A Simple Deterministic Near-Linear Time Approximation Scheme for Transshipment with Arbitrary Positive Edge Costs

Emily Fox
The University of Texas at Dallas, Richardson, TX, USA

Abstract

We describe a simple deterministic near-linear time approximation scheme for uncapacitated minimum cost flow in undirected graphs with positive real edge weights, a problem also known as transshipment. Specifically, our algorithm takes as input a (connected) undirected graph $G = (V, E)$, vertex demands $b \in \mathbb{R}^V$ such that $\sum_{v \in V} b(v) = 0$, positive edge costs $c \in \mathbb{R}^E_{>0}$, and a parameter $\varepsilon > 0$. In $O(\varepsilon^{-2} m \log^{O(1)} n)$ time, it returns a flow f such that the net flow out of each vertex is equal to the vertex's demand and the cost of the flow is within a $(1 + \varepsilon)$ factor of optimal. Our algorithm is combinatorial and has no running time dependency on the demands or edge costs.

With the exception of a recent result presented at STOC 2022 for polynomially bounded edge weights, all almost- and near-linear time approximation schemes for transshipment relied on randomization to embed the problem instance into low-dimensional space. Our algorithm instead deterministically approximates the cost of routing decisions that would be made if the input were subject to a random *tree embedding*. To avoid computing the $\Omega(n^2)$ vertex-vertex distances that an approximation of this kind suggests, we also take advantage of the clustering method used in the well-known Thorup-Zwick distance oracle.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Network flows

Keywords and phrases Transshipment, minimum cost flow, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2024.56

Related Version *Full*: `https://arxiv.org/abs/2307.07440`

Funding *Emily Fox*: Supported in part by NSF grant CCF-1942597.

1 Introduction

Let $G = (V, E)$ be an undirected graph with positive edge **costs** $c \in \mathbb{R}^E_{>0}$, and let $b \in \mathbb{R}^V$ be a set of vertex **demands** (alternatively, one may prefer the term *supplies*). While we formally define c and b as vectors with components indexed by E and V, respectively, we use the familiar function application notation $c(e)$ and $b(v)$ for the cost of an edge $e \in E$ and demand for a vertex $v \in V$, respectively. We say b is **proper** if $\sum_{v \in V} b(v) = 0$.

Let $\vec{E}$ denote an arbitrary orientation of the edges E. We denote the oriented instance of an edge $e \in E$ as $\vec{e}$. Let $I_G \in \mathbb{R}^{V \times \vec{E}}$ be the vertex-edge incidence matrix for G with $I_G(v, \vec{e})$ equal to 1 if v is the tail of $\vec{e}$, equal to -1 if v is the head of $\vec{e}$, and equal to 0 otherwise. We say a **flow** $f \in \mathbb{R}^{\vec{E}}$ **routes** b if $I_G f = b$. In the (uncapacitated) minimum cost flow problem, one seeks a flow of minimum cost $c(f) = \sum_{e \in E} c(e)|f(\vec{e})|$ subject to f routing b. In other words, we seek a minimum cost way to send units of some single commodity throughout the edges of G such that each vertex $u \in V$ with $b(u) > 0$ sends out $b(u)$ units of commodity into the graph and each vertex $v \in V$ with $b(v) < 0$ removes $-b(v)$ units from the graph. This special case of minimum cost flow in an undirected graph without edge capacities is also called **transshipment**.

Transshipment generalizes various problems that have been studied in their own right such as shortest paths in undirected graphs, the discrete optimal transport problem [17], and other assignment problems on metric spaces. In fact, several recent papers studying these more specific problems have relied on reductions to the more general transshipment problem [1, 2, 8–11, 14, 19]. The study of new algorithms for transshipment can provide immediate improvements or simplifications to many of these results along with providing new insights that may be beneficial to minimum cost and other flow problems in general.

1.1 Recent results

The study of flow problems such as transshipment has a long history going back several decades. Here, we highlight some of the strongest or more recent closely related results to the current work. We use n and m to denote the number of vertices and edges, respectively, in the input graph.

As a special case of minimum cost flow, there are several polynomial time algorithms for computing exact solutions to transshipment. Orlin [12] described a strongly polynomial transshipment algorithm that runs in $O(n \log n(m+n \log n))$ time, and this algorithm remains the fastest algorithm known for real edge costs and vertex demands. There has been a great deal of recent activity in the design of minimum cost flow and transshipment algorithms that assume integer costs and capacities or demands in some range $[1, U]$, starting with an $O(m^{3/2} \log^{O(1)}(nU))$ time algorithm by Daitch and Spielman [6] and culminating in a pair of very recent **almost-linear** $m^{1+o(1)} \log^2 U$ time algorithms [4, 5].

The existence of almost-linear time exact algorithms for minimum cost flow was alluded to a few years earlier by the demonstration of various almost- and near-linear time **approximation schemes** for transshipment. Let $\varepsilon > 0$, and let $\mathrm{OPT}(b)$ denote the minimum cost of any flow that routes b. Sherman [15] described an $O(\varepsilon^{-2} m^{1+o(1)})$ time algorithm that finds a flow f routing b with total cost $c(f) \leq (1+\varepsilon)\mathrm{OPT}(b)$. Sherman's main observation was a novel method for finding solutions to linear systems $Ax = d$ that approximately minimize an arbitrary norm $||x||$. His method involved composing solutions from well-known weak approximate solvers by repeatedly applying the solver to residual vectors $d - Ax$. The number of iterations needed for this method to converge is a function of the so-called **generalized condition number** of A. To reduce the condition number, he proposed finding a left-cancellable matrix P called a **generalized preconditioner** such that PA is well-conditioned and then working with the system $PAx = Pb$. He then expressed transshipment as such a linear system problem and described a preconditioner P that could be efficiently applied in iterations of his composition algorithm.

Recently, the authors of [11] and [1] independently discovered **near-linear** $O(\varepsilon^{-2} m \log^{O(1)}(nU))$ time approximation schemes for transshipment. Here, U is best understood as the **aspect ratio** of the edge costs found by dividing the largest edge cost by the smallest. Shortly after, Fox and Lu [8] proposed a near-linear $O(\varepsilon^{-2} m \log^{O(1)} n)$ time approximation scheme *without* the dependence on the aspect ratio. While the above results were presented here with sequential running times in mind, there have been several recent approximation schemes proposed for various models of parallel and distributed computing, including some appearing in a subset of the work cited above [1, 2, 11, 14, 19].

Perhaps unsurprisingly, the algorithms of [1] and [8] use the aforementioned framework of Sherman [15] explicitly but with a more-efficiently evaluated choice for the preconditioner P. However, *all* of the approximation schemes for transshipment mentioned above rely on methods for refining loose approximate solutions into stronger ones. Zuzic [18] recently provided an explanation for this commonality by uniting the approaches of these works

under a single simple **boosting** framework. In short, all of these works implicitly build approximately optimal results to the linear programming dual for transshipment and then take advantage of the newly realized fact that *any* black-box dual approximation can be boosted to a $(1+\varepsilon)$-approximate solution for transshipment.

Another commonality between all of the almost- and near-linear time approximation schemes cited above, with a single exception, is that they rely *heavily* on randomization. In particular, they all compute random Bourgain [3] embeddings of the shortest path metric into low-dimensional space and then they compute (the cost of) random *oblivious* flows that are based only on the location of their sources within that space. The single near-linear time exception to this use of randomization is a recent paper [14] describing an $O(\varepsilon^{-2}m\log^{O(1)}(nU))$ time *deterministic* approximation scheme. However, unlike some of the results mentioned above [8, 15], its running time is still polylogarithmic in the aspect ratio of the edge costs.

1.2 Our results

We present the first near-linear time approximation scheme for transshipment that is both deterministic and with a running time independent of the aspect ratio of the edge costs. Specifically, our algorithm computes a flow f that routes b at cost $c(f) \leq (1+\varepsilon)\mathrm{OPT}(b)$ in $O(\varepsilon^{-2}m\log^{O(1)} n)$ time. It is also (in our opinion) simpler and likely easier to implement than previous near-linear time approximation schemes for transshipment, even after excising the extra complications needed for them to efficiently function in parallel or distributed settings. Outside what is explicitly described in this report, it depends upon just two black box results, the aforementioned boosting framework of Zuzic [18] implicitely used by all previous almost- and near-linear time transshipment approximation schemes and the deterministic construction [13] of a well-known distance oracle of Thorup and Zwick [16]. Our algorithm is also combinatorial in that the only operations it performs with the input costs and demands are comparisons, addition, multiplication, and division.

Linear cost approximators

Prior work

One method of instantiating the transshipment boosting framework [18] is to design an **α-approximate linear cost approximator** $P \in \mathbb{R}^{k\times V}$ based only on the input graph $G=(V,E)$ and edge costs c. In particular, for *any* set of proper demands b, we must have $\mathrm{OPT}(b) \leq ||Pb||_1 \leq \alpha\mathrm{OPT}(b)$. Approximator P need not be computed explicitly. If matrix-vector multiplications with P and P^T can be performed in some time M, then we can compute a flow f' with $c(f') \leq (1+\varepsilon/2)\mathrm{OPT}(b)$ and $\mathrm{OPT}(b-I_Gf') \leq \mathrm{OPT}(b)/n^2$ in $O(\varepsilon^{-2}\alpha^2 M\log^{O(1)} n)$ time [18, Corollaries 12 and 16]. We can then route an n-approximate flow for demands $b-I_Gf'$ along a minimum spanning tree to get our desired $(1+\varepsilon)$-approximate flow that routes b exactly.

Linear cost approximators are almost the same as the generalized preconditioners mentioned above, and we can look to prior work on how to design one. In particular, the construction we use is partially motivated by the near-linear time approximation schemes of [1, 8]. As in most of the previous approximation schemes for transshipment, they compute a random Bourgain [3] embedding of the input graph's shortest path metric, mapping vertices to points in low dimensional space. They then construct preconditioners for estimating the optimal solution value to the *geometric transportation problem* over the vertices' points. In this latter problem, the goal is to compute a weighted matching between several pairs of points of minimum total distance.

Consider a hierarchy of subsets of the vertices' points based on a sequence of progressively finer randomly shifted uniform grids. By greedily matching points within lower levels of the hierarchy before moving to the top, one obtains a weighted matching with expected cost close to optimal. The approximation schemes of [1, 8] construct preconditioners to estimate this expected cost by building a collection of *deterministic* (i.e., not randomly shifted) grids and explicitly computing the net expected amount of demand within each grid cell, *as if they had been randomly shifted*. Their preconditioners (modulo appropriate scaling) simply output the diameter of each grid cell times the net expected demand it contains.

Novel construction inspired by tree embeddings

Instead of using a random Bourgain embedding and then building a hierachy of subsets, our approximation scheme skips straight to considering the hierarchies formed from random embeddings into dominating *tree metrics* [7] where the distance between any given pair of vertices is stretched (distorted) by a factor of at most $O(\log n)$ in expectation. Consider the following variation of the tree embedding of [7]. Let r be the root of our tree. We compute a 2-approximation Δ of the diameter of G and choose a partition $\langle v_1, v_2, \ldots, v_n \rangle$ of the vertices uniformly at random. We create a sequence of disjoint clusters $\langle C_1, C_2, \ldots, C_n \rangle$ where $C_i \subseteq V$ for all i. Let Δ' be chosen uniformly at random from $[\Delta/2, \Delta]$. For each i from 1 to n, we add to C_i all vertices within distance Δ' that have not already been claimed for another cluster. We create a child c_i of r for each non-empty cluster C_i, connected by an edge of weight $O(\Delta)$. Finally, we recursively build a tree rooted at c_i for each non-empty C_i. Despite the low expected stretch, some distances may be distorted by a factor of $\Omega(n)$. So while for any fixed b, the expected cost of an optimal flow routing it in the tree is $O(\log n) \cdot \mathrm{OPT}(b)$, there may be some choices of b for which the cost blows up by that $\Omega(n)$ factor. Therefore, we cannot simply create a linear cost approximator based on the cost of routing different demands within the tree and expect it to give us good approximate costs relative to G for the large number of b vectors used in the transshipment boosting framework.

However, the *expected* costs of routing demand through potential cluster centers is a fixed value that can be computed accurately. Ignoring running time concerns, we can construct a linear cost approximator P that accurately lists these expected costs up to constant factors. Because the expected stretch from an actual random tree embedding using the above algorithm is $O(\log n)$, the value $||Pb||_1 \leq O(\log n) \cdot c(b)$ always.

Distance oracles and graph minors

Unfortunately, we cannot afford to compute the $\Omega(n^2)$ distances required to properly compute these expected costs. Instead we take advantage of the clustering method used in the well-known distance oracle of Thorup and Zwick [16]. Let $k \geq 1$ be an integer. After $O(kmn^{1/k} \log n)$ time deterministic preprocessing, their $O(kn^{1+1/k})$-space oracle can compute $(2k-1)$-approximate distances between any pair of vertices in $O(k)$ time [13, 16]. By setting $k := \lg n$, we get an $O(\log n)$-approximate distance oracle with construction time $O(m \log^2 n)$ and size $O(n \log n)$. We never directly use the construction for its stated purpose as an oracle. Instead, we use the fact that the deterministic construction stores for each $v \in V$ a set of $O(\log^2 n)$ vertices $w \in V$ that suffice as the possible cluster centers for our expected cost computations. The actual algebra proving we get a good approximator using the expected costs is, unsurprisingly, very similar to the algebra proving a random tree embedding has low expected stretch.

There is one thing left to consider in the design of our algorithm. Even using the distance oracle, we would have to consider all possible distance scales to get correct expected costs for all possible cluster centers across all possible cluster diameters. Doing so would lead to a polylogarithmic dependence on the aspect ratio in our running time. To avoid that dependence, we construct a sequence of $O(m \log n)$ minors of G, each maintaining a range of possible shortest path distances up to a constant factor. Edges of higher cost than the range of a minor are discarded, and those of significantly smaller cost are contracted, so the total size of these minors is also $O(m \log n)$. Our construction of the linear cost approximator P simply considers expected costs within each minor separately. When establishing the approximation ratio of P, we charge against the cost of individual flow paths in a decomposition of an optimal flow. Fortunately, only $O(\log n)$ minors charge meaningful costs to each flow path, bringing the approximation ratio of P to a relatively small $O(\log^3 n)$.

Oblivious routing and comparison to [14]

Along with the explanation given above, the linear cost approximator P can be interpreted as providing constant approximations to the actual cost of a certain **oblivious** flow where a unit flow from/to each vertex u to/from an arbitrary vertex s is chosen without prior knowledge of b and then multiplied by $b(u)$. The description of the flow is incredibly simple; for each adjacent pair of scales for which we consider how a random tree embedding might affect u, between each pair of potential cluster centers between those scales, the unit flow for u sends the product of the probabilities that u would join their respective clusters. Another way to think about the flow is that for each scale, we want the demand of u to arrive at various nearby cluster centers, and using the product of proportions between scales to route flow is the most natural way to do so. The actual construction of P follows this oblivious routing/reassignment interpretation, as we believe it more easily suggests that P is estimating the cost of an actual solution to the transshipment problem. That said, our algorithm never actually computes an oblivous flow, because we merely need applications of its cost approximator P to use the boosting framework. Also, the actual choice of *where* to send the flow units and the analysis for P's approximation ratio is much better explained using the tree embedding motivation described above.

This oblivious routing of actual flow interpretation/approach is used much more heavily throughout the $O(\varepsilon^{-2} m \log^{O(1)}(nU))$ time deterministic approximation scheme of [14], so it provides a means by which to compare our work to theirs. Their paper contains the many additional details necessary for working in parallel and distributed models that are beyond the scope of the current work, so we focus just on the parts related to approximating transshipment in any model. Similar to how our approach is inspired by random tree embeddings, theirs is inspired by random *low-diameter decompositions* of the input graph at different scales and their deterministic counterparts, the *sparse neighborhood covers*. They observe that sending of a vertex u's demand from cluster center to cluster center can lead to an oblivious flow having high cost, so they propose sending portions of the demand to nearby cluster centers proportionally to their distance and then routing flow between scales based on the product of these proportions. Our expected cost/demand reassignment calculations also bias sending demand to nearby cluster centers. However, the algorithm of [14] and its analysis is made more complicated compared to ours by, for example, them only sending flow between centers of nesting clusters.

Also similar to our work, they build various simplifications of the input graph designed to efficiently consider clusters of a certain scale. They build $O(\log(nU))$ simplifications to handle all possible different scales they might need to consider. The analysis of their

oblivious routing's cost must charge to the full cost of the optimal flow paths once per scale, leading to their obliviously routed flow having a cost $O(\log^{O(1)}(nU))$ times optimal. This approximation ratio then becomes part of their algorithm's running time as in all boosting based approximation schemes. In contrast, our minors have total size $O(m \log n)$ and are designed so each path in an optimal flow will receive significant charges from only $O(\log n)$ of them, leading to a tidy $O(\log^3 n)$ approximation ratio for our linear cost approximator.

1.3 Organization

We proceed as follows. We discuss a few more needed details concerning the Thorup-Zwick distance oracle in Section 2. We discuss the construction of the minor graphs (thereafter referred to as *layers* of G) in Section 3. The construction and application of our $O(\log^3 n)$-approximate linear cost approximator P is given in Section 4; the reader merely interested in *how* our algorithm works can stop there given the description of the boosting framework available above. In Section 5, we prove P is an $O(\log^3 n)$-approximate linear cost approximator. We briefly wrap things up in Section 6 with the presentation of a theorem stating our main result.

2 Thorup-Zwick distance oracle

Thorup and Zwick [16] presented a **distance oracle** that for any integer $k \geq 1$ has size $O(kn^{1+1/k})$ and can return $(2k-1)$-approximate distances between any two vertices in $O(k)$ time. It can be constructed deterministically in $O(kmn^{1/k} \log n)$ time [13, 16]. We now discuss some more details of the oracle relevant to our algorithm.

Let $\delta(u, v)$ denote the distance between vertices u to v in $G = (V, E)$ and let $\delta(u, V') = \min_{v \in V'} \delta(u, v)$ for any subset $V' \subseteq V$. The distance oracle stores a sequence of vertex subsets $V = S^0 \supseteq S^1 \supseteq \cdots \supseteq S^k = \emptyset$ we refer to as **samples**. We have $S^{k-1} \neq \emptyset$. We assume distances between any fixed vertex v and the other vertices of G are distinct, breaking ties as necessary. The oracle also stores, for each vertex v, a **bundle** $B(v) = \cup_{j=0}^{k-1} B^j(v)$ of vertices and their distances from v where each **bundle piece** $B^j(v) = \left\{w \in S^j \mid \delta(v, w) < \delta(v, S^{j+1})\right\}$. In particular, $B^j(v) \subseteq S^j \setminus S^{j+1}$.

The total size of all bundles is $O(kn^{1+1/k})$. Some bundles may have size larger than the average size of $O(kn^{1/k})$. However, a careful examination of the deterministic construction of the oracle [13] shows $|B(v)| = O(kn^{1/k} \log n)$ for all $v \in V$.

3 Layer graphs

Let $G = (V, E)$ be a connected undirected graph with positive edge costs $c \in \mathbb{R}^E_{>0}$, and let $n := |V|$ and $m := |E|$. We assume without loss of generality that G contains no loops or parallel edges and that $n \geq 4$. Our approximation scheme begins by computing a sequence $\langle (G_0, \Delta_0), (G_1, \Delta_1), \ldots, (G_L, \Delta_L) \rangle$ of pairs, each consisting of a *minor* G_i of G, also referred to as a **layer** of G, and a **reach** Δ_i such that $\Delta_i \leq \Delta_{i-1}/2$ for all $i \geq 1$. Each iteration of the approximate optimization procedure will take time near-linear in the sum of the minors' sizes, so we must make sure that each edge of G appears in only $O(\log n)$ different minors. Accordingly, our construction sets each G_i, including for $i = 0$, to be the graph G after contracting all edges of cost at most Δ_i/n, deleting all edges of cost strictly greater than $2\Delta_i$, and removing all vertices left isolated given the edge deletions and contractions. We let V_i and E_i denote the vertices and edges, respectively, of each layer G_i, and let $n_i := |V_i|$ and $m_i := |E_i|$ denote the cardinality of both sets. Let $\delta_i(v, w)$ and $\delta_i(v, W)$ denote the distance from v to another vertex or set of vertices within G_i.

Let s be an arbitrary vertex of G, and let x and y be the two vertices farthest from s. We set $\Delta_0 := \delta(s,x) + \delta(s,y)$. Reach Δ_0 is at least, but no more than twice, the diameter of G. Given Δ_{i-1}, we set Δ_i as follows: If G_{i-1} is non-empty (contains at least one edge), then $\Delta_i := \Delta_{i-1}/2$. Otherwise, let $e^{||} = \arg\max_{e \in E | c(e) \leq \Delta_{i-1}/n} c(e)$ be the costliest contracted edge of G_{i-1}. If $e^{||}$ is well-defined, let $\Delta_i := c(e^{||}) \cdot n/2$. If $e^{||}$ is not well-defined, then (G_{i-1}, Δ_{i-1}) is the final pair in the sequence and $L := i - 1$.

For a vertex v in some layer G_i, we let $V(v)$ denote the set of vertices from the input graph G contracted to form v. Observe that the sets $V(\cdot)$ form a *laminar family* in that for each pair of sets, either they are disjoint or one completely contains the other. Accordingly, let i' be the largest index such that $i' < i$ and there exists a vertex $v' \in V_{i'}$ such that $V(v) \subseteq V(v')$. We define the **parent** of v to be $p(v) := v'$.

The **ancestors** of $v \in V_i$, denoted $p^\infty(v)$ are all layer graph vertices obtained by repeatedly applying the parent operation zero or more times starting with v. In particular, $p_{i'}(v)$ for some $i' \leq i$ denotes the ancestor of v in $V_{i'}$ if one exists. The **children** of v are $p^{-1}(v) := \{v' \mid p(v') = v\}$. Vertex v is called a **leaf** if it has no children. Despite the evocative names, we do not actually connect the layer graphs using any kind of rooted forest data structures and instead use them mostly separately when defining our linear cost approximator. The proofs of the following lemmas appear in the full version of this paper.

▶ **Lemma 1.** *The graphs $\langle G_0, G_1, \ldots, G_L \rangle$ have at most $O(m \log n)$ edges and vertices in total.*

▶ **Lemma 2.** *The sequence $\langle (G_0, \Delta_0), (G_1, \Delta_1), \ldots, (G_L, \Delta_L) \rangle$ along with their vertices' parents and children can be computed in $O(m \log n)$ time.*

▶ **Lemma 3.** *Let $v \in V_i$ for $i > 0$, and let $p(v) \in V_{i'}$ with $\Delta_{i'} > 2\Delta_i$. The children of $p(v)$ are exactly the members of the connected component of v in G_i. Further, all edges incident to $p(v)$ have length strictly greater than $\Delta_{i'}$, and the diameter in G of $V(p(v)) < 2\Delta_i$.*

Along with each layer G_i, we construct a Thorup-Zwick distance oracle (Section 2) with parameter $k := \lg n$. Let S_i^j, $B_i^j(v)$, and $B_i(v)$ denote the jth sample, jth bunch piece of v, and bunch of v, respectively, within layer G_i. By Lemma 1, the oracles have total size $O(\log n \cdot m \log n \cdot n^{1/\lg n}) = O(m \log^2 n)$, and $B_i(v) = O(\log^2 n)$ for each i, v. They can be constructed in $O(m \log^3 n)$ time total.

4 A linear cost approximator

In this section, we describe how to implicitly build and efficiently evaluate matrix-vector multiplications with an $O(\log^3 n)$-approximate linear cost approximator $P \in \mathbb{R}^{((\cup_i V_i) \times (\cup_i V_i)) \times V}$, i.e., the number of rows is $|\cup_i V_i|^2$ and the number of columns is n. Most rows are empty. To simplify the exposition, we will actually define three separate matrices $A \in \mathbb{R}^{(\cup_i V_i) \times V}$, $R \in \mathbb{R}^{((\cup_i V_i) \times (\cup_i V_i)) \times (\cup_i V_i)}$, and $C \in \mathbb{R}^{((\cup_i V_i) \times (\cup_i V_i)) \times ((\cup_i V_i) \times (\cup_i V_i))}$ that represent **A**ggregating demands, **R**outing flow, and estimating the **C**ost of the flow, respectively. Approximator $P := CRA$ is their product. Each of these three matrices serves a limited purpose in a three-step process of obtaining a cost approximation. Matrices R and C are sparse and can be built and stored explicitly by our algorithm. On the other hand, matrix A may be dense, so each multiplication with it will be done using a simple dynamic programming procedure.

4.1 A: Aggregating demands

Recall, each vertex of each layer graph is formed from the contraction of one or more edges from G. The flow modeled by our linear cost approximator sends the net amount of demand within each $v' \in V_i$ to other vertices of G_i. We define the **aggregate demand** of each vertex $v' \in V_i$ to be $b(v') := \sum_{v \in V(v')} b(v)$; our linear cost approximator is based on the cost of flow paths moving these aggregate demands between layer graph vertices. Matrix A computes these aggregate demands so we may subsequently understand the cost of the flow. For any i, for any $v' \in V_i$, and for any input vertex $v \in V$,

$$A(v', v) := [v \in V(v')]$$

where $[Q]$ denotes the $0, 1$-indicator variable for proposition Q. For any demand vector $b \in \mathbb{R}^V$, we have $(Ab)(v') = b(v')$.

▶ **Lemma 4.** *Let $b \in \mathbb{R}^V$ and $b' \in \mathbb{R}^{\cup_i V_i}$. Vectors Ab and $(A)^T b'$ can both be computed in $O(m \log n)$ time.*

Proof. Consider any layer graph vertex v'. If $V(v') = \{v\}$ for some $v \in V$, then $(Ab)(v') = b(v)$. Otherwise, we have $b(v') = \sum_{w \in p^{-1}(v')} b(w)$. We compute all entries $Ab(v)$ in $O(m \log n)$ time by iterating through vertices in *decreasing* order of layer graph index.

Let $v \in V$ be any input graph vertex, and let v' be the unique leaf such that $V(v') = \{v\}$. We have $((A)^T b')(v) = \sum_{w | v \in V(w)} b'(w) = \sum_{w \in p^\infty(v')} b'(w)$. We compute $\sum_{w' \in p^\infty(w)} b'(w')$ for all layer graph vertices w in $O(m \log n)$ time by iterating through vertices in *increasing* order of layer graph index. We then look up the values for the leaves in $O(n)$ additional time. ◀

4.2 R: Routing flow

Matrix R is meant to model an oblivious routing of flows to satisfy the aggregate demands within each layer graph. In Section 5.1, we show how the entries in RAb can be cleanly turned into a flow satisfying b itself. The flows within each layer graph are determined by considering how much demand would be distributed to each potential cluster center in a flow based on a random tree embedding. The hope is that opposing demands of nearby vertices see distribution to common targets, causing the demands to cancel. In turn, the flows in subsequent layers end up routing only relatively light uncanceled portions of the original demands, keeping costs low.

We begin by computing a matrix $D \in \mathbb{R}^{(\cup_i V_i) \times (\cup_i V_i)}$ that takes aggregate demands to their targets after **D**istribution. Fix index i. Consider any $v \in V_i$. Imagine continuously increasing a distance parameter λ starting from 0 and ending at Δ_i. We wish to distribute the demand $b(v)$ of v to members of $B_i(v)$, giving precedence to those vertices in $B_i(v)$ that are closer to v. The total fraction of demand distributed up to each moment λ should be equal to λ / Δ_i.

Fix a moment λ. There is a maximum index j such that $B_i^j(v)$ contains at least one vertex of distance at most λ from v. As λ continues to increase, we will distribute the demand equally among exactly the vertices in $B_i^j(v)$ at distance at most λ from v. This choice of equal distribution models each of those vertices being equally likely to be the center of the first ball of radius λ to contain v. As λ increases, the specific vertices within distance λ will change along with the index j.

We now describe how to compute the total proportion of demand that should be distributed to each vertex in $B_i(v)$. Fix any $j \in \{0, \dots k - 1 = \lg n - 1\}$. For any $\lambda \geq 0$, let $\bar{B}_i^j(v, \lambda) := \left\{ w \in B_i^j(v) \mid \delta_i(v, w) \leq \lambda \right\}$ denote those members of $B_i^j(v)$ that are within distance λ of v.

Let $\lambda^{j+} := \min\left\{\delta(v, B_i^{j+1}(v)), \Delta_i\right\}$ (we define the distance to an empty set such as $B_i^k(v)$ to be $+\infty$). We sort the members of $\bar{B}_i^j(v, \lambda^{j+})$ in increasing order of distance from v in G_i in $O(|B_i^j(v)| \log n)$ time. Let $\langle w_1, w_2, \ldots, w_r\rangle$ be this sorted sequence of vertices. Finally, for each $q \in \{1, \ldots, r\}$, we set

$$D(w_q, v) := \frac{\lambda^{j+} - \delta_i(v, w_r)}{r\Delta_i} + \sum_{\ell=q}^{r-1} \frac{\delta_i(v, w_{\ell+1}) - \delta_i(v, w_\ell)}{\ell\Delta_i}.$$

After sorting, the values $D(w_q, v)$ for a particular i, v, and j can be computed in $O(r) = O(|B_i^j(v)|)$ time as a running suffix sum. All members of D not defined above are set to 0.

Observe for any vertex $v \in (\cup_i V_i)$, $\sum_{w \in (\cup_i V_i)} D(w, v) = 1$. Each non-zero entry of D corresponds to the member of some bundle $B(v)$, so there are at most $O(m \log^2 n)$ of them. Accordingly, we store D explicitly. Accounting for the time needed to sort, D can be constructed in $O(m \log^3 n)$ time total.

After computing D, we are able to compute R itself. For a given vertex $v \in V_i$ with parent $p(v) \in V_{i'}$ and $w \in V_i$, we model routing the portion of v's aggregate demand distributed to w to all $w' \in V_{i'}$ proportionally to how much of $p(v)$'s demand should be distributed to the various w'. More concisely, we set

$$R((w, w'), v) := D(w', p(v)) \cdot D(w, v).$$

Finally, we need to route the aggregate demand of vertices in G_0. Let $s \in V_0$ be arbitrarily chosen. For each $v, w \in V_0$, we set

$$R((w, s), v) := D(w, v).$$

Recall, each bundle $B_i(v)$ has size at most $O(\log^2 n)$. Therefore, the total number of non-zero entries in R is at most $O(m \log n) \cdot O(\log^2 n) \cdot O(\log^2 n) = O(m \log^5 n)$. Again, we store the matrix explicitly.

4.3 C: Estimating flow costs

We now compute the matrix C whose job it is to estimate the costs of the flows described by RAb. As we shall see in the next section, these flows can follow relatively short paths so that the cost per unit flow sent within a layer graph G_i is $O(\Delta_i)$. We define C to be a diagonal matrix. In order to keep it sparse, we only give it non-zero entries for pairs (w, w') where row (w, w') of R has at least one non-zero entry. For each such $w \in V_i$ and $w' \in V_{i'}$, with $\Delta_{i'} = 2\Delta_i$, we set

$$C((w, w'), (w, w')) := 3\Delta_{i'},$$

and for each such $w \in V_i$ and $w' \in V_{i'}$, with $i' > 2\Delta_i$, we set

$$C((w, w'), (w, w')) := 2\Delta_i.$$

(Note that $i' = i$ for the case of $w' = s$ as defined above.) As before, the number of non-zero entries is $O(m \log^5 n)$.

Recall, our $O(\log^3 n)$-approximate linear cost approximator $P := CRA$. We can perform matrix-vector multiplications with P and its transpose by applying Lemma 4 when working with A and the standard multiplication algorithm when working with R and C.

▶ **Lemma 5.** *Let $b \in \mathbb{R}^V$ and $b' \in \mathbb{R}^{\cup_i V_i}$. Vectors Pb and $P^T b'$ can both be computed in $O(m \log^5 n)$ time.*

5 Cost approximation analysis

In this section, we establish the approximation ratio $\alpha = O(\log^3 n)$ of the linear cost approximator P. Fix any proper demand vector $b \in \mathbb{R}^V$. We define a flow $f \in \mathbb{R}^{\vec{E}}$ routing b where $c(f) \leq ||Pb||_1$. Then, we prove $||Pb||_1 \leq \alpha \text{OPT}(b)$, giving a concrete expression for α in the process.

5.1 A flow based on P

For each pair of vertices $u, v \in V$, let $\pi(u, v)$ denote the flow sending one unit from u to v along an arbitrary chosen **canonical shortest path** in G. For each layer graph vertex w, we pick an arbitrary representative $r(w) \in V(w)$.

We construct the flow f as follows. Initially, $f = 0^{\vec{E}}$. Then, for each non-zero $R((w, w'), v)$, we add $R((w, w'), v) \cdot b(v) \cdot \pi(r(w), r(w'))$ to f. In words, we send an $R((w, w'), v)$ proportion of the $b(v)$ units of flow demanded by v along the canonical shortest path between the representatives for w and w'. The proofs of the following lemmas appear in the full version.

▶ **Lemma 6.** *Flow $f \in \mathbb{R}^{\vec{E}}$ as defined above routes b.*

▶ **Lemma 7.** *Let $v, w \in V_i$ and $w' \in V_{i'}$. If $R((w, w'), v) \neq 0$ then $c(\pi(r(w), r(w'))) < 3\Delta_{i'}$.*

▶ **Lemma 8.** *We have $c(f) \leq ||Pb||_1$.*

5.2 Approximation ratio

Our approximation ratio upper bound depends on the following lemma, loosely mirroring the fact that a random tree embedding of a graph distorts distances by only a small factor in expection. The lemma essentially states that the closer two vertices sharing a layer graph are to one-another, the less they disagree on where their aggregate demand should be reassigned. Let $H_n = 1/1 + 1/2 + \cdots + 1/n$ denote the nth harmonic number.

▶ **Lemma 9.** *Let $u, v \in V_i$ for some i. We have*

$$\sum_{w \in V_i} |D(w, u) - D(w, v)| < \frac{8\delta_i(u, v) H_n \lg n}{\Delta_i}.$$

Proof sketch. Recall in the construction of D, we consider a continuously increasing $\lambda \in [0, \Delta_i]$. As λ increases at a rate of 1, we increase the value of at least one $D(w, u)$ at a rate of $1/(\ell\Delta_i)$ where ℓ is the number of vertices in a particular set $\bar{B}_i^j(u, \lambda)$. Only vertices $w \in \bar{B}_i^j(u, \lambda) \subseteq B_i^j(u)$ see $D(w, u)$ increase for this value of λ. Let $N(u, \lambda)$ denote the set of vertices for which $D(w, u)$ is increasing for parameter λ, and define $N(v, \lambda)$ similarly.

Considering all λ, we can show

$$\sum_{w \in V_i} |D(w, u) - D(w, v)| \leq \frac{2}{\Delta_i} \int_0^{\Delta_i} \sum_{w \in V_i} \left(\frac{[w \in (N(u, \lambda) \setminus N(v, \lambda))]}{|N(u, \lambda)|} + \frac{[w \in (N(v, \lambda) \setminus N(u, \lambda))]}{|N(v, \lambda)|} \right) d\lambda.$$

Therefore, our goal is to, for each $w \in V_i$, bound the measure of λ that puts w in exactly one of $N(u, \lambda)$ or $N(v, \lambda)$ and then divide by a number at most the size of that same set for the length of those λ.

Fix w, and consider the set of λ such that $w \in (N(u, \lambda) \setminus N(v, \lambda))$. We have two (not mutually exclusive) cases to consider.

1. $\delta_i(u,w) \le \lambda$ but $\delta_i(v,w) > \lambda$: By the triangle inequality, $\delta_i(v,w) - \delta_i(u,w) \le \delta_i(u,v)$. Therefore, the range of λ for which this case can occur has measure at most $\delta_i(u,v)$ as well. Recall, $w \in N(u,\lambda)$ implies $w \in B_i^j(u)$ for some j. Let $\langle w_1, \ldots, w_r \rangle$ denote the vertices of $B_i^j(u)$ sorted by increasing distance from u, and let $w = w_q$. We have $\{w_1, \ldots, w_q\} \subseteq N(u,\lambda)$, so $|N(u,\lambda)| \ge q$.
2. $N(u,\lambda) \subseteq B_i^j(u)$ but $N(v,\lambda) \subseteq B_i^{j'}(v)$ for some $j' \ne j$: This case occurs when $(\delta_i(u,S_i^j) \le \lambda$ and $\delta_i(v,S_i^j) > \lambda)$ or $(\delta_i(u,S_i^{j+1} > \lambda)$ and $\delta_i(v,S_i^{j+1}) \le \lambda)$. As before, the triangle inequality implies $\delta_i(v,S_i^j) - \delta_i(u,S_i^j)$ and $\delta_i(u,S_i^{j+1}) - \delta_i(v,S_i^{j+1})$ are both at most $\delta_i(u,v)$. Therefore, the range of relevant λ for this case has measure at most $2\delta_i(u,v)$.

There are $k = \lg n$ different choices for j in either case. The total measure of λ resulting in case 2. is at most $2\delta_i(u,v)\lg n < \delta_i(u,v)H_n \lg n$. Summing across $w \in V_i$, and noting each index q can be used at most once per choice of j, we have

$$\begin{aligned}
\sum_{w \in V_i} |D(w,u) - D(w,v)| &\le \frac{2}{\Delta_i} \int_0^{\Delta_i} \sum_{w \in V_i} \left(\frac{[w \in (N(u,\lambda) \setminus N(v,\lambda))]}{|N(u,\lambda)|} + \frac{[w \in (N(v,\lambda) \setminus N(u,\lambda))]}{|N(v,\lambda)|} \right) d\lambda \\
&< \frac{2}{\Delta_i} \left(2\delta_i(u,v) H_n \lg n + 2 \lg n \sum_{q=1}^{n} \frac{\delta_i(u,v)}{q} \right) \\
&= \frac{8\delta_i(u,v) H_n \lg n}{\Delta_i}.
\end{aligned}$$

◀

We set $\alpha := 180 H_n \lg^2 n = O(\log^3 n)$.

▶ **Lemma 10.** *We have* $||Pb||_1 \le \alpha \cdot OPT(b)$.

Proof. Let f^* be an optimal flow with $c(f^*) = \mathrm{OPT}(b)$. Standard flow theory implies there exists a decomposition of f^* into a linear combination $a_1 f_1 + a_2 f_2 + \ldots$ of unit path flows such that each $a_j > 0$ and $c(f^*) = \sum_j a_j c(f_j)$. We will charge each of the non-zero terms in Pb to one or more of these path flows and argue that each flow f_j is charged at most $\alpha a_j c(f_j)$.

Fix f_j. Let $u \in V$ and $v \in V$ be the source and sink of f_j, respectively. We have $c(f_j) \ge \delta(u,v)$ (in fact, equal). We consider charges based on a_j units in $b(u)$ and $-a_j$ units in $b(v)$. These charges are handled in a few cases that are not necessarily exclusive.

Consider any (i',u') where $p(u') \in V_{i'}$ and $u \in V(u')$, and $\Delta_{i'} \le \delta(u,v)$. By construction of R, $\sum_{w \in (\cup_{i>i'} V_i)} \sum_{w' \in V_{i'}} R((w,w'),u') = 1$. Summing over all such pairs (i',u'), we see

$$\begin{aligned}
\sum_{(i',u') | p(u') \in V_{i'} \wedge u \in V(u') \wedge \Delta_{i'} \le \delta(u,v)} \; \sum_{w \in (\cup_{i>i'} V_i)} \sum_{w' \in V_{i'}} 3\Delta_{i'} R((w,w'),u') &= \sum_{i' | \Delta_{i'} \le \delta(u,v)} 3\Delta_{i'} \\
&\le 6\delta(u,v) \\
&< 2\delta(u,v) H_n \lg^2 n.
\end{aligned}$$

In words, the a_j units of demand from u contribute at most $2a_j\delta(u,v)H_n \lg^2 n$ total among various $|(Pb)((w,w'))|$ where $w' \in V_{i'}$ with $\Delta_{i'} \le \delta(u,v)$. We charge $2a_j\delta(u,v)H_n \lg^2 n$ to f_j for these contributions.

Now, consider any (i',u') where $p(u') \in V_{i'}$, $u \in V(u')$, and $\Delta_{i'} > \delta(u,v)$. Having $\Delta_{i'} > \delta(u,v)$ implies either there is a path from $p(u')$ to a distinct contracted set of vertices in G_i containing v or $v \in V(p(u'))$ as well. Let v' such that $p(v') \in V_{i'}$ with $v \in V(v')$.

Suppose either u' or v' is in some V_i with $\Delta_i < \Delta_{i'}/2$. Lemma 3's guarantee of long incident edges for $p(u')$ and $p(v')$ implies $p(v') = p(u')$, and no other vertex of $V_{i'}$ lies within $\Delta_{i'}$ of them. Therefore, $D(p(v'), v) = D(p(v'), u') = 1$, and

$$\begin{aligned}
&\sum_{w \in V_i} \sum_{w' \in V_{i'}} |R((w,w'),u') - R((w,w'),v')| \\
&\qquad = \sum_{w \in V_i} \sum_{w' \in V_{i'}} |D(w', p(u'))D(w,u') - D(w', p(v'))D(w,v')| \\
&\qquad = \sum_{w \in V_i} |D(w,u') - D(w,v')| \\
&\qquad \leq \frac{8\delta(u,v)H_n \lg n}{\Delta_i} \\
&\qquad \leq \frac{4\delta(u,v)H_n \lg^2 n}{\Delta_i}.
\end{aligned}$$

Therefore, all but $4a_j\delta(u,v)H_n \lg^2 n/\Delta_i$ units of the a_j demand from u contributing to various $|(Pb)(w,w')|$ of this sort are canceled by opposite demand from v. Each unit is multiplied by $2\Delta_i$, so we charge $8a_j\delta(u,v)H_n \lg n$ for these contributions.

This subcase can only occur once, because for smaller choices of i', vertices u', v', and their parents have identical aggregate demand distributions, implying *all* a_j units of demand from u are cancelled by opposite demand from v.

If the above subcase does not occur, then both $p(u')$ and $p(v')$ belong to the specific common V_i with $\Delta_i = \Delta_{i'}/2$. By Lemma 9 and the fact that $|ab - cd| \leq |a - c| + |b - d|$ for $a, b, c, d \in [0, 1]$,

$$\begin{aligned}
&\sum_{w \in V_i} \sum_{w' \in V_{i'}} |R((w,w'),u') - R((w,w'),v')| \\
&\qquad = \sum_{w \in V_i} \sum_{w' \in V_{i'}} |D(w', p(u'))D(w,u') - D(w', p(v'))D((w,v')| \\
&\qquad \leq \sum_{w \in V_i} \sum_{w' \in V_{i'}} (|D(w', p(u')) - D(w', p(v'))| + |D(w,u') - D(w,v')|) \\
&\qquad \leq \frac{8\delta(u,v)H_n \lg n}{\Delta_{i'}} + \frac{8\delta(u,v)H_n \lg n}{\Delta_{i'}/2} \\
&\qquad \leq \frac{24\delta(u,v)H_n \lg n}{\Delta_{i'}}.
\end{aligned}$$

Therefore, all but $24a_j\delta(u,v)H_n \lg n/\Delta_{i'}$ units of the a_j demand from u contributing to various $|(Pb)(w,w')|$ of this sort are canceled by opposite demand from v. Each unit is multiplied by $3\Delta_{i'}$, so we charge $72a_j\delta(u,v)H_n \lg n$ for these contributions.

The contractions used in building the layer graphs guarantees there is at most one choice of i' where $\Delta_{i'} \geq n \cdot \delta(u,v)$ while $u' \neq v'$. For smaller choices of i', vertices u', v', and their parents have identical aggregate demand distributions, implying *all* a_j units of demand from u are cancelled by opposite demand from v. There are at most $\lg n$ different values i' with large reach that result in any non-zero charge, bringing this set of charges to a total of $72a_j\delta(u,v)H_n \lg^2 n$.

Finally, we consider $u' \in V_0$, $v' \in V_0$, $u \in V(u')$, and $v \in V(v')$. We have

$$\sum_{w \in V_0} |D(w,u') - D(w,v')| \leq \frac{8\delta(u,v)H_n \lg n}{\Delta_0}.$$

Similar to before, all but $8a_j\delta(u,v)H_n \lg n/\Delta_0$ units of the a_j demand from u are canceled by opposite demand from v. Multiplying by $2\Delta_0$, we charge $16a_j\delta(u,v)H_n \lg n \leq 8a_j\delta(u,v)H_n \lg^2 n$.

Summing over all types of charges and then doubling the value for the contributions of v to various $|(Pb)(w,w')|$, we get a total charge of $180a_j\delta(u,v)H_n \lg^2 n \leq \alpha \cdot a_j c(f_j)$ to f_i. All units of demand for all $u, v \in V$ are considered throughout these charges, so we have charged at least $||Pb||_1$ in total. On the other hand, we charge at most $\sum_j \alpha \cdot a_j c(f_j) = \alpha \cdot c(f^*) = \alpha \cdot \mathrm{OPT}(b)$ in total. ◀

6 Approximation scheme

We are now ready to present our main theorem.

▶ **Theorem 11.** *There exists a deterministic algorithm that given an undirected graph $G = (V, E)$ over n vertices and m edges, positive edge costs $c \in \mathbb{R}^E_{>0}$, a proper set of demands $b \in \mathbb{R}^V$, and a parameter $\varepsilon > 0$ computes a flow f routing b with $c(f) \leq (1+\varepsilon)OPT(d)$ in $O(\varepsilon^{-2} m \log^{O(1)} n)$ time.*

Proof. We begin by constructing the layer graphs as presented in Section 3 along with the Thorup-Zwick distance oracles for each in $O(m \log^3 n)$ time total. We construct sparse representations of the matrices R and C in $O(m \log^5 n)$ time so they can be used in matrix-vector multiplications with the $O(\log^3 n)$-approximate linear cost approximator P and its transpose. Each matrix-vector multiplication takes $O(m \log^5 n)$ time. We perform $O(\varepsilon^{-2} \log^{O(1)} n)$ multiplications with P according to the boosting framework of Zuzic [18, Corollaries 12 and 16] to find an infeasible flow f' of cost $c(f') \leq (1+\varepsilon/2)\mathrm{OPT}(b)$ where $\mathrm{OPT}(b - I_G f') \leq \mathrm{OPT}(b)/n^2$. Finally, we add in an n-approximate solution that routes $b - I_G f'$ as in [15] in $O(m \log n)$ additional time. The total cost of the flow returned is $(1+\varepsilon/2+1/n)\mathrm{OPT}(b) \leq (1+\varepsilon)\mathrm{OPT}(b)$. ◀

References

1 Alexandr Andoni, Clifford Stein, and Peilin Zhong. Parallel approximate undirected shortest paths via low hop emulators. In *Proc. 52nd Ann. ACM SIGACT Symp. Theory of Comput.*, pages 322–335, 2020.

2 Ruben Becker, Sebastian Forster, Andreas Karrenbauer, and Christoph Lenzen. Near-optimal approximate shortest paths and transshipment in distributed and streaming models. *SIAM J. Comput.*, 50(3):815–856, 2021.

3 Jean Bourgain. On Lipschitz embedding of finite metric spaces in Hilbert space. *Israel J. Math.*, 52(1–2), 1985.

4 J. Brand, L. Chen, R. Peng, R. Kyng, Y. P. Liu, M. Gutenberg, S. Sachdeva, and A. Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *Proc. 64th IEEE Ann. Symp. Found. Comput. Sci.*, pages 503–514, 2023.

5 Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *Proc. 63rd IEEE Symp. Found. Comput. Sci.*, pages 612–623, 2022.

6 Samuel I. Daitch and Daniel A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proc. 40th Ann. ACM Symp. Theory Comput.*, pages 451–460, 2008.

7 Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.

8 Emily Fox and Jiashuai Lu. A deterministic near-linear time approximation scheme for geometric transportation. In *Proc. 64th IEEE Ann. Symp. Found. Comput. Sci.*, pages 1301–1315, 2023.

9 Kyle Fox and Jiashuai Lu. A near-linear time approximation scheme for geometric transportation with arbitrary supplies and spread. *J. Comput. Geom.*, 13(1):204–225, 2022.

10 Andrey Boris Khesin, Aleksandar Nikolov, and Dmitry Paramonov. Preconditioning for the geometric transportation problem. *J. Comput. Geom.*, 11(2):234–259, 2021.

11 Jason Li. Faster parallel algorithm for approximate shortest path. In *Proc. 52nd Ann. ACM SIGACT Symp. Theory of Comput.*, pages 308–321, 2020.

12 James B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations Research*, 41(2):338–350, 1993.

13 Liam Roditty, Mikkel Thorup, and Uri Zqick. Deterministic constructions of approximate distance oracles and spanners. In *Proc. 32nd Int. Colloq. Automata Lang. Prog.*, pages 261–272, 2005.

14 Václav Rozhoň, Christoph Grunau, Bernhard Haeupler, Goran Zuzic, and Jason Li. Undirected $(1+\varepsilon)$-shortest paths via minor-aggregates: Near-optimal deterministic parallel and distributed algorithms. In *Proc. 54th Ann. ACM Symp. Theory Comput.*, STOC 2022, pages 478–487, New York, NY, USA, 2022. Association for Computing Machinery.

15 Jonah Sherman. Generalized preconditioning and undirected minimum-cost flow. In *Proc. 28th Ann. ACM-SIAM Symp. Discrete Algorithms*, pages 772–780, 2017.

16 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, January 2005.

17 Cédric Villani. *Optimal Transport: Old and New*. Springer Science & Business Media, 2008.

18 Goran Zuzic. A simple boosting framework for transshipment. In *Proc. 31st Ann. Europ. Symp. Algorithms*, pages 104:1–104:14, 2023.

19 Goran Zuzic, Gramoz Goranci, Mingquan Ye, Bernhard Haeupler, and Xiaorui Sun. Universally-optimal distributed shortest paths and transshipment via graph-based ℓ_1-oblivious routing. In *Proc. 2022 Ann. ACM-SIAM Symp. Discrete Algorithms*, pages 2549–2579, 2022.