arbitrarily large. Fortunately, it is possible to describe a *preconditioner* matrix B such that BA has low condition number. Repeatedly finding approximate solutions for constraints of the form $BAf = Bb$ suffices for finding an approximately optimal solution to the minimum cost flow problem that meets its original supply constraints exactly.

The preconditioner B is set up so that $||Bb||_1$ acts as an estimate on the cost of an approximately optimal flow f where f is found using a very restrictive kind of "oblivious" greedy approximation algorithm. Specifically, the greedy approximation algorithm must move the surplus out of/into any vertex mostly without regard to the other vertices' surpluses. The condition number of BA is proportional to the approximation factor of this greedy approximation algorithm. The specific greedy algorithm used in both papers just repeatedly moves the surplus of each vertex up to its parent. If a minimum cost flow sends some flow from a vertex u another vertex v, then the surpluses pushed up from u and v are likely to cancel at a common ancestor not too far away from either vertex. Therefore, the cost of pushing these surpluses up is bounded.

When the two algorithms finally have an approximate minimum cost flow that respects the original supplies of P, they then need to transform it into a proper transportation map without increasing its cost. To do so, they shortcut flow to avoid each of the Steiner vertices one by one, using a binary search tree based data structure to do several shortcuts at once in the case of Fox and Lu [13].

Derandomization: We now discuss our techniques for derandomizing the above algorithms. To make the these discussions easier to follow, we will begin with techniques that lead to a polylogarithmic dependence on the spread of P before adding more detail into how we handle the case of arbitrary spread.

In the previous algorithms [13], [16], randomness is directly used only for picking a randomly shifted quadtree. The "obvious" solution, then, to derandomizing the algorithms is to use a tree cover similar to how [1] derandomize the algorithm for geometric bipartite matching. Indeed, we essentially take this approach in order to construct of our sparse spanner graph. However, instead of describing it as a collection of 2^d quadtrees with different shifts, it becomes easier to think of it as a single arbitrary quadtree where each cell is given a single Steiner vertex, hereinafter referred to as its *net point*, that is directly connected to a collection of $O(\varepsilon^d)$ nearby net points of equal sized cells. These nearby connections allow for paths to go directly between close-by cells that are not comparable in the quadtree's hierarchy. Therefore, we do not need to worry about two close-by points having a distant lowest common ancestor net point in the tree, and we can guarantee distances are maintained up to a $(1+\varepsilon)$-factor while using a mere $O((n/\varepsilon^d)\log \textsc{Sp}(P))$ net points total.

The big issue with this approach becomes apparent when attempting to compute minimum cost flows within Sherman's [26] framework. Our spanner contains edges going between quadtree cells with potentially distant lowest common ancestors. Therefore, the greedy approximation algorithm from before that simply pushes surpluses to net points' parents no longer has an acceptable approximation factor. In fact, the condition number of BA may be polynomial in $\text{SP}(P)$ (and, we emphasize, *not* polynomial in $\log \text{SP}(P)$). We start to really miss the simplicity afforded us by using a single randomized quadtree without shortcuts between nearby cells.

The solution to our problem is to simulate random shifting within the greedy algorithm and preconditioner themselves. Our greedy approximation algorithm treats the initial surplus of each vertex described by the vector b as a separate commodity. For each vertex u, for each cell C closer to the root of the tree than u, we explicitly compute the probability that a random shift of the quadtree would cause the cell C to contain u. We then route an equal portion of u's original surplus to C's net point. Now, suppose a minimum cost flow sends some flow from u to another vertex v. Using similar algebra to that used in the analyses of the randomly shifted quadtree, we argue that the cost of surplus from u and v that does not cancel out at any given level is proportional to the total cost of flow sent from u to v. Therefore, the approximation factor of the greedy algorithm is proportional to $\log \text{SP}(P)$, the height of the quadtree. This greedy algorithm is still oblivious enough to imply a preconditioner with condition number roughly $\log \text{SP}(P)$, so we can make Sherman's framework run in near-linear time.

Adding edges between nearby non-related cells also complicates recovering a transportation map from the approximately optimal flow, because the connected components on each level of the quadtree now have unbounded size. We describe a new method of recovering the transportation map that no longer relies on the spanner having a particular structure. In fact, our method is general enough that it can be applied to any flow on a spanner over P, with or without Steiner vertices. In short, we process vertices in topological order according to the flow's support graph, shortcutting flow passing through each vertex we process. We continue performing the shortcuts through a single vertex in groups using the data structure of Fox and Lu [13].

Unbounded spread: The deterministic algorithm for the geometric bipartite matching problem does an $O(n \log^2 n)$ time reduction to an instance where the spread is polynomial in n [1]. Applying this reduction with the above observations is already enough to speed up the previous result; see Section V. However, we do require more work to account for unbounded spread cases of geometric transportation. The main observation used by Fox and Lu [13] to avoid dependencies on spread is that, with high probability, no input point lies within distance $\Delta/\operatorname{poly} n$ of the edge of any enclosing cell where Δ is the side length of that cell (see also [2]). These forbidden regions for cell boundaries are referred to as *moats* around the input points. Most of their algorithm design and analysis is conditioned on this high probability event. In particular, the event occuring has two main implications of interest to us: First, if the set of points within a cell has a bounding box of side length $\Delta/\operatorname{poly} n$, then those points are far enough away from everything else that we can essentially treat them as a separate instance. In turn, one can "compress" the quadtree so it has height $O(n \log n)$ using purely combinatorial operations. Second, the expected distortion of distances between any pair of points when using a single Steiner point per cell is $O(\log n)$ instead $O(\log \text{SP})$, implying a reasonable approximation ratio is acheivable by adding extra dependencies on $\log n$ to the running time. The gist of their argument is that expected distortion is proportional to the number of quadtree levels in which a pair of points may be separated with probability strictly between 0 and 1.

To replicate the first implication, we subtly move the axis-aligned hyperplanes bounding cells while building our quadtree so that no input point lies within the aforementioned $\Delta/\operatorname{poly} n$ distance from the edge of a cell. The amount we move the hyperplanes is small enough as to not affect the quality of the spanner, but it does make it possible to compress the quadtree in a similar manner to Fox and Lu [13]. In order to consistently move individual hyperplanes, we first build a collection of binary search tree based data structures that help us quickly determine whether a particular placement of a hyperplane would lie too close to one or more points.

We replicate the second implication by modifying how we simulate random shifts during the minimum cost flow phase of the algorithm. In short, our goal is to compute probabilities conditioned on shifts not placing cell boundaries too close to any vertex point. Suppose we wish to compute how much surplus various net points throughout the spanner should send to a net point at level ℓ where cells at level ℓ have side length $(1 \pm 1/\operatorname{poly} n)\Delta$. We group together maximal collections of net points called *blobs* that cannot be separated by a shift without one or more of them lying very close to the boundary of a level ℓ cell. The surpluses of a blob's constituent points are treated as a single commodity as we figure out how much surplus to route to each level ℓ net point. Now, if a minimum cost flow sends flow from net point u to net point v, there are only $O(\log n)$ levels in which the cost of moving their surpluses is non-negligible; at levels closer to the root, u and v appear in the same blob and their surpluses cancel perfectly. In order to guarantee the uncancelled portions of their surpluses still have cost proportional to the flow between u and v across the $O(\log n)$ levels that matter, we build a collection of d binary search based data structures that describe the full collection of shifts that do not separate members of any one blob into distinct level ℓ cells. Probability computations are based on the proportional amount of shift allowed according to these data structures, and we can still use similar algebra to that of the randomized algorithm analyses to prove approximation quality.

C. Organization

We describe the construction of the sparse spanner graph in Section II. We describe how to approximate the minimum cost flow within the spanner in Section III. We describe how to recover an actual transportation map from the approximate minimum cost flow and give a theorem stating our main result in Section IV. We describe some simplifications we can make to our algorithm for the case where $\textsc{Sp}(P)$ is sufficiently small in Section V. These simplifications ultimately result in a slightly lower running time for the special case of geometric bipartite matching. Finally, we describe our approximation algorithm for uncapacitated minimum cost flow in general undirected graphs in Section VI.

II. Reduction to minimum cost flow in a sparse spanner graph

In this section, we describe a way to build a sparse spanner graph $G = (V, E)$ based on a deterministically constructed quadtree. As we construct the quadtree, we will subtly shift the hyperplanes bounding its cells so that no hyperplane goes through a small rectangular ***moat*** around each input point. At the end of this section, we describe a way to reduce the geometric transportation problem to finding a minimum cost flow in our graph.

Throughout the rest of this report, we assume without loss of generality that $1/\varepsilon$ is a power of 2 and that $n \geq 1/\varepsilon$. We use $\mathbf{d}$ to denote the set of d dimensions, and $\lg$ to denote the logarithm with base 2. As is standard, we will directly prove our algorithm returns a $(1+O(\varepsilon))$-approximate transportation map. An actual $(1+\varepsilon)$-approximation can be obtained in the same asymptotic running time by dividing ε by a sufficiently large constant.

A. A data structure for avoiding moats

Before we begin constructing our quadtree, we need to build a collection of d data structures that can be queried to quickly decide if a given axis-aligned hyperplane will intersect any moats of a given size. For each dimension $i \in \mathbf{d}$, we store a sequence of balanced binary search trees in a persistent data structure [12] parameterized by moat size where the nodes of each tree correspond to maximal collections of points that cannot be separated by the hyperplane orthogonal to dimension i. Each node stores the least ith coordinate of the points in its collection along with the greatest ith coordinate of its points. Given a value x_i and a moat size λ, we can easily check in $O(\log n)$ time whether the hyperplane orthogonal to direction i with ith coordinate x_i intersects a moat, and if so, how far back in the i direction it would need to shift to no longer intersect any moat. To do so, we do both a predecessor and successor search for x_i in the tree for λ. If x_i lies between the two values $l_i \leq r_i$ stored for a node, then we need to move the hyperplane back to $l_i - \lambda$. Otherwise, if its predecessor has highest coordinate r_i and $r_i + \lambda > x_i$, we let l_i be the least coordinate of the predecessor node and move the hyperplane to $l_i - \lambda$. Finally, if the successor has least coordinate l_i with $l_i - \lambda < x_i$, we again move the hyperplane to $l_i - \lambda$.

To build the data structure for dimension i, we begin by building the tree for moat size $\lambda = 0$: it is simply a balanced binary search tree over the members of P where their ith coordinates act as the keys. We now act as if λ is continuously increasing, processing the next ***event moment*** λ where the moats around two consecutive nodes' points meet. These event moments can be computed and ordered in advance in $O(n \log n)$ time by looking at the difference in i-coordinate between consecutive points and sorting. At each event moment, we remove the two nodes whose moats are meeting and replace them with a single node. It takes $O(n \log n)$ time total to processes all the events. Again, we store the sequence of trees in a persistent data structure so we can easily access the current tree for any given value λ in $O(\log n)$ time. We require $O(\log n)$ additional space per tree in the sequence, for $O(n \log n)$ space total.

Lemma 2.1: In $O(n \log n)$ time, we can create a collection of d data structures using $O(n \log n)$ space each so that, for any given dimension $i \in \mathbf{d}$, coordinate x_i, and moat size $\lambda \geq 0$, we can lookup in $O(\log n)$ time whether the hyperplane orthogonal to dimension i at x_i intersects any point's moat, and if so, how far back in the ith dimension it needs to be shifted to avoid hitting any moats.

B. Warped quadtree

With the preprocessing out of the way, we can turn to constructing the spanner itself. We begin by building what we call a ***warped quadtree*** T on P. Let $\square_P$ be the minimum bounding hypercube containing P. Let C^* be the hypercube centered at the center of $\square_P$ but with twice its side length. Warped quadtree T has root cell C^*. The other cells of T are not necessarily hypercubes, but we do guarantee each cell is an axis-parallel box. We generally use $\Delta_{C,i}$ to denote the length of a cell C in the ith dimension and ℓ_C to denote its ***level*** or number of edges on the unique path in T from C^* to C.

We construct T iteratively as follows. We first add C^* to a queue of unprocessed cells. While there exists a cell C we have not yet processed, we remove C from the queue and perform the following steps. If there are $\lg(n^2/\varepsilon)$ ancestor cells of C including C itself that all contain a single point from P, then C is a leaf. We are done processing C.

Otherwise, let $P' \subseteq P = C \cap P$, and let $\square_{P'}$ denote the minimum bounding *hypercube* containing P'. Let $\Delta_C = \min_i \Delta_{C,i}$ and $\Delta_{P'}$ denote the side length of $\square_{P'}$. If $|P'| \geq 2$ and $\Delta_{P'} < \Delta_C/n^4$, we *contract* P' to a single point $p \in P'$ as described below before moving on to the next steps in processing C.

Now, we partition C into 2^d *approximately* equal sized axis-aligned boxes by splitting C along the following d hyperplanes. For each dimension $i \in \mathbf{d}$, the ith hyperplane lies orthogonal to dimension i. Let x_i be the average of the ith coordinates for the two bounding faces of C lying orthogonal to dimension i. We query the moat-avoiding data structure for dimension i and place hyperplane i at the coordinate the

data structure says we should use instead of x_i so it does not intersect any moat of size $\lambda = \Delta_{C,i}/(2n^2)$.

As stated, the d hyperplanes partition C into 2^d boxes. For each such box C' such that $C' \cap P \neq \emptyset$, we add C' as a child of C and add C' to the queue of unprocessed cells. We are done processing C.

We now specify how to ***contract*** a subset of points $P' \subseteq P$ as mentioned above. Let $p \in P'$ be an arbitrary member of P'. We create a new instance of the geometric transportation problem (P', μ') such that $\mu'(q) = \mu(q)$ for all $q \in P' \setminus \{p\}$ and $\mu'(p) = -\sum_{q \in (P' \setminus \{p\})} \mu(q)$. Finally, we remove all points in $P' \setminus \{p\}$ from P and change $\mu(p)$ to be $\sum_{q \in P'} \mu(q)$, the total supply of all points in P', including the original supply of p. We do not modify the currently constructed tree T or the data structures for avoiding moats when we modify P.

Later, we describe how to build a sparse spanner graph G over the contracted set of points P (note that we may perform further contractions to P before we actually construct G). We then compute a $(1+O(\varepsilon))$-approximately minimum cost flow f on G. The last component of our contraction procedure is to recursively compute one or more spanners for (P', μ') and $(1+O(\varepsilon))$-approximate flows on those spanners. In Section IV, we recover an approximately optimal transportation map from the union of these flows. The following lemma will be of use when we analyze the total cost of these flows and our final transportation map.

Lemma 2.2: Let $(P^{/}, \mu^{/})$ denote the problem instance (P, μ) after contracting P'. We have $\textsc{cost}^*(P^{/}, \mu^{/}) + \textsc{cost}^*(P', \mu') \leq (1 + O(1/n^2))\textsc{cost}^*(P, \mu)$.

Proof: Let τ be an arbitrary transportation map for (P, μ), and let $p \in P'$ be the point replacing P' during its contraction. We will construct two transportation maps $\tau^{/}$ and τ' for $(P^{/}, \mu^{/})$ and (P', μ'), respectively, such that $\textsc{cost}(\tau^{/}) + \textsc{cost}(\tau') \leq (1 + O(1/n^2))\textsc{cost}(\tau)$. For all $(a, b) \in ((P^{/})^{+} \times (P^{/})^{-})$, $a, b \neq p$, we set $\tau^{/}(a, b) := \tau(a, b)$. Similarly, for all $(a, b) \in ((P')^{+} \times (P')^{-})$, $a, b \neq p$, we set $\tau'(a, b) := \tau(a, b)$. For all $q \in ((P^{/})^{+} \setminus \{p\})$, we set $\tau^{/}(q, p) := \sum_{r \in (P')^{-}} \tau(q, r)$, and for all $q \in ((P^{/})^{-} \setminus \{p\})$, we set $\tau^{/}(p, q) := \sum_{r \in (P')^{+}} \tau(r, q)$. Note that we might now have non-zero pair assignments with p in both the first and second position, but we can shortcut "flow" going through p to make $\tau^{/}$ a valid transportation map while only reducing its cost. Similarly, for all $q \in ((P')^{+} \setminus \{p\})$, we set $\tau'(q, p) := \sum_{r \in (P^{/})^{-}} \tau(q, r)$, and for all $q \in ((P')^{-} \setminus \{p\})$, we set $\tau'(p, q) := \sum_{r \in (P^{/})^{+}} \tau(r, q)$, again shortcutting as necessary to make τ' a valid transportation map.

We now verify our claim on the total cost of $\tau^{/}$ and τ'. Consider any $q \in (P^{/} \setminus \{p\})$ and $r \in (P' \setminus \{p\})$. Our algorithm contracts P' while processing a cell C. By construction of C, every point in P' is distance at least Δ_C/n^2 from the boundary of C, and our choice to contract implies the diameter of P' is less than $\sqrt{d}\Delta_C/n^4$. Therefore, $||p - q||_2 + ||r - p||_2 \leq (1 + O(1/n^2))||r - q||_2$. To keep the algebra concise, we define $\tau^{/}(q, r)$ or $\tau'(q, r)$ to be 0 whenever (q, r) is not really in the domain of $\tau^{/}$ or τ'. We see

$$
\begin{aligned}
&\textsc{cost}(\tau^{/}) + \textsc{cost}(\tau') \\
&\quad= \sum_{(q,r) \in (P^{/} \times P^{/})} \tau^{/}(q, r) \cdot ||r - q||_2 \\
&\qquad + \sum_{(q,r) \in (P' \times P')} \tau'(q, r) \cdot ||r - q||_2 \\
&\quad= \sum_{(q,r) \in ((P^{/} \times P^{/}) \cup (P' \times P'))|q,r \neq p} \tau(q, r) \cdot ||r - q||_2 \\
&\qquad + \sum_{q \in (P^{/} \setminus \{p\})} \left(\tau^{/}(p, q) + \tau^{/}(q, p)\right) ||p - q||_2 \\
&\qquad + \sum_{q \in (P' \setminus \{p\})} (\tau'(p, q) + \tau'(q, p)) \, ||p - q||_2 \\
&\quad\leq \sum_{(q,r) \in ((P^{/} \times P^{/}) \cup (P' \times P'))|q,r \neq p} \tau(q, r) \cdot ||r - q||_2 \\
&\qquad + \sum_{q \in (P^{/} \setminus \{p\})} \sum_{r \in P'} (\tau(r, q) + \tau(q, r)) \cdot ||p - q||_2 \\
&\qquad + \sum_{q \in (P' \setminus \{p\})} \sum_{r \in P^{/}} (\tau(r, q) + \tau(q, r)) \cdot ||p - q||_2 \\
&\quad= \sum_{(q,r) \in ((P^{/} \times P^{/}) \cup (P' \times P'))} \tau(q, r) \cdot ||r - q||_2 \\
&\qquad + \sum_{q \in (P^{/} \setminus \{p\})} \sum_{r \in (P' \setminus \{p\})} (\tau(r, q) + \tau(q, r)) \\
&\qquad\quad \cdot (||p - q||_2 + ||r - p||_2) \\
&\quad\leq \sum_{(q,r) \in ((P^{/} \times P^{/}) \cup (P' \times P'))} \tau(q, r) \cdot ||r - q||_2 \\
&\qquad + \sum_{q \in (P^{/} \setminus \{p\})} \sum_{r \in (P' \setminus \{p\})} (\tau(r, q) + \tau(q, r)) \\
&\qquad\quad \cdot (1 + O(1/n^2))||r - q||_2 \\
&\quad\leq (1 + O(1/n^2))\textsc{cost}(\tau).
\end{aligned}
$$

□

C. Properties of warped quadtrees

We now prove some basic properties of the warped quadtree T.

Lemma 2.3: Suppose P has n' points remaining after all contractions used in the construction of T. Warped quadtree T contains at most $O(n' \log n)$ cells.

Proof: Consider a path of cells $\langle C_1, C_2, \ldots, C_k \rangle$ where each cell C_j in the path contains the same point subset P'. If $|P'| = 1$, then $k \leq \lg(n^2/\varepsilon) = O(\log n)$. Now suppose otherwise. For each $j \in \{2, \ldots, k\}$, we have $\Delta_{C_j} \leq ((1/2 + 1/(2n^2))\Delta_{C_{j-1}}$. Therefore, $\Delta_{P'} \leq \Delta_{C_k} \leq (1/2 + 1/(2n^2))^{k-1}\Delta_{C_1}$. On the other hand, $\Delta_{P'} \geq \Delta_{C_1}/n^4$, because we did not contract P' to one point. We again conclude that $k = O(\log n)$. We complete the proof by recalling there are at most $n' - 1$ cells C where each child of C contains strictly fewer points than C. □

Lemma 2.4: Let m be the total number of cells in all warped quadtrees, including those constructed recursively during contractions. We can construct all the warped quadtrees in $O((m+n)\log n)$ time total.

Proof: We use a similar strategy to that used in prior work [10], [13], complicated only by the existence of our data structures for avoiding moats. Given the original input point set P, we begin by creating d doubly-linked lists of the points P, each sorted by a different coordinate. As we process any cell C with point subset P', we will provide access to a sorted sublists containing just the points P'. We will also provide the total supply of points in P'.

Suppose we wish to process a cell C. If $|P'| = 1$, we check if the $\lg(n^2/\varepsilon) - 1$ first strict ancestors of C have other children in $O(\log n)$ time, and if not, declare C to be a leaf and stop processing it. If not, we use the sorted lists to determine whether $\Delta_{P'} < \Delta_C/n^4$ in constant time. If so, let p be the point chosen to represent P' after contraction. We can use the total supply of P' to compute the new supplies of p in both the current and recursive instances of the geometric transportation problem in constant time. Further, we directly hand off the sorted lists for P' to the recursive instance so that the root cell of the recursive instance can be computed in constant time as well.

After the possible contraction, we still need to find the child cells of C. For each dimension $i \in \mathbf{d}$, we use its moat avoiding data structure to compute the coordinate for the axis-aligned hyperplane orthogonal to i in $O(\log n)$ time. We then search that dimension's linked list of points from both ends in simultaneously to find where that hyperplane splits in the points in time proportional to the number of points in the less populated side of the split. We also perform individual deletions and insertions of the other dimensions' linked lists to create the sorted lists of every dimension for the less populated side in time proportional to its number of points. The remains of the original linked lists are the sorted point sets for the more populated side. When we have finished splitting along each dimension, we add all cells with at least one point to be children of C and then add them to the queue of cells to process.

Outside of splitting the point sets, we spend at most $O(\log n)$ time per each of the m cells. The total time spent doing splits throughout all cells is proportional to the total number of points going to the less populated sides of splits. Every time a point goes to a less populated cell, it shares that cell with at most half as many points as it did before, meaning we move each point to a less populated cell at most $O(\log n)$ times. The total time computing splits is therefore $O(n \log n)$, and we spend $O((m+n)\log n)$ time total computing all warped quadtrees. □

Lemma 2.5: Any cell at level ℓ has sides of length at least $\Delta_{C^*}/(2^\ell e)$ and at most $(e\Delta_{C^*})/2^\ell$ where $e \approx 2.72$ denotes Euler's number.

Proof: Assuming n is sufficiently large, Lemma 2.3 implies warped quadtree T has height strictly less than n^2. Let $i \in \mathbf{d}$ be any dimension. For any cell C and child cell C', we have $(1/2 - 1/(2n^2))\Delta_{C,i} \le \Delta_{C',i} \le (1/2 + 1/(2n^2))\Delta_{C,i}$. Therefore, the minimum side length at level ℓ is at least

$$\left(\frac{1}{2} - \frac{1}{2n^2}\right)^\ell \Delta_{C^*} > \frac{\Delta_{C^*}}{2^\ell}\left(1 - \frac{1}{n^2}\right)^{n^2} > \frac{\Delta_{C^*}}{2^\ell e},$$

and the maximum side length is at most

$$\left(\frac{1}{2} + \frac{1}{2n^2}\right)^\ell \Delta_{C^*} < \frac{\Delta_{C^*}}{2^\ell}\left(1 + \frac{1}{n^2}\right)^{n^2} < \frac{e\Delta_{C^*}}{2^\ell}.$$

□

D. Constructing the spanner

We now describe how to build our sparse spanner graph $G = (V, E)$ using the warped quadtree described above. For each cell C in T, we add a single ***net point*** N_C at the center of C. The vertices V of G constitute the set P unioned with the set of net points.

We add an edge from each point $p \in P$ to the net point N_C where C is the leaf cell containing p. For each pair of net points N_C and $N_{C'}$ such that C' is the parent cell of C, we add an edge between N_C and $N_{C'}$. By construction of T, the cells at any single level ℓ form a subset of a d-dimensional grid, albeit with somewhat uneven spacing between consecutive parallel boundary hyperplanes. For each cell C belonging to some level ℓ, we add edges from N_C to the at most $(1/\varepsilon + 1)^d - 1 = O(1/\varepsilon^d)$ other net points $N_{C'}$ where C' is another level ℓ cell at most $1/(2\varepsilon)$ grid cells away in each of the d dimensions. All edges are weighted according to the Euclidean distance between their endpoints. We let $\mathrm{dist}_G(p, q)$ denote the shortest path distance between vertices p and q in G.

Lemma 2.6: Let n' be the number of points in P after contractions. The sparse spanner graph $G = (V, E)$ has $O(n' \log n)$ vertices and $O((n'/\varepsilon^d)\log n)$ edges. Given the warped quadtree T, it can be built in $O((n'/\varepsilon^d)\log n)$ time.

Proof: The number of vertices follows immediately from Lemma 2.3. We add $O(1/\varepsilon^d)$ edges per net point, establishing the claimed total number of edges. We do the following to construct G. Recall, the cells at any particular level form a subset of a grid. For each level ℓ, we sort its cells by their location in their grid so we may efficiently find all adjacent pairs of cell. We then search the neighborhood around each cell to figure out which edges to add to that cell's net point. Each level contains at most n cell, so the total time spent sorting at all levels is $O(n' \log n)$. The time spent searching neighbors is proportional to the number of edges in G. □

Lemma 2.7: The distance between any pair of points $p, q \in P$ in G is at most $(1 + O(\varepsilon)) \cdot ||p - q||_2$.

Proof: Let C be the deepest/smallest cell containing both p and q. There exists at least one axis-aligned hyperplane splitting C into children cells that specifically separates p and q. Let this hyperplane be orthogonal to dimension i. By

construction, both p and q are distance $\Delta_{C,i}/n^2$ or more from this hyperplane. Applying Lemma 2.5, we conclude $||q-p||_2 \geq \Delta_{C^*}/(2^{\ell_C-1}en^2)$.

Let C_p and C_q be the leaf cells containing p and q, respectively. Both leaf cells lie at level $\ell_C+\lg(n^2/\varepsilon)$ or higher. Therefore, they have side lengths at most $(\varepsilon e\Delta_{C^*})/(2^{\ell_C}n^2) \leq ((\varepsilon e^2)/2)\cdot||q-p||_2$.

Let ℓ' denote the greatest level where either C_p and C_q have the same ancestor at level ℓ' or the net points of their level ℓ' ancestors are adjacent in G. Let C'_p and C'_q denote the ancestors of C_p and C_q, respectively, at level ℓ'. Suppose $C'_p \neq C_p$ and $C'_q \neq C_q$. In that case, their children cells containing p and q, respectively, do not contain adjacent net points. There must be at least $1/(2\varepsilon)$ level $\ell'+1$ cells separating those two children in some dimension, implying $||q-p||_2 \geq \Delta_{C^*}/(2^{\ell'+2}e\varepsilon)$. Meanwhile, the sides of C'_p and C'_q have length at most $(e\Delta_{C^*})/2^{\ell'} \leq (4e^2\varepsilon)\cdot||q-p||_2$. We conclude, whether or not one of C'_p or C'_q is a leaf, they have sides of length at most $(4e^2\varepsilon)\cdot||q-p||_2$.

Let $N'_p = N_{C'_p}$ and $N'_q = N_{C'_q}$ denote the net points of C'_p and C'_q, respectively. Triangle inequality implies $\mathrm{dist}_G(N'_p,N'_q) = ||N'_q-N'_p||_2 \leq ||p-N'_p||_2 + ||q-p||_2 + ||N'_q-q||_2 \leq (1+4\sqrt{d}e^2\varepsilon)\cdot||q-p||_2$. Some (admittedly loose) algebra based on the diameters of the descendent cells of C'_p and C'_q implies both $\mathrm{dist}_G(p,N'_p)$ and $\mathrm{dist}_G(q,N'_q)$ to be at most $(\sqrt{d}/2)\sum_{k=0}^{n^2}(1/2+1/(2n^2))^k((4e^2\varepsilon)\cdot||q-p||_2) \leq (4\sqrt{d}e^3\varepsilon)\cdot||q-p||_2$. Finally, we see $\mathrm{dist}_G(p,q) \leq \mathrm{dist}_G(p,N'_p)+\mathrm{dist}_G(N'_p,N'_q)+\mathrm{dist}_G(N'_q,q) \leq (1+4\sqrt{d}(e^2+e^3)\varepsilon)\cdot||q-p||_2 = (1+O(\varepsilon))\cdot||q-p||_2$. □

E. Reduction to minimum cost flow

We are now ready to reduce the problem of computing an approximately optimal transportation map for contracted instance (P,μ) to one of computing an approximately optimal minimum cost flow in our sparse spanner graph $G=(V,E)$. Our formulation of the uncapacitated minimum cost flow problem follows prior work [13], [16].

Let $\vec{E}$ be the set of edges E oriented arbitrarily. A vector $f \in \mathbb{R}^{\vec{E}}$ indexed by the oriented edges $\vec{E}$ is called a ***flow vector*** or often simply a ***flow***. Let A denote the $|V|\times|\vec{E}|$ ***vertex-edge incidence matrix*** where for each vertex-edge pair $(u,(v,w)) \in V\times\vec{E}$, we have $A_{u,(v,w)} = 1$ if $u=v$, $A_{u,(v,w)} = -1$ if $u=w$, and $A_{u,(v,w)} = 0$ otherwise. The ***divergence*** of a flow f at a vertex v is defined as $(Af)_v = \sum_{(v,w)} f_{(v,w)} - \sum_{(u,v)} f_{(u,v)}$. For each edge $(u,v)\in\vec{E}$, we abuse notation by letting $f_{(v,u)} := -f_{(u,v)}$.

Let $||\cdot||_{\vec{E}}$ denote the norm on $\mathbb{R}^{\vec{E}}$ where $||f||_{\vec{E}} = \sum_{(u,v)\in\vec{E}}|f_{(u,v)}|\cdot||v-u||_2$. We define an instance of the ***uncapacited minimum cost flow problem*** in our spanner graph G as a pair (G,b) where $b\in\mathbb{R}^V$ is a given set of vertex divergences. A feasible solution to the problem is a flow vector f such that $Af=b$. Let $\textsc{cost}^*(G,b)$ to be the minimum value $||f||_{\vec{E}}$ among feasible flow vectors f. The goal is to find a flow vector achieving this minimum.

For our reduction from the geometric transportation problem, we define $b^*\in\mathbb{R}^V$ to be the set of divergences such that $b^*_p = \mu(p)$ for all $p\in P$ and $b^*_v = 0$ for all net points v. By the construction of G and Lemma 2.7, we have $\textsc{cost}^*(P,\mu) \leq \textsc{cost}^*(G,b^*) \leq (1+O(\varepsilon))\textsc{cost}^*(P,\mu)$. Our goal in Section III is to compute a feasible flow f for (G,b^*) of cost $||f||_{\vec{E}} \leq (1+O(\varepsilon))\cdot\textsc{cost}^*(G,b^*) = (1+O(\varepsilon))\textsc{cost}^*(P,\mu)$.

We perform at most $n-1$ contraction operations across the whole of our algorithm. Lemma 2.2 implies the total cost of optimal transportation maps across all contracted instances form a $(1+O(1/n))$-approximation of the optimal cost for the original input point set. We build spanners for all of these instances, and use them to find flows of cost at most $(1+O(\varepsilon))$ times the optimal cost of a transportation map for each of these instances. The total cost of these flows is a $(1+O(\varepsilon))$-approximation of the optimal cost for the original input transportation instance. If n' is the size of any of these minimal point sets, its approximate flow will be computed in $O(n'\varepsilon^{-(d+2)}\log^5 n\log\log n)$ time, implying computing and combining all the individual flows will take $O(n\varepsilon^{-(d+2)}\log^5 n\log\log n)$ time. Finally, in Section IV, we turn this combined flow into a transportation map of no greater cost in $O(n\log^2 n)$ time.

III. Preconditioning for minimum cost flow

Let $G=(V,E)$ be the spanner defined in Section II for contracted geometric transportation instance (P,μ), and let b^* be the set of divergences defined for the corresponding instance of minimum cost flow. In this section, we describe a way to find a $(1+O(\varepsilon))$-approximate solution for the minimum cost flow instance (G,b^*) using Sherman's generalized preconditioning framework [26].

Let C_p be denote the leaf cell containing a point $p\in P$. By the definition of G, point p is incident to exact one edge connecting p to N_{C_p}. For simplicity, let f' be a flow such that, for all points $p\in P$, $f'_{(p,N_{C_p})} = b^*_p$. From now on, we assume $G=(V,E)$ does not have any point in P, and let $b\in\mathbb{R}^V$ such that $b_{N_{C_p}} = b^*_p, \forall p\in P$ and $b_v=0$ otherwise. We focus on finding an $(1+O(\varepsilon))$-approximation f of the minimum cost flow instance on (G,b). The flow $f+f'$ is then a $(1+O(\varepsilon))$-approximate solution for the minimum cost flow instance (G,b^*).

Consider any instance of the minimum cost flow problem in G with an arbitrary divergence vector $\tilde{b}\in\mathbb{R}^V$, and let $f^*_{\tilde{b}} := \mathrm{argmin}_{f\in\mathbb{R}^{\vec{E}},Af=\tilde{b}}||f||_{\vec{E}}$. A flow vector $f\in\mathbb{R}^{\vec{E}}$ is an ***$(\boldsymbol{\alpha},\boldsymbol{\beta})$-solution*** to the problem if

$$||f||_{\vec{E}} \leq \alpha||f^*_{\tilde{b}}||_{\vec{E}}$$
$$||Af-\tilde{b}||_1 \leq \beta||A||\,||f^*_{\tilde{b}}||_{\vec{E}}$$

where $||A||$ is the norm of the linear map represented by A. An algorithm yielding an (α,β)-solution is called an ***$(\boldsymbol{\alpha},\boldsymbol{\beta})$-solver***.

By arguments in [16], we seek a preconditioner $B\in\mathbb{R}^{V\times V}$ of full column rank such that, for any $\tilde{b}\in\mathbb{R}^V$ with $\sum_{v\in V}\tilde{b_v} = 0$, it satisfies

$$||B\tilde{b}||_1 \leq \min\{||f||_{\vec{E}} : f \in \mathbb{R}^{\vec{E}}, Af = \tilde{b}\} \leq \kappa||B\tilde{b}||_1 \quad (1)$$

for some sufficiently small function κ of n, ε, and d.

Let M be the time it takes to multiply BA and $(BA)^T$ by a vector. Then there exists a $(1+\varepsilon, \beta)$-solver for any $\varepsilon, \beta > 0$ for this problem with running time bounded by $O(\kappa^2(|V| + |\vec{E}| + M)\log|\vec{E}|(\varepsilon^{-2} + \log\beta^{-1})$ [26]. Moreover, if a feasible flow $f \in \mathbb{R}^{\vec{E}}$ with cost $||f||_{\vec{E}} \leq \kappa||B\tilde{b}||_1$ can be found in time K, there is a $(\kappa, 0)$-solver with running time K. By setting $\beta = \varepsilon\kappa^{-2}$ [16], the composition of these two solvers is a $(1+2\varepsilon, 0)$-solver with running time bounded by

$$O(\kappa^2(|V| + |\vec{E}| + M)\log|\vec{E}|(\varepsilon^{-2} + \log\kappa) + K).$$

A. Legal shifts, blobs, and probabilities

Our preconditioner B essentially simulates the effects of randomly shifting T along a diagonal. For each level of T, we need to determine the probabilities that a random shift, conditioned on cell boundaries not touching any moats, causes certain cells to contain certain subsets of vertices. In order to do so efficiently, we build two sets of data structures. One set helps us determine the set of legal shifts for T relative to each dimension, and the other set helps us maintain collections of points that cannot be separated into distinct cells during a legal shift. The second set of data structures will prove useful for efficiently determining how much flow to send from several vertices at once as figuring these flows out for each individual vertex would be too expensive.

Let $\Delta_\ell = \Delta_{C^*}/(2^\ell e)$ be the lower bound on the side length of any level ℓ cell as established in Lemma 2.5. For each level ℓ, we compute a maximal set s_{ℓ_i} of values in $[0, \Delta_\ell)$ per dimension $i \in \mathbf{d}$ such that no grid lines clip any moat of size $\frac{2\Delta_\ell}{n^2}$ around any vertex $v \in V$ when we shift the grid by any value in s_{ℓ_i}. We call s_{ℓ_i} the ***legal shifts*** at level ℓ relative to dimension i, and we use $s_\ell := \cap_i s_{\ell_i}$ to denote the collection of legal shifts relative to all dimensions.

For computing s_ℓ, we define a ***blob*** at level ℓ as a maximal set of points that are guaranteed to be in the same cell at this level in our quadtree after an arbitrary shift. Now we describe how to compute the set of all blobs bl_ℓ at each level ℓ. Because the moat size is equal to $\frac{2\Delta_\ell}{n^2}$, a blob can only become larger when ℓ decreases. Therefore, we compute each bl_ℓ in decreasing order of ℓ by combining smaller blobs to bigger ones as we consider each level. For any net point N_C and level $\ell < \ell_C$, let $bl_\ell(N_C)$ denote the blob containing N_C at level ℓ. We leave $bl_\ell(N_C)$ undefined for any $\ell \geq \ell_C$. Let $\square_{bl}$ be a *near*-minimum bounding box of all points in a blob bl. More specifically, if blob bl is at level ℓ, we make $\square_{bl}$ the box obtained by extending the minimum bounding box of bl by $\frac{\Delta_\ell}{n^2}$ in each dimension. We use $\square^{l,i}_{bl}$ and $\square^{r,i}_{bl}$ to denote the coordinates of left and right sides of $\square_{bl}$ for each dimension i.

We now describe a way to compute bl_ℓ given $bl_{\ell+1}$. First, for every $bl \in bl_{\ell+1}$, we extend $\square_{bl}$ so it is now a near-minimum bounding box of bl at level ℓ. Then, we sort blobs in $bl_{\ell+1}$ by the least coordinate of their bounding boxes in the first dimension. After that, we split these blobs to subsets by putting any two blobs $bl1$ and $bl2$ to the same subset if $[\square^{l,i}_{bl1}, \square^{r,i}_{bl1}]$ and $[\square^{l,i}_{bl2}, \square^{r,i}_{bl2}]$ are not disjoint, because we cannot put a grid line between these two blobs without hitting the moat around at least one point in them. For each of these subsets, we recursively perform this procedure for the remaining $d-1$ dimensions. Every subset at the lowest level of recursion is a blob in bl_ℓ. If ℓ is the largest level, we may assume $bl_{\ell+1}$ is the set of blobs where each blob contains a single vertex in V. For each blob $bl \in bl_\ell$, we call the blobs at level $\ell+1$ inside it its ***child blobs***. We use $\mathrm{children}(bl)$ to denote the set of child blobs of bl and $\mathrm{parent}(bl)$ to denote the parent blob of bl. Since the sets of points in blobs at the same level are disjoint, it is easy to see that the number of distinct blobs among all levels together is at most $2|V|-1$. Now, consider a blob bl for some level ℓ, and let C be the level ℓ cell containing bl. By the definition of T, there is an ancestor of C at most $O(\log n)$ levels closer to the root that has a sibling cell C'. Cell C' contains at least one blob bl'. Within another $O(\log n)$ levels, blobs bl and bl' will have near-bounding boxes large enough to touch. Therefore, every blob can only appear in at most $O(\log n)$ levels. We define the ***blob forest*** as the hierarchical structure of blobs defined above. For simplicity, we allow the same blob (with the same set of points) to appear multiple times in the blob forest, once per level it appears. The blob forest has $O(|V|\log n)$ nodes in total.

We now compute the legal shifts at each level ℓ using blobs in bl_ℓ. To compute s_{ℓ_i} for some dimension i, we look at $\square^{l,i}_{bl}$ and $\square^{r,i}_{bl}$ for $bl \in bl_\ell$. Let C be the cell that contains bl at level ℓ. Let $\mathrm{coor}_{l,i}(C)$ and $\mathrm{coor}_{r,i}(C)$ denote the coordinates of the left and right sides of C in dimension i, respectively. Then we call $[0, \Delta_\ell) \cap (\square^{l,i}_{bl} - \mathrm{coor}_{l,i}(C), \square^{r,i}_{bl} - \mathrm{coor}_{l,i}(C))$ the set of ***forbidden shifts*** for bl. Naturally, the set of legal shifts s_{ℓ_i} is equal to $[0, \Delta_\ell)$ minus the union of forbidden shifts of all blobs in this dimension. We can store s_{ℓ_i} in an array of size at most the number of blobs in bl_ℓ such that every element in the array is a maximal continuous subset in the union. From now on, we assume elements in s_{ℓ_i} is sorted in increasing order by their lower bounds. Therefore, we can precompute the total size of legal shifts before any element and then query the size of all legal shifts in $[x, y] \cap s_{\ell_i}$ for any values x, y in $O(\log n)$ time. The construction of all data structures mentioned above can be accomplished in $O(|V|\log^2 n)$ time total through careful use of dynamic ordered dictionaries such as balanced binary search trees.

The last preparation for constructing the preconditioner is to compute the probability that a cell C contains a blob bl if we shift the grid using a random value in s_{ℓ_C}, for every pair of bl and C at the same level. We use $\mathbb{P}[bl \in C]$ to denote this probability. Recall, the side length of any cell at level ℓ is at least Δ_ℓ. Let $C_{\ell,bl}$ be the cell containing bl at level ℓ. If we consider the legal shifts putting bl in different cells in increasing order, we see each dimension is crossed at most

once. Therefore, there are at most $d+1$ cells for which we need to calculate the probability for each blob per level. Let $\mathbb{C}_{\ell,bl}$ denote this subset of cells. Starting from the root level, for every level ℓ, we process the cells $\mathbb{C}_{\ell,bl}$. Not every cell in T has all 2^d possible children, so some of the left neighbors of $C_{\ell,bl}$ in the grid at level ℓ may not exist in T itself. For simplicity, we put a ***softlink*** to $C_{\ell,bl}$ in place of such a grid cell if it does not already have one. For each $C \in \mathbb{C}_{\ell,bl}$, we define $s_{C,bl,i}$ to be the subset of s_{ℓ_i} that could make C cover bl in dimension i. Let $l_\ell := |s_\ell|$ and $l_{C,bl} := |\cap_i s_{C,bl,i}|$. We have $\mathbb{P}[bl \in C] = \frac{l_{C,bl}}{l_\ell}$.

B. The preconditioner

The preconditioner B is the $V \times V$ matrix defined as follows. For every net point $u = N_C$, for every level $\ell < \ell_C$, we let $\mathbb{P}'[bl_\ell(u) \in C]$ denote the sum of values $\mathbb{P}[bl_\ell(u) \in C']$ for all C' equal to or softlinked to C. We set $B_{N_{C''},u} = \frac{\Delta_{\ell_C}}{3\Lambda} \cdot \mathbb{P}'[bl_\ell(u) \in C'']$ where $\Lambda = 48d^{3/2}e^2 \lg n$, for each cell $C'' \in \mathbb{C}_{\ell,bl_\ell(u)}$ that is part of T. In addition, we set $B_{u,u} = \frac{\Delta_{\ell_C}}{3\Lambda}$ for all $u \in V$ and set all other entries to 0.

Observe how for each column N_C of B, the entries for each row $N_{C'}$ with $\ell_{C'} \geq \ell_C$ are all 0, with the exception of row N_C itself. Any linear combination of columns excluding column N_C with $B_{N_C,N_C} = \frac{\Delta_{\ell_C}}{3\Lambda}$ must have at least one non-zero value for some row $N_{C'}, \ell_{C'} > \ell_C$, implying the combination does not equal column N_C. Matrix B has full column rank.

We now describe an oblivious greedy algorithm that computes a flow f such that $Af = \tilde{b}$ and the cost is at most Λ times that of the minimum cost. This algorithm is used in the algorithm explicitly as the $(\kappa, 0)$-solver discussed above, and its existence is also used in establishing the condition number of BA. We treat each blob as if it is moving the total divergence of higher level constituent vertices together up toward the root. By the time all the divergences reach the root, they will cancel each other out and the flow will be valid for the vector $\tilde{b}$. For a blob $bl \in bl_\ell$, define $b_{bl} := \sum_{N_C \in bl | \ell_C \geq \ell} \tilde{b}_{N_C}$. Observe, $b_{bl} = \sum_{bl' \in \text{children}(bl)} b_{bl'} + \sum_{N_C \in bl | \ell_C = \ell} \tilde{b}_{N_C}$. To aid in moving divergences we treat the total divergence of each child blob of bl as a separate commodity. The flow along each edge will be the sum of flows of all commodities on that edge. Precisely, for every cell C in a postorder traversal of T, for every bl with $\mathbb{P}'[bl \in C] > 0$, for every cell $C' \in \mathbb{C}_{\ell_C - 1, \text{parent}(bl)}$, we add $\mathbb{P}'[\text{parent}(bl) \in C']\,\mathbb{P}'[bl \in C] b_{bl}$ units of flow along the unique path from N_C to its parent to N'_C. Observe that for any blob $bl \in bl_\ell$ for some level ℓ, we send b_{bl} units total to level $\ell - 1$ cells.

We now establish both the approximation ratio of the greedy algorithm and the condition number of BA. Let $f^*_{\tilde{b}} := \text{argmin}_{f \in \mathbb{R}^{\vec{E}}, Af = \tilde{b}} ||f||_{\vec{E}}$. We arbitrarily decompose $f^*_{\tilde{b}}$ into a set of flows $F = \{f^1, f^2, \ldots\}$ with the following properties: 1) each flow follows a simple path between two vertices u and v; 2) for each flow $f^i \in F$ and edge $(u,v) \in \vec{E}$ either $f^i(u,v) = 0$ or its sign is equal to the sign of $f^*_{\tilde{b}}(u,v)$; 3) for each flow $f^i \in F$ and vertex v, either $(Af^i)_v = 0$ or its sign is equal to that of $\tilde{b}_v$; and 4) for each edge $(u,v) \in \vec{E}$, we have $f^*_{\tilde{b}}(u,v) = \sum_{f^i \in F} f^i(u,v)$. The existence of such a decomposition is a standard part of network flow theory.

Let f be the flow found by our greedy algorithm. We charge a portion of $||f||_{\vec{E}}$ to $||f^i||_{\vec{E}}$ for each flow f^i so that the sum of charges over all choices for f^i sum to at least $||f||_{\vec{E}}$ and for any one f^i, we overcharge by a factor of at most Λ. Fix some $f^i \in F$ sending flow from some vertex u to some vertex v. Let $b^i_u = Af^i_u$. We define $b_u(C)$ as the part of divergence b^i_u that the greedy algorithm sends to C. Observe $b^i_u = -b^i_v$. Without loss of generality we assume $b^i_u \geq 0$ and $b^i_v \leq 0$.

Now we are ready to give the main lemma of the greedy algorithm.

Lemma 3.1: Let C_u and C_v be the leaf cells containing u and v, respectively. Let ℓ be any level with $\ell \leq \min\{\ell_{C_u}, \ell_{C_v}\}$. Then the total amount of b^i_u not cancelled out by b^i_v at level ℓ is

$$\sum_{C \in T, \ell_C = \ell} \max\{0, (b_u(C) + b_v(C))\} \leq \frac{4d||u-v||_2}{\Delta_\ell} b^i_u.$$

Proof: Let $\mathbb{L}_j \subseteq \mathbb{C}_{\ell,bl_\ell(u)}$ denote the subset of cells in $\mathbb{C}_{\ell,bl_\ell(u)}$ that have lesser coordinates in dimension j, and let $\mathbb{R}_j := \mathbb{C}_{\ell,bl_\ell(u)} \setminus \mathbb{L}_j$. If $\mathbb{C}_{\ell,bl_\ell(v)} \cap \mathbb{L}_j = \emptyset$ or $\mathbb{C}_{\ell,bl_\ell(v)} \cap \mathbb{R}_j = \emptyset$ for any j, then $||u-v||_2 \geq |\text{coor}_j(u) - \text{coor}_j(v)| \geq (1 - 2/n^2)\Delta_\ell$ and the lemma holds. From here on, we assume both $\mathbb{C}_{\ell,bl_\ell(v)} \cap \mathbb{L}_j$ and $\mathbb{C}_{\ell,bl_\ell(v)} \cap \mathbb{R}_j$ are non-empty for all dimensions j.

Let $b_u(L_j) := \sum_{C \in \mathbb{L}} b_u(C)$ be the total amount of b^i_u sent to $\mathbb{L}_j$, and define $b_u(R_j)$, $b_v(L_j)$, and $b_v(R_j)$ similarly. There are $O(|V| \log n)$ nodes in the blob forest, and we may assume $|V| \leq O(n \log n)$. Therefore, $|s_\ell| \geq \Delta_\ell - \frac{d \cdot O(n \log^2 n)\Delta_\ell}{n^2} \geq \Delta_\ell/2$, assuming n is sufficiently large. Let $l_{\mathbb{L}_j, bl_\ell(u)}$ denote the total length of legal shifts in s_ℓ that make *any* cell of $\mathbb{L}_j$ cover $bl_\ell(u)$ and define $l_{\mathbb{L}_j, bl_\ell(v)}$ similarly. We have $l_{\mathbb{L}_j, bl_\ell(u)} - l_{\mathbb{L}_j, bl_\ell(v)} \leq ||u-v||_2$. Therefore,

$$b_u(L_j) + b_v(L_j) = \frac{l_{\mathbb{L}_j, bl_\ell(u)} - l_{\mathbb{L}_j, bl_\ell(v)}}{s_\ell} b^i_u \leq \frac{2||u-v||_2}{\Delta_\ell} b^i_u.$$

Similarly, $b_u(R_j) + b_v(R_j) \leq 2||u-v||_2/\Delta_\ell \cdot b^i_u$.

Finally, summing over all dimensions j, we have

$$\begin{aligned}
&\sum_{C \in T, \ell_C = \ell} \max\{0, (b_u(C) + b_v(C))\} \\
&\quad \leq \sum_{j \in \mathbf{d}} \max\{0, b_u(L_j) + b_v(L_j)\} \\
&\qquad + \max\{0, b_u(R_j) + b_v(R_j)\} \\
&\quad \leq \frac{4d||u-v||_2}{\Delta_\ell} b^i_u.
\end{aligned}$$

□

Lemma 3.2: The flow computed by the greedy algorithm overcharges the cost of f^i by a factor of at most $48d^{3/2}e^2 \lg n$.

Proof: Let ℓ be the smallest value such that $4d||u-v||_2 \geq \Delta_\ell$. By Lemma 3.1, the divergence of b^i_u remaining at level ℓ and greater is b^i_u. The divergence at each level ℓ' is sent to

net points at level $\ell' - 1$ through paths of length at most $3\sqrt{d}e^2\Delta_{\ell'}$. So the cost of carrying b_u^i to level ℓ is at most $\sum_{\ell' > \ell} 3\sqrt{d}e^2\Delta_{\ell'} b_u^i \leq 3\sqrt{d}e^2\Delta_\ell b_u^i \leq 12d^{3/2}e^2||f^i||_{\vec{E}}$ in total. Starting from level $\ell' = \ell$, the cost of carrying the remain divergence of b_u^i to one level less than the current level is at most $\frac{4d||u-v||_2}{\Delta_{\ell'}} b_u^i \cdot 3\sqrt{d}e^2\Delta_{\ell'} \leq 12d^{3/2}e^2||f^i||_{\vec{E}}$. Because $\Delta_{\ell-1} > 4d||u-v||_2$, we have $\Delta_{(\ell-1)-(2\lg n - 2 - \lg d)} > n^2||u-v||_2$. Moats will force u and v to be in the same blob at any level $\ell' \leq (\ell-1) - (2\lg n - 2 - \lg d)$. This means for any cell C with $\ell_C \leq (\ell-1) - (2\lg n - 2 - \lg d)$, we have $b_u(C) + b_v(C) = 0$. So we spend at most $12d^{3/2}e^2||f^i||_{\vec{E}}$ cost to send divergence of b_u^i per level from level ℓ to level $(\ell-1) - (2\lg n - 2 - \lg d)$, and we spend zero cost for b_u^i after that. In total, the greedy algorithm charges at most $(2\lg n - \lg d)12d^{3/2}e^2||f^i||_{\vec{E}}$ cost to divergence of u in f^i. Similarly, we can show the cost it charges to divergence of v has the same upper bound, and the lemma holds. □

In our algorithm, $\frac{|B\tilde{b}_{N_C}|3\Lambda}{\Delta_{\ell_C}}$ divergence leaves the net point N_C of a cell C through paths of length at most $3\sqrt{d}e^2\Delta_{\ell_C}$. On the other hand, these divergences leave N_C through edges of length at least $\sqrt{d}(1/2 - (1/(n^2+1)))\Delta_{\ell_C}$. All together, we see $||B\tilde{b}||_1 < \frac{||f||_{\vec{E}}}{\Lambda} \leq ||f^*||_{\vec{E}}$. Therefore, by setting $\kappa = 9\sqrt{d}e^2\Lambda$, we have

$$||B\tilde{b}||_1 < \min\{||f||_{\vec{E}} : f \in \mathbb{R}^{\vec{E}}, Af = \tilde{b}\} \leq ||f||_{\vec{E}} \leq \kappa||B\tilde{b}||_1.$$

Lemma 3.3: Applications of BA and $(BA)^T$ to arbitrary vectors $f \in \mathbb{R}^{\vec{E}}$ and $\tilde{b} \in \mathbb{R}^V$, respectively, can be done in $O(|E|\log n)$ time.

Proof: Let $A' = Af$ and let $b' = B^T\tilde{b}$. Both A, f and b' has $O(|E|)$ non-zero entries, so we can compute A' and A^Tb' in $O(|E|)$ time given b'. We show how to compute BA' and $B^T\tilde{b}$ in $O(|E|\log n)$ time.

Computing BA': Let C_u be the cell with u as its net point. By the definition of B, for each cell C,

$$\begin{aligned}(BA')_{N_C} &= \frac{\Delta_C}{3\Lambda}A'_{N_C} + \frac{\Delta_C}{3\Lambda}\sum_{u\in V, \mathbb{P}'[bl_{\ell_C}(u)\in C]>0} \mathbb{P}'[bl_{\ell_C}(u)\in C]A'_u \\ &= \frac{\Delta_C}{3\Lambda}A'_{N_C} + \frac{\Delta_C}{3\Lambda}\sum_{bl\in bl_{\ell_C}, \mathbb{P}'[bl\in C]>0} \mathbb{P}'[bl\in C]\sum_{u\in bl, \ell_{C_u}>\ell} A'_u\end{aligned}$$

There are at most $2|V| - 1$ different blobs and $\sum_{u\in bl,\ell_{C_u}>\ell} A'_u = \sum_{bl'\in\text{children}(bl)}\sum_{u\in bl,\ell_{C_u}>\ell+1} A'_u + \sum_{u\in bl',\ell_{C_u}=\ell+1, bl'\in\text{children}(bl)} A'_u$ for each blob at some level ℓ. So we can compute $\sum_{u\in bl,\ell_{C_u}>\ell} A'_u$ for each blob bl in $O(|V|\log n)$ time in total during a postorder traversal of the blobs. After that, we can fill in all entries in BA' in $O(|E|)$ time.

Computing b': For every point u, except $B^T_{u,u}$, every non-zero entry in B^T_u corresponds to a net point N_C of a cell C with $\mathbb{P}'[bl_{\ell_C}(u) \in C] > 0$. Let C_u be the cell with u as its net point. We have $b'_u = \frac{\Delta_{C_u}}{3\Lambda}\tilde{b}_u + \sum_{C,\mathbb{P}'[bl_{\ell_C}(u)\in C]>0}\frac{\Delta_C}{3\Lambda}\mathbb{P}'[bl_{\ell_C}(u) \in C]\tilde{b}_{N_C}$. Let bl^- denote the set of strict ancestor blobs of a blob bl. Let ℓ be any level where $bl_\ell(u)$ is defined. We have $\sum_{C,\ell_C\leq\ell,\mathbb{P}'[bl_{\ell_C}(u)\in C]>0}\frac{\Delta_C}{3\Lambda}\mathbb{P}'[bl_{\ell_C}(u) \in C]\tilde{b}_{N_C} = \sum_{C,\ell_C=\ell,\mathbb{P}'[bl_\ell(u)\in C]>0}\frac{\Delta_C}{3\Lambda}\mathbb{P}'[bl_\ell(u) \in C]\tilde{b}_{N_C} + \sum_{bl'\in bl_\ell(u)^-}\sum_{C,\mathbb{P}'[bl'\in C]>0}\frac{\Delta_C}{3\Lambda}\mathbb{P}'[bl' \in C]\tilde{b}_{N_C}$. We can compute $\sum_{bl'\in bl^-}\sum_{C,\mathbb{P}'[bl'\in C]>0}\frac{\Delta_C}{3\Lambda}\mathbb{P}'[bl'\in C]\tilde{b}_{N_C}$ for each blob bl in $O(|V|\log n)$ time in total during a preorder traversal of the blobs. Then we can fill in each entry of b' in constant time. □

We have shown there exists a $(1+2\varepsilon, 0)$-solver for the minimum cost flow problem on G. Plugging in all the pieces, we see the running time of the solver is at most $O(|E|\varepsilon^{-2}\log^4 n \log\log n)$.

IV. Recovering a transportation map from a flow

Let $\hat{G} = (V, E)$ be *any* connected graph such that $P \subseteq V \subset \mathbb{R}^d$ and each edge has weight equal to the Euclidean distance between endpoints. Let A be the vertex-edge incidence matrix of $\hat{G}$, and let $\hat{f} \in \mathbb{R}^{\vec{E}}$ be any flow in $\hat{G}$ such that $A\hat{f} = \mu$ where $\mu(v) = 0$ if $v \notin P$. In this section, we show how to transform $\hat{f}$ into a transformation map for (P, μ) where $\text{COST}(\tau) \leq ||\hat{f}||_{\vec{E}}$. Throughout this section, we let $m = |E|$. We also assume $m = O(n^4)$, as we could simply compute an *optimal* transportation map from scratch otherwise using an algorithm for minimum cost flow in general graphs [20].

Let $\vec{E}'$ denote the edges of the complete graph over V where each edge is oriented consistently with its counterpart in $\vec{E}$ if it exists and oriented arbitrarily otherwise. We maintain a flow $f \in \mathbb{R}^{\vec{E}'}$ where initially $f_{(u,v)} = \hat{f}_{(u,v)}$ if $uv \in E$, and $f_{(u,v)} = 0$ otherwise. We will eventually guarantee $f_{(u,v)} \neq 0$ only for $u, v \in P$.

For each point $p \in P$, there are potentially $\Theta(|E|)$ other vertices that may at some point during the process directly send flow to or receive flow from p. We cannot afford to update these flow assignments individually, so for each vertex $v \in V$, we instead maintain a single ***prefix split tree*** [13] $S(v)$ that will contain representations of certain vertices sending flow to v. A prefix split tree S is an ordered binary tree where each node η is assigned a non-negative ***potential*** $\phi(\eta)$. We let $\phi(S)$ denote the total potential of nodes in S. Prefix split trees containing s nodes support the following operations in amortized $O(\log s)$ time each:

- INSERT(S, ϕ): Insert a node of potential ϕ into tree S and return a reference to this node.
- DELETE(S, η): Delete the node η from the tree S.
- MERGE(S, S'): Modify S by adding all nodes of S' after the nodes of S, emptying S' in the process.
- PREFIXSPLIT(S, t): Assume $0 \leq t \leq \phi(S)$. If a prefix of nodes in S has total potential exactly t, then let η_1 be

the last member of this prefix. Otherwise, let η be the first node where the prefix through η has total potential greater than t. ***Split*** η by replacing it in-place with two nodes η_1 and η_2 such $\phi(\eta_1)+\phi(\eta_2) = \phi(\eta)$ and the prefix through η_1 has total potential exactly t. Either way, create a new tree S' containing the prefix through η_1 and remove this prefix from S.

Each node η in $S(v)$ represents a vertex $u \in V$, and each vertex may be represented by multiple nodes, even within a single prefix split tree. We denote the vertex represented by η as $r(\eta)$. All split tree $S(v)$ are initially empty. When a node η is split into two nodes η_1 and η_2, we set $r(\eta_1) = r(\eta_2) := r(\eta)$. Along with the prefix split trees, we maintain a so-called ***base flow*** $f' \in \mathbb{R}^{\vec{E}}$ that is initially equal to f. We maintain the invariant that for each pair of vertices u and v, $f'_{(u,v)} + \sum_{\eta \in S(v) | r(\eta)=u} \phi(\eta) = f_{(u,v)}$.

The ***support*** of flow f' is the set of undirected edges uv for which $f'_{(u,v)} \neq 0$. We begin by changing f' and therefore f so that its support is a forest. We use a process inspired by the acyclic flow algorithm of Sleator and Tarjan [27]. Let $\bar{G} = (V, \bar{E})$ be initially empty. We iteratively process each directed edge $(u, v) \in \vec{E}$ such that $f'_{(u,v)} > 0$. When it comes time to process (u, v), we check if u and v are in the same component of $\bar{G}$. If not, we add uv to $\bar{G}$. Otherwise, let π be the directed path from u to v in $\bar{G}$. We define the ***unit cost*** of π to be $|\pi| := \sum_{(x,y)\in\pi | f_{(x,y)}>0} ||y - x||_2 - \sum_{(x,y)\in\pi | f_{(x,y)}<0} ||y - x||_2$, the amount $||f||_{\vec{E}}$ increases per unit of flow added to the directed edges of π. If $||v-u||_2 \geq |\pi|$, let $(o, p) = \arg\min_{(x,y)\in\left(\left\{(x',y')\in\pi | f_{(x',y')}<0\right\}\cup(v,u)\right)} -f_{(x,y)}$, the first edge to go to 0 flow if we "reroute" as much flow along π instead of (u, v) as we can, and let $F = f_{(o,p)}$. If $||v-u||_2 < |\pi|$, let $(o, p) = \arg\min_{(x,y)\in\pi | f_{(x,y)}>0} f_{(x,y)}$ and $F = -f_{(o,p)}$ instead. We modify f' by increasing the flow along all directed edges of π by F and decreasing the flow along (u, v) by F. Doing so causes $f'_{(o,p)} = 0$. If $op \neq uv$, we remove op from $\bar{G}$ and add uv in its place. We are now done processing uv. Observe $\bar{G}$ remains a forest after processing each edge. Therefore, each edge can be processed in (amortized) $O(\log n)$ time using standard extensions to dynamic tree data structures [27].

Lemma 4.1: The above procedure does not change Af, the cost of $||f||_{\vec{E}}$ does not increase, and the support of f becomes a forest. Further, if $\mu(p)$ is an integer for all $p \in P$, then the procedure guarantees $f_{(u,v)}$ is an integer for all $uv \in E$.

Proof: Each time the flow f', and thus f, are changed, we do so by changing the route some flow takes between a pair of vertices u and v. We change the flow along the path π by the opposite amount we change $f'_{(u,v)}$, so the vector Af does not change. Further, the choice to increase or decrease flow along π is made so that the change cannot increase $||f||_{\vec{E}}$. Whenever an edge uv is about to be added to $\bar{G}$ and create a cycle, we remove an edge (possibly uv itself) from that cycle. Therefore $\bar{G}$ and the support of f is a forest.

For the claim about f being integral, observe that it is trivially true if every component of $\bar{G}$ contains one vertex. If some component contains multiple vertices, let u be a leaf in that component, and let uv be its one incident edge. Because $(Af)_u$ is integral, $f_{(u,v)}$ must be integral as well. If we (for the sake of proof) remove u from $\bar{G}$ and set $f_{(u,v)} = 0$, then $(Af)_v$ remains integral. The claim follows by induction on the number of vertices in $\bar{G}$. □

Consider the orientation of $\bar{G}$ such that for each directed edge (u, v) in the orientation, $f_{(u,v)} > 0$. We now process each vertex in topological order with respect to this orientation.

Suppose it is time to start processing vertex v. Our procedure guarantees that 1) $f'_{(u,v)} = 0$ for each vertex u that has already been processed, 2) $f'_{(v,w)}$ has not yet changed for each vertex w that we have not yet processed, and 3) v is not yet represented in $S(w)$ for *any* vertex w.

From the above guarantees and the definition of $\bar{G}$, we may conclude that $f'_{(v,w)} \geq 0$ for any vertex w we have not yet processed. Our goal is to shortcut flow passing through v from a processed vertex u to an unprocessed vertex w by adding to w's split tree. If $v \in P^+$, then let η be the node returned by INSERT$(S(v), \mu(v))$ and set $r(\eta) \leftarrow v$. In doing so, we're implicitly declaring that v is receiving $\mu(v)$ units of flow from itself, and we don't have to set up any special cases for when we want v to actually send flow. This moment is the only time we create new nodes for the split trees. Observe whether or not we create a new node, we now have $\phi(S(v)) \geq \sum_{w\in V} f'_{(v,w)}$.

While there exists a vertex w such that $f'_{(v,w)} > 0$, we do the following. Let S' be the tree returned by PREFIXSPLIT$(S(v), f'_{(v,w)})$. We perform a MERGE$(S(w), S')$, shortcutting the flow through v to w as desired. Finally, we set $f'_{(v,w)} \leftarrow 0$ as all flow into w originally from v is now represented in $S(w)$. We are done processing v when the while loop concludes. We may easily verify each of our guarantees hold for later vertices in the topological order.

Consider when we have finished processing all the vertices. Those vertices $v \in P^+$ are represented as one or more nodes in the vertices' split trees, and these nodes have total potential $\mu(v)$. Those vertices $v \in P^-$ each have a split tree of total potential $\phi(S(v)) = -\mu(v)$. We now construct the transportation map τ. Initially $\tau(u, v) = 0$ for all $(u, v) \in P^+ \times P^-$. While there exists a split tree $S(v)$ containing at least one node η, we increase $\tau(r(\eta), v)$ by $\phi(\eta)$ and perform a DELETE$(S(v), \eta)$. When the loop completes, we are done constructing τ.

Lemma 4.2: The algorithm above produces a transportation map τ for (P, μ) such that COST$(\tau) \leq ||\hat{f}||_{\vec{E}}$ in $O(m \log n)$ time. Further, if $\mu(p)$ is an integer for all $p \in P$, then the procedure guarantees $\tau(p, q)$ is an integer for all $(p, q) \in (P^+ \times P^-)$.

Proof: The fact that τ is a transportation map for (P, μ) follows from the above discussions. Observe that every time we change f while processing vertices in topological order, we do so by rerouting flow going from some vertex u through a vertex v and then to a vertex w. By triangle inequality, these shortcuts can only reduce the cost of f, implying our bound on COST(τ). If $\mu(p)$ is integral for all $p \in P$, then $f_{(u,v)}$

is integral immediately before we start processing vertices in topological order. Each change reroutes an amount of flow equal to the flow along an edge, so the flow values remain integral.

For running time, we observe we perform a constant number of split tree operations for each of the m or fewer edges in $\bar{G}$ while processing the vertices in topological order. These operations takes $O(m \log n)$ time total. We then do a number of split trees operations equal to the total number of nodes in all split trees while adding values to pairs in the transportation map τ. The only operations that can add nodes to a split tree are the INSERTs done for each vertex of positive supply, and the PREFIXSPLITs done for each edge in $\bar{G}$. Therefore, we create $O(m)$ nodes total and remove them from the split trees in $O(m \log n)$ time. Adding in the $O(m \log n)$ time needed to construct and topologically sort $\bar{G}$, we conclude our proof of the running time. □

We are now ready to state and prove our main theorem.

Theorem 4.3: There exists a deterministic algorithm that, given a set of n points $P \subset \mathbb{R}^d$ and a supply function $\mu : P \to \mathbb{R}$, runs in time $O(n\varepsilon^{-(d+2)} \log^5 n \log\log n)$ and returns a transportation map τ with cost at most $(1+\varepsilon) \cdot \text{COST}^*(P, \mu)$. Further, if $\mu(p)$ is an integer for all $p \in P$, then $\tau(p, q)$ is an integer for all $(p, q) \in (P^+ \times P^-)$.

Proof: Recall, we build a warped quadtree T while contracting certain subsets of P. Let (P', μ') denote the geometric transportation instance after contraction and let $n' = |P'|$. We build the sparse spanner graph $G = (V, E)$ over P' in $O(n'\varepsilon^{-d} \log n)$ time. Let $m = |E| = O(n'\varepsilon^{-d} \log n)$. We define an instance of uncapacitated maximum flow (G, b^*) where b^*_v is equal to $\mu'(v)$ if $v \in P'$ and equal to 0 otherwise. By Lemma 2.7, $\text{COST}^*(G, b^*) \leq (1 + O(\varepsilon))\text{COST}^*(P', \mu')$. We compute a flow f of cost $(1 + O(\varepsilon)) \cdot \text{COST}^*(G, b^*) = O(1 + O(\varepsilon))\text{COST}^*(P', \mu')$ in $O(m\varepsilon^{-2} \log^4 n \log\log n) = O(n'\varepsilon^{-(d+2)} \log^5 \log\log n)$ time using the algorithm described in Section III.

By the discussion at the end of Section II, we can combine the spanner G and the flow f with the recursively computed spanners' $(1 + O(\varepsilon))$-approximate flows for each contracted subset of P to yield a flow $\hat{f}$ for a single spanner $\hat{G}$ on (P, μ). This flow has cost $||\hat{f}||_{\vec{E}} = (1+O(\varepsilon)) \cdot \text{COST}^*(P, \mu)$. Finally, we compute a transportation map τ for (P, μ) with cost at most $||\hat{f}||_{\vec{E}} = (1+O(\varepsilon))\text{COST}^*(P, \mu)$. If $\mu(p)$ is an integer for all $p \in P$, then $\tau(p, q)$ is an integer for all $(p, q) \in (P^+ \times P^-)$ per the above discussions.

By Lemmas 2.3 and 2.4, we spend $O(n \log^2 n)$ time total constructing all warped quadtrees across the various recursive subproblems. We then spend $O(n\varepsilon^{-(d+2)} \log^5 n \log\log n)$ time computing flows for all individual subproblems and $O(n \log^2 n)$ time transforming the flows into a transportation map. We conclude our proof. □

V. SIMPLIFYING THE ALGORITHM FOR LOW SPREAD CASES

In this section, we sketch some simplifications that can be made to our algorithm for the case that $\text{SP}(P)$ is small. Our simplified algorithm computes a $(1 + \varepsilon)$-approximation of the optimal transportation map in $O(n\varepsilon^{-(d+2)}(\log n + \log^3 \text{SP}(P) \log\log \text{SP}(P)) \log \text{SP}(P))$ time. When $\text{SP}(P) = n^{O(1)}$, the running time of the simplified algorithm is slightly better than the one for the unbounded spread case.

Instead of building a warped quadtree as in the first half of Section II, we use a standard quadtree T where all cells at a level have exactly the same size and the leaves are exactly those cells containing one point of P. Therefore, we do not need the moat avoidance data structures. There is no need to contract subsets of P, and the depth of T is $\log \text{SP}(P) + 1$. We build our sparse graph $G = (V, E)$ on T using the procedure described in Section II-D. Lemma 2.7 still holds on G. The time to construct T and G is $O(n\varepsilon^{-d} \log \text{SP})$ and $|E| = O(n\varepsilon^{-d} \log \text{SP})$ as well.

When finding the $(1 + O(\varepsilon))$-approximation for the minimum cost flow instance (G, b^*), we no longer worry about moats. For the greedy algorithm and preconditioner in Section III, we essentially treat each point $u \in V$ as its own blob appearing at every level of the quadtree. At level ℓ, we allow *all* shifts in $[0, \Delta^*/2^\ell]^d$, thus eliminating the need for the legal shift and blob data structures. Lemma 3.1 and Lemma 3.2 together imply the conditioner number κ of the preconditioner in the low spread case is at most $144d^2 \log \text{SP}(P)$. Therefore, we can compute a flow with cost at most $(1+O(\varepsilon)) \cdot \text{COST}^*(P, \mu)$ in $O(n\varepsilon^{-(d+2)}(\log^4 \text{SP}(P) \log\log \text{SP}(P)))$ time using Sherman's preconditioner framework.

Our procedure for recovering a transportation map from the flow is unchanged, running in $O(n\varepsilon^{-d} \log n \log \text{SP}(P))$ time. Considering everything above, we get the following theorem.

Theorem 5.1: There exists a deterministic algorithm that, given a set of n points $P \subset \mathbb{R}^d$ of spread $\text{SP}(P)$ and a supply function $\mu : P \to \mathbb{R}$, runs in time $O(n\varepsilon^{-(d+2)}(\log n + \log^3 \text{SP}(P) \log\log \text{SP}(P)) \log \text{SP}(P))$ and returns a transportation map τ with cost at most $(1 + \varepsilon) \cdot \text{COST}^*(P, \mu)$. Further, if $\mu(p)$ is an integer for all $p \in P$, then $\tau(p, q)$ is an integer for all $(p, q) \in (P^+ \times P^-)$.

Recall, the geometric bipartite matching problem is the special case where $\mu(p) \in \{-1, 1\}$ for all P, and the transportation map is required to assign either 0 or 1 to each pair of points. Approximating an arbitrary case of the geometric bipartite matching problem can be reduced in $O(n \log^2 n)$ time to an instance where the spread is polynomial in n [1]. As our algorithm is guaranteed to return a $0, 1$ map given such an instance, we conclude with the follow corollary.

Corollary 5.2: There exists a deterministic algorithm that, given an n-point instance of the geometric bipartite matching problem, returns a $(1 + \varepsilon)$-approximately optimal matching in time $O(n\varepsilon^{-(d+2)} \log^4 n \log\log n)$.

VI. Uncapacitated minimum-cost flow in general graphs

Previously, we reduced approximating the geometric transportation problem to approximating a special case of minimum-cost flow without edge capacities. In this section, we turn the situation around by showing how to approximate minimum-cost flow in a *general* graph via reductions to our algorithm for geometric transportation. Our algorithm is based on the one given in [6] for the case of moderate integer edge costs.

Let $G = (V, E)$ be an arbitrary undirected graph, let $||\cdot||_{\vec{E}}$ denote an arbitrary norm on $\mathbb{R}^{\vec{E}}$, and let $b \in \mathbb{R}^V$ denote an arbitrary set of vertex divergences. In this section, we let $n := |V|$ and $m := |E|$. Fix a parameter $\varepsilon > 0$. We again use Sherman's [26] framework as described in Section III. Accordingly, we need a preconditioner $Q \in \mathbb{R}^{r \times V}$ of full column rank such that

$$||Q\tilde{b}||_1 \leq \min\{||f||_{\vec{E}} : f \in \mathbb{R}^{\vec{E}}, Af = \tilde{b}\} \leq \kappa ||Q\tilde{b}||_1 \quad (2)$$

for any $\tilde{b} \in \mathbb{R}^V$ and with κ small. Note that $r \neq |V|$ in this case; we'll leave it unspecified for now. We also need to describe an efficient κ-approximate "oblivious" greedy algorithm to help us estimate κ. However, as in [6], we'll actually run iterations of Sherman's framework until it suffices to use a simple n-approximation to satisfy the final set of divergences.

We'll begin with the greedy algorithm as it makes it easier to describe the preconditioner itself. We start with the following lemma.

Lemma 6.1 ([9]): There is a randomized algorithm which can output a mapping $\psi : V \to \mathbb{R}^d$ with $d = O(\log^2 n)$ with constant probability in $O(m \log^2 n)$ time such that for all $u, v \in V$

$$\text{dist}_G(u, v) \leq ||\psi(u) - \psi(v)||_2 \leq O(\log n) \cdot \text{dist}_G(u, v).$$

A solution to the geoemtric transportation problem for $\psi(V)$ should form a reasonable estimate of the cost of the optimal flow. Unfortunately, the dimension of the target space is moderately large. We can deal with the large dimensionality by using the following weakening of our main result.

Theorem 6.2: Suppose d *is not* constant. There exists a deterministic algorithm that, given a set of n points $P \subset \mathbb{R}^d$ and a supply function $\mu : P \to \mathbb{R}$, runs in time $O(dn \log n)$ and returns a transportation map τ with cost at most $O(d^2 \log n) \cdot \text{COST}^*(P, \mu)$.

Proof: We build the spanners as before, except we place leaves immediately when a cell contains exactly one point and add edges only between net points and the net points of their neighboring cells. The resulting spanners have $O(dn \log n)$ vertices and edges total, and they maintain shortest path distances up to an $O(\sqrt{d})$ factor. See Lemma 2.7.

We define the preconditioner B as before. A single iteration of the greedy algorithm results in an $O(d^{3/2} \log n)$ approximation to the spanner's minimum cost flow instance. See Lemma 3.2. We run a single iteration of the greedy algorithm in each spanner in $O(dn \log^2 n)$ time total, resulting in $O(d^2 \log n)$ approximately optimal flows. We combine and transform them into proper transportation maps in $O(dn \log^2 n)$ additional time as described in Section IV. □

Our greedy algorithm for seeking approximately optimal flows on G computes a Bourgain embedding as described in Lemma 6.1. We can then use the algorithm of Theorem 6.2 to get an $O(d^2 \log n) \cdot O(\log n) = \log^{O(1)} n$ approximation on the minimum-cost flow value for the original problem in $n \log^{O(1)} n$ time. Note that our algorithm for minimum cost flow need not actually extract a transportation map from the spanner flows.

We are now ready to describe the preconditioner Q needed for the minimum-cost flow instance on G. Let V' denote the full set of net points in each of the spanners built by the algorithm of Theorem 6.2. Let Q^1 denote the $|V'| \times |V|$ $0-1$ matrix where $Q^1_{\psi(u), N_{C_{\psi(u)}}} = 1$ for all vertices $u \in V$, and all other entries are 0. Let Q^2 denote the $|V'| \times |V'|$ real-valued matrix composed of the spanners' individual preconditioner matrices where $Q^2_{u,v} = B_{u,v}$ if u and v belong to the same spanner with preconditioner B. All other entries of Q^2 are 0. Finally, let $Q = Q^2 Q^1 \in \mathbb{R}^{V' \times V}$.

Matrix Q has full column rank. The value of κ in (2) is $\log^{O(1)} n$. For any $f \in \mathbb{R}^{\vec{E}}$, we can compute QAf in $O(m \log n) + n \log^{O(1)} n$ time by first computing $q' := Q^1 Af$ and then applying the algorithm of Lemma 3.3 to compute $Q^2 q'$. In fact, we can compute $Q^2 q'$ in time proportional to the size of the spanners, because we no longer need to track which blob flow originally came from. For any $\tilde{b} \in \mathbb{R}^V$, we can compute $(QA)^T \tilde{b}$ in the same time by first computing $b' := {Q^2}^T \tilde{b}$ and then computing $A^T {Q^1}^T b'$.

Assuming we compute a good embedding with Lemma 6.1, there exists a $(1 + \varepsilon, \varepsilon^{1+\lg n}/\kappa)$-solver for the minimum cost flow instance that performs $\varepsilon^{-2} \log^{O(1)} n$ matrix multiplications. We can compose this solver with a simple $(n, 0)$-solver that runs in $m \log^{O(1)} n$ time to get a $(1 + O(\varepsilon))$-approximate solution to the minimum cost flow instance. The total time spent is $m \log^{O(1)} n$. We can run our algorithm $O(\log n)$ times to guarantee success with high probability $1 - 1/n^c$ for any constant c.

Theorem 6.3: There exists a randomized algorithm that, given an undirected graph $G = (V, E)$ with n vertices and m edges, an arbitrarily norm on $\mathbb{R}^{\vec{E}}$, and an arbitrarily set of vertex divergences $b \in \mathbb{R}^V$, runs in time $m\varepsilon^{-2} \log^{O(1)} n$ and returns a $(1+\varepsilon)$-approximate uncapacitated minimum cost flow in G with high probability.

Acknowledgments

The authors would like to thank the anonymous reviewers of earlier versions of this paper for their helpful and insightful comments.

References

[1] P. K. Agarwal, H.-C. Chang, S. Raghvendra, and A. Xiao, "Deterministic, near-linear ε-approximation algorithm for geometric bipartite matching," in *Proc. 54th Symp. Theory Comput.*, 2022, pp. 1052–1065.

[2] P. K. Agarwal, K. Fox, D. Panigrahi, K. R. Varadarajan, and A. Xiao, "Faster algorithms for the geometric transportation problem," in *Proc. 33rd Intern. Symp. Comput. Geom.*, 2017, pp. 7:1–7:16.

[3] P. K. Agarwal, S. Raghvendra, P. Shirzadian, and R. Sowle, "An improved ϵ-approximation algorithm for geometric bipartite matching," in *Proc. 18th Scandinavian Symp. and Workshops Algo. Theory*, vol. 227, 2022, pp. 6:1–6:20.

[4] P. K. Agarwal and R. Sharathkumar, "Approximation algorithms for bipartite matching with metric and geometric costs," in *Proc. 46th Symp. Theory Comput.*, 2014, pp. 555–564.

[5] A. Andoni, A. Nikolov, K. Onak, and G. Yaroslavtsev, "Parallel algorithms for geometric graph problems," in *Proc. 46th Symp. Theory Comput.*, 2014, pp. 574–583.

[6] A. Andoni, C. Stein, and P. Zhong, "Parallel approximate undirected shortest paths via low hop emulators," in *Proc. 52nd Ann. ACM SIGACT Symp. Theory of Comput.*, 2020, p. 322–335.

[7] B. Awerbuch, S. Kutten, and D. Peleg, "On buffer-economical store-and-forward deadlock prevention," *IEEE transactions on communications*, vol. 42, no. 11, pp. 2934–2937, 1994.

[8] N. Bonneel, M. van de Panne, S. Paris, and W. Heidrich, "Displacement interpolation using Lagrangian mass transport," *ACM Trans. Graph.*, vol. 30, no. 6, pp. 1–12, 2011.

[9] J. Bourgain, "On Lipschitz embedding of finite metric spaces in Hilbert space," *Israel J. Math.*, vol. 52, no. 1–2, 1985.

[10] P. B. Callahan and S. R. Kosaraju, "Faster algorithms for some geometric graph problems in higher dimensions," in *Proc. 4th Ann. ACM-SIAM Symp. Discrete Algorithms*, 1993, pp. 291–300.

[11] M. Cuturi and A. Doucet, "Fast computation of Wasserstein barycenters," in *Proc. 31st Intern. Conf. Machine Learning*, ser. JMLR Workshop and Conference Proceedings, vol. 32, 2014, pp. 685–693.

[12] J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan, "Making data structures persistent," *J. Comput. System Sci.*, vol. 38, no. 1, pp. 86–123, 1989.

[13] K. Fox and J. Lu, "A near-linear time approximation scheme for geometric transportation with arbitrary supplies and spread," *J. Comput. Geom.*, vol. 13, no. 1, pp. 204–225, 2022.

[14] P. Giannopoulos and R. C. Veltkamp, "A pseudo-metric for weighted point sets," in *Proc. 7th Europ. Conf. Comput. Vision*, ser. Lecture Notes in Computer Science, A. Heyden, G. Sparr, M. Nielsen, and P. Johansen, Eds., vol. 2352, 2002, pp. 715–730.

[15] K. Grauman and T. Darrell, "Fast contour matching using approximate earth mover's distance," in *Proc. 24th IEEE Conf. Comput. Vision and Pattern Recog.* IEEE Computer Society, 2004, pp. I:220–I:227.

[16] A. B. Khesin, A. Nikolov, and D. Paramonov, "Preconditioning for the geometric transportation problem," *J. Comput. Geom.*, vol. 11, no. 2, pp. 234–259, 2021.

[17] N. Lahn, D. Mulchandani, and S. Raghvendra, "A graph theoretic additive approximation of optimal transport," in *Proc 32nd Adv. Neur. Info. Proces. Sys.*, 2019, pp. 13 813–13 823.

[18] Y. T. Lee and A. Sidford, "Path finding methods for linear programming: Solving linear programs in $\tilde{o}$(vrank) iterations and faster algorithms for maximum flow," in *Proc. 55th IEEE Ann. Symp. Found. Comput. Sci.*, 2014, pp. 424–433.

[19] J. Li, "Faster parallel algorithm for approximate shortest path," in *Proc. 52nd Ann. ACM SIGACT Symp. Theory of Comput.*, 2020, pp. 308–321.

[20] J. B. Orlin, "A faster strongly polynomial minimum cost flow algorithm," *Operations Research*, vol. 41, no. 2, pp. 338–350, 1993.

[21] G. Peyré and M. Cuturi, "Computational optimal transport: With applications to data science," *Foundations and Trends in Machine Learning*, vol. 11, no. 5-6, pp. 355–607, 2019.

[22] S. Raghvendra and P. K. Agarwal, "A near-linear time ε-approximation algorithm for geometric bipartite matching," *Journal of the ACM (JACM)*, vol. 67, no. 3, p. Article No. 18, 2020.

[23] Y. Rubner, C. Tomasi, and L. J. Guibas, "The earth mover's distance as a metric for image retrieval," *Intern. J. Comput. Vision*, vol. 40, no. 2, pp. 99–121, 2000.

[24] R. Sharathkumar and P. K. Agarwal, "Algorithms for the transportation problem in geometric settings," in *Proc. 23rd Ann. ACM-SIAM Symp. Discrete Algorithms*, 2012, pp. 306–317.

[25] ——, "A near-linear time ε-approximation algorithm for geometric bipartite matching," in *Proc. 44th Symp. Theory Comput.*, 2012, p. 385–394.

[26] J. Sherman, "Generalized preconditioning and undirected minimum-cost flow," in *Proc. 28th Ann. ACM-SIAM Symp. Discrete Algorithms*, 2017, pp. 772–780.

[27] D. D. Sleator and R. E. Tarjan, "A data structure for dynamic trees," *J. Comput. Syst. Sci.*, vol. 26, no. 3, pp. 362–391, 1983.

[28] J. Solomon, F. de Goes, G. Peyré, M. Cuturi, A. Butscher, A. Nguyen, T. Du, and L. Guibas, "Convolutional Wasserstein distances: Efficient optimal transportation on geometric domains," *ACM Trans. Graph.*, vol. 34, no. 4, 2015.

[29] J. Solomon, R. M. Rustamov, L. J. Guibas, and A. Butscher, "Earth mover's distances on discrete surfaces," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 67:1–67:12, 2014.

[30] C. Villani, *Optimal Transport: Old and New*. Springer Science & Business Media, 2008, vol. 338.