A simple deterministic near-linear time approximation scheme for transshipment with arbitrary positive edge costs

Emily Fox*

July 14, 2023

Abstract

We describe a simple deterministic near-linear time approximation scheme for uncapacitated minimum cost flow in undirected graphs with 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 in two main ways: 1) to embed the problem instance into low-dimensional space and 2) to randomly pick target locations to send flow so nearby opposing demands can be satisfied. 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 limit the available routing decisions using distances explicitly stored in the well-known Thorup-Zwick distance oracle.

*Department of Computer Science, The University of Texas at Dallas; emily.fox@utdallas.edu. 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***. 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 [Vil08], 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 [KNP21, FL22, BFKL21, ASZ20, Li20, ZGY+22, RGH+22, FL23a, FL23b]. 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 [Orl93] 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 [DS08] and culminating in the very recent ***almost-linear*** $m^{1+o(1)} \log^2 U$ time algorithm of [CKL+22].

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 [She17] 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 instead 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 [Li20] and [ASZ20] proposed 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 [FL23a, FL23b] proposed near-linear $O(\varepsilon^{-2}m\log^{O(1)}n)$ time approximation schemes *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 [Li20, ASZ20, BFKL21, RGH+22, ZGY+22].

Perhaps unsurprisingly, the algorithms of [ASZ20] and [FL23a, FL23b] use the aforementioned framework of Sherman [She17] 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 [Zuz23a, Zuz23b] 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 revealed 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 heavily rely on randomization. In particular, they all compute random Bourgain [Bou85] 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 [RGH+22] describing an $O(\varepsilon^{-2}m\log^{O(1)}(nU))$ time *deterministic* approximation scheme. However, unlike some of the results mentioned above [She17, FL23a, FL23b], its running time is still polylogarithmic in the aspect ratio of the edge costs.

1.2 Our results

We present the first deterministic near-linear time approximation scheme for transshipment with a running time that is *not* dependent on the aspect ratio of the edge costs. Specifically, our algorithm computes a flow f that routes b of cost $c(f) \leq (1+\varepsilon)\text{OPT}(b)$ in $O(\varepsilon^{-2}m\log^{O(1)}n)$ time. It is also (in our opinion) simpler and likely easier to implement than all previous near-linear time approximation schemes for transshipment, even ignoring the considerable extra work many of them require to function in parallel or distributed settings. In fact, other than using the aforementioned transportation boosting framework of Zuzic [Zuz23a, Zuz23b] and a deterministic construction of the well-known distance oracle of Thorup and Zwick [TZ05, RTZ05], the entire presentation of our algorithm and its analysis is self-contained and presented within this 15 page report. 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 One method of instantiating the transshipment boosting framework 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 with the property that for *any* set of proper demands b, we have $\text{OPT}(b) \leq ||Pb||_1 \leq \alpha\text{OPT}(b)$. 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 $c(f') \leq (1+\varepsilon/2)\text{OPT}(b)$ and $\text{OPT}(b - I_G f') \leq \text{OPT}(b)/n^2$ in $O(\varepsilon^{-2}\alpha^2 M\log^{O(1)}n)$

time [Zuz23b, Corollaries 12 and 16]. We can then route an n-approximate flow for demands $b - I_G f'$ 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 largely motivated by recent work of Fox and Lu [FL23a, FL23b] where they deterministically build a preconditioner for a special case of transshipment that arises in approximation schemes for the *geometric transportation problem* in low-dimensional Euclidean space. Here, the goal is to compute a weighted matching between several pairs of points of minimum total distance. Recall, most of the previous approximation schemes for transshipment compute a random Bourgain [Bou85] embedding of the shortest path metric into low dimensional space. In particular, the authors of [ASZ20] use the embedding to reduce transshipment to the geometric transportation problem. They then use randomly shifted grids to provide a hierarchy of subsets of points. Their preconditioner is based on the cost of greedily matching points within lower levels of the hierarchy before moving to the top, and the expected cost of the process compares favorably enough to $\mathrm{OPT}(b)$ for their approach to work.

The main insight of Fox and Lu [FL23a, FL23b] in their algorithm for transportation is that instead of building a hierarchy (or quadtree) based on randomly shifted grids, it suffices to build a collection of deterministic grids and explicitly compute the net expected amount of demand within each grid cell, *as if they had been randomly shifted*. Their preconditioner (modulo appropriate scaling) simply outputs the diameter of each grid cell times the expected demand it contains.

Instead of using a Bourgain embedding, our approximation scheme skips straight to considering the hierarchies formed from random embeddings into dominating *tree metrics* [FRT04] 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 [FRT04]. 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 $C_i \in V$ 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 for each non-empty C_i rooted at 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 Pb 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 That said, we cannot afford to compute the $\Omega(n^2)$ distances required to properly compute these expected values. Instead, we perform a deterministic construction the well-known distance oracle of Thorup and Zwick [TZ05, RTZ05]. Let $k \geq 1$ be an integer. After $O(kmn^{1/k} \log^{O(1)} n)$ time preprocessing, their $O(kn^{1+1/k})$-space oracle can compute $2k-1$-approximate distances between any pair of vertices in $O(k)$ time. By setting $k := \lg n$, we get

an $O(\log n)$-approximate distance oracle with construction time $O(m \log^{O(1)} n)$ and size $O(n \log n)$. We never actually use the oracle for its stated purpose, however. 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 set of $O(n \log n)$ minors of G, each maintaining a constant factor 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 $O(m \log n)$. 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, bring the approximation ratio of P to a relatively small $O(\log^3 n)$.

Oblivious routing and comparison to [RGH+22] 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 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. To help make the algorithm as a set of rules to follow easier to understand, we tend to stick more closely with this oblivious routing/reassignment of demand interpretation throughout the rest of the paper. We describe this flow in more detail in the analysis of P. 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.

This oblivious routing of actual flow interpretation/approach is used much more heavily in the $O(\varepsilon^{-2} m \log^{O(1)}(nU))$ time deterministic approximation scheme of [RGH+22], 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 [RGH+22] is made more complicated by, for example, them only sending flow between centers of nesting clusters. As a result, their analysis, while similar to ours, appears a fair bit more complicated.

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.

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 [TZ05] 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 $O(kmn^{1+1/k} \log^{O(1)} n)$ time [RTZ05]. 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 $V = A^0 \supseteq A^1 \supseteq \cdots \supseteq A^k = \emptyset$ we refer to as ***samples***. We have $A^{k-1} \neq \emptyset$. 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*** $B^j(v) = \left\{w \in A^j \mid \delta(v, w) < \delta(v, A^{j+1})\right\}$. In particular, $B^j(v) \subseteq A^j \setminus A^{j+1}$.

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 [RTZ05] 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 isolated (degree 0) vertices, no loops, and no 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, we define 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 after the edge deletions. 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. Let $\Delta_0 := \delta(s, x) + \delta(s, y)$. Reach Δ_0 is at least, but no more than twice, the diameter of G. Given Δ_{i-1}, we define 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 $e^{||} = \arg\max \{e \in E \mid c(e) \leq \Delta_{i-1}/n\}$ be the costliest contracted edge of G_{i-1}. If $e^{||}$ is well-defined, let $\Delta_i := c(e^{||}) \cdot n$. 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.

Lemma 3.1. *The graphs $\langle G_0, G_1, \ldots, G_L \rangle$ have at most $O(m \log n)$ edges and vertices in total.*

Proof: Let $e \in E$ be any edge of G, and let G_j be the first graph in the sequence to contain e. Because e is not contracted in G_j, we have $\Delta_j/n \leq c(e)$. As stated, $\Delta_i \leq \Delta_{i-1}/2$ for all $i \geq 1$. For all $i \geq j + \lg n + 2$, we have $\Delta_i \leq \Delta_j/(4n) < c(e)/2$ and e is deleted from G_i. The edge e exists in at most $1 + \lg n$ graphs in the sequence, implying the sequence of graphs contains $O(m \log n)$ edges in total. The graphs contain no isolated vertices, so the total number of vertices is also $O(m \log n)$. □

Lemma 3.2. *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.*

Proof: Reach Δ_0 can be computed in $O(m \log n)$ time by running Dijkstra's algorithm [Dij59] with a binary heap to compute shortest paths. We store each edge of cost at most Δ_0/n in a binary heap keyed by cost, contract these edges, and delete edges of cost more than $2\Delta_0$ in $O(m)$ additional time. We now iteratively compute (G_i, Δ_i) for each $i \geq 1$. Suppose we've computed (G_{i-1}, Δ_{i-1}). We compute Δ_i in $O(1)$ time by peeking at the top of the binary heap if necessary. We repeatedly remove edges from the binary heap that should belong to G_i but not G_{i-1}. We then uncontract these edges and delete costly edges from G_{i-1} to create G_i in $O(m_{i-1} + m_i)$ time. We add parent pointers during this time as well. After all layers have been constructed, we loop over each vertex one more time to add a child pointer from their parent. In total, we spend $O(m \log n)$ time removing edges from the binary heap and $O(m \log n)$ time constructing the individual graphs (Lemma 3.1). □

Lemma 3.3. *Let $v \in V_i$ for $i > 0$, and let $p(v) \in V_{i'}$. If $i' < i - 1$, then 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 $V(p(v)) < 2\Delta_i$.*

Proof: Suppose $i' < i-1$. All edges leaving $p(v)$ are deleted in layer $i'+1$, so they have length strictly greater than $2(\Delta_{i'}/2) = \Delta_{i'}$. Children of $p(v)$ appear only when the reach is small enough to uncontract some of their edges. No other edges are added except those being uncontracted, so the entire connected component consists of children of $p(v)$. Finally, each edge of this component has length at most $(2\Delta_i)/n$ or some would have uncontracted one layer earlier. A shortest path between any two vertices of the component has length at most $(n-1)\cdot(2\Delta_i)/n < 2\Delta_i$. □

Along with each layer G_i, we construct a Thorup-Zwick distance oracle (Section 2) with parameter $k := \lg n$. Let A_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 3.1, 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 $O(m \log^{O(1)} n)$ 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 $P_1 \in \mathbb{R}^{(\cup_i V_i)\times V}$, $P_2 \in \mathbb{R}^{((\cup_i V_i)\times(\cup_i V_i))\times(\cup_i V_i)}$, and $P_3 \in \mathbb{R}^{((\cup_i V_i)\times(\cup_i V_i))\times((\cup_i V_i)\times(\cup_i V_i))}$ and let $P := P_3 P_2 P_1$ be their product. Each of these three matrices serves a clear purpose in a three-step process of obtaining a cost approximation. Matrices P_2 and P_3 are sparse and can be built and stored explicitly by our algorithm. On the other hand, matrix P_1 may be dense, so each multiplication with it will be done using a simple dynamic programming procedure.

4.1 P_1: Aggregating demands

Recall, each vertex of each layer graph is formed from the contraction of one or more edges from G. We define the ***aggregate demand*** of a vertex $v' \in V_i$ to be $b(v') := \sum_{v \in V(v')} b(v)$. Matrix P_1 simply computes these aggregate demands. For any i, for any $v' \in V_i$, and for any input vertex $v \in V$,

$$P_1(v', v) := [v \in V(v')]$$

where $[Q]$ denotes the 0, 1-indicator variable for proposition Q. For any demand vector $b \in \mathbb{R}^V$, we have $(P_1 b)(v') = b(v')$.

Lemma 4.1. *Let $b \in \mathbb{R}^V$ and $b' \in \mathbb{R}^{\cup_i V_i}$. Vectors $P_1 b$ and $(P_1)^T b'$ 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 $(P_1 b)(v') = b(v)$. Otherwise, we have $b(v') = \sum_{w \in p^{-1}(v')} b(w)$. We compute all entries $P_1 b(v)$ 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 $((P_1)^T b')(v) = \sum_{w | v \in V(w)} b'(w) = \sum_{w \in p^\infty(v')} b'(w)$. 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 P_2: Reassigning aggregate demands

Matrix P_2 is meant to capture the probabilistic reassignment of aggregate demands within individual layers along with the cost of moving the demand between layers. Intuitively, we imagine reassigning units of aggregate demand from each vertex of a layer to several nearby vertices within the same layer. The hope is that opposite demands will be reassigned to common vertices, cancelling them out. Matrix P_2 determines the total amount of demand reassigned to each vertex.

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 reassign 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 reassigned up to each moment λ will 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 reassign the demand equally among exactly the vertices in $B_i^j(v)$ at distance at most λ from v. As λ increases, the specific vertices within distance λ and j itself will change. We need to compute these moments of change in order to figure out how much demand should be reassigned to various vertices.

We now describe how to compute the total proportion of demand that should be reassigned 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 $O(|B_i^j(v)| \log\log n)$ time. Let $\langle w_1, w_2, \dots, w_r \rangle$ be this sorted sequence of vertices. Finally, for each $q \in \{1, \dots, r\}$, we set

$$P_2((w_q, w_q), v) := \frac{\lambda^{j+} - \delta_i(v, w_r)}{r\Delta_i} + \sum_{\ell=q}^{r-1} \frac{\delta_i(v, w_{\ell+1}) - \delta_i(v, w_\ell)}{(\ell - q + 1)\Delta_i}.$$

After sorting, the values $P_2((w_q, w_q), v)$ for a particular i, v, and j can be computed in $O(r) = O(|B_i^j(v)|)$ time as a running suffix sum.

In addition to recording the proportion of demand that should be reassigned to each vertex, we record the amount of demand traveling between pairs of vertices in (nearly) adjacent layers. Let $v \in V_i$. For each $w \in B_i(v)$, for each $v' \in p^{-1}(v)$, let $v' \in V_{i'}$. For each $w' \in B_{i'}(v')$, we set

$$P_2((w', w), v') := P_2((w', w'), v') \cdot P_2((w, w), v).$$

All members of P_2 not defined above are set to 0. Observe for any vertex $v \in (\cup_i V_i)$, $\sum_{w \in (\cup_i V_i)} P_2((w, w), v) = 1$. Again, matrix P_2 is sparse, so we explicitly store its $O(m \log n) + O(m \log n) \cdot O(\log^2 n) \cdot O(\log^2 n) = O(m \log^5 n)$ non-zero entries. Summing over all variables above and including the time to compute values for pairs w, w', the total time spent computing P_2 is $O(m \log^5 n)$.

4.3 P_3: Charging for what remains

As suggested above, the reassignment of aggregate demands described by P_2 should result in some cancelling of positive and negative demands. However, the portions of demand that have not been cancelled must be realized by actual costly flow. As we shall see in the next section, these flows can follow relatively short paths so that the cost per unit of uncanceled demand within a layer graph G_i is $O(\Delta_i)$. We define P_3 to be a diagonal matrix. For each w', w with $w' \in V_{i'}$ and $w \in V_i$, we

do the following. If $w' = w$ and $i = 0$, then

$$P_3((w', w), (w', w)) := \Delta_0.$$

If $w' \neq w$ and $\sum_{v \in V_i} P_2((w', w), v) \neq 0$, then

$$P_3((w', w), (w', w)) := 6\Delta_{i'}.$$

All other entries are 0. The number of non-zero entries is $O(m \log^5 n)$.

Recall, our $O(\log^3 n)$-approximate linear cost approximator $P := P_3 P_2 P_1$. We can perform matrix-vector multiplications with P and its transpose by applying Lemma 4.1 when working with P_1 and the standard multiplication algorithm when working with P_2 and P_3.

Lemma 4.2. *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

We construct the flow f as follows. Initially, $f = 0^{\vec{E}}$. For each pairs of vertices $u, v \in V$, let $\pi(u, v)$ denote any flow sending one unit from u to v along a shortest path in G. For each layer graph vertex w, we pick an arbitrary representative $r(w) \in V(w)$.

For each $i \in \{L, \ldots, 0\}$, for each $v, w \in V_i$, we do the following. We add $\sum_{v' \in p^{-1}(v)} \sum_{w' \in (\cup_{i' > i} V_{i'})} P_2((w', w), v') b(v') \cdot \pi(r(w'), r(w))$ to f. In words, we take the demand reassigned by each child $v' \in \mathit{children}(v)$ to some vertex w' and route it proportionally to the vertices w for which we are reassigning the demand of v. The demands are only routed along the canonical shortest path chosen for each pair of vertices.

Once we are done iterating over all layer graph vertices, we do a final cleanup step. Let $s \in V$ be an arbitrary vertex. For each $v \in V_0$ and $w \in V_0$, we add $P_2((w, w), v) b(v) \cdot \pi(r(w), s)$ to f.

Lemma 5.1. *Flow $f \in \mathbb{R}^{\vec{E}}$ as defined above routes b.*

Proof: By construction, for any $v \in V_i$, we have $\sum_{w \in V_i} P_2((w, w), v) = 1$. Also, $\sum_{v' \in p^{-1}(v)} b(v') = b(v)$. Together, these facts imply the shortest path flows for v send a total of $b(v)$ units of flow to various targets w. Further, the total amount taken from sources w' for each $v' \in p^{-1}(v)$ is equal to $b(v')$.

The only targets w that do not have their flow carried further along by processing vertices v' in lower index layers are those $w \in V_0$. Each receives $\sum_{v \in V_0} P_2((w, w), v) b(v)$ units of flow total, which are then carried to s. Vertex s receives a net of $\sum_{v \in V_0} b(v) = 0$ units from these paths.

Finally, each vertex $v \in G$ appears in exactly one layer graph leaf where it sends $b(v)$ units out along one of the shortest path flows. □

Lemma 5.2. *Let $v \in V_i$, $v' \in p^{-1}(v) \cap V_{i'}$ for some i', $w \in V_i$, and $w' \in V_{i'}$. If $P_2((w, w), v)$ and $P((w', w'), v')$ are both non-zero, then $c(\pi(r(w'), r(w))) < 6\Delta_{i'}$.*

Proof: If $i' > i+1$, then Lemma 3.3 states $\delta(r(w'), r(w)) < 2\Delta_{i'} < 6\Delta_{i'}$.

Suppose $i' = i+1$. Value $P((w', w'), v') > 0$ only if $\delta_{i'}(v', w') \leq \Delta_{i'}$, and $P((w, w), v) > 0$ only if $\delta_i(v, w) \leq \Delta_i = 2\Delta_{i'}$. The projection of any path in G_i to G includes some edges of E_i in addition to at most $n-1$ additional contracted edges of total length at most $(n-1) \cdot \Delta_i / n < 2\Delta_{i'}$. Similarly, projecting paths in $G_{i'}$ to G increases the total length by at most $(n-1) \cdot \Delta_{i'} / n < \Delta_{i'}$. By the triangle inequality, $c(\pi(r(w'), r(w))) < \delta_{i'}(w', v') + \Delta_{i'} + \delta_i(v, w) + 2\Delta_{i'} \leq 6\Delta_{i'}$. □

Lemma 5.3. *We have* $c(f) \leq ||Pb||_1$.

Proof: We send exactly $\sum_{v' \in (\cup_i V_i)} P_2((w', w), v') = (P_2 P_1 b)((w', w))$ units of flow in any shortest path between two vertices $r(w')$ and $r(w)$. Let $w' \in V_{i'}$. By Lemma 5.2, this flow costs at most $|6\Delta_{i'}(P_2 P_1 b)((w', w))| = |(Pb)((w', w))|$ to send. Also, for each $w \in V_0$, we send $\sum_{v \in V_0} P_2((w, w), v) = (P_2 P_1 b)((w, w))$ units of flow to s at cost at most $|\Delta_0 (P_2 P_1 b)((w, w))| = |(Pb)((w, w))|$. □

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 $H_o = 1/1 + 1/2 + \cdots + 1/o$ denote the oth harmonic number.

Lemma 5.4. *Let* $u, v \in V_i$ *for some* i. *We have*

$$\sum_{w \in V_i} |P_2((w, w), u) - P_2((w, w), v)| < \frac{8\delta_i(u, v) H_n \lg n}{\Delta_i}.$$

Proof: Recall in the construction of P_2, 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 $P_2(w, u)$ 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 $P_2(w', u)$ increase for this value of λ. Let $C(u, \lambda)$ denote the set of vertices for which $P_2(w, u)$ is increasing for parameter λ, and define $C(v, \lambda)$ similarly. For a particular $w \in V_i$, $|P_2(w, u) - P_2(w, v)|$ is increasing at a rate of no more than $|[w \in C(u, \lambda)]/(|C(u, \lambda)|\Delta_i) - [w \in C(v, \lambda)]/(|C(v, \lambda)|\Delta_i)|$. Suppose $1 \leq |C(u, \lambda)| \leq |C(v, \lambda)|$. Then, the total rate of increase in difference among all $w \in V_i$

for a particular λ is at most

$$\begin{aligned}
\sum_{w\in V_i}\left|\frac{[w\in C(u,\lambda)]}{\Delta_i|C(u,\lambda)|}-\frac{[w\in C(v,\lambda)]}{\Delta_i|C(v,\lambda)|}\right| &= \frac{1}{\Delta_i}\left(\begin{array}{l}\dfrac{|C(u,\lambda)\setminus C(v,\lambda)|}{|C(u,\lambda)|}+\dfrac{|C(v,\lambda)\setminus C(u,\lambda)|}{|C(v,\lambda)|}\\ \quad+|C(u,\lambda)\cap C(v,\lambda)|\left(\dfrac{1}{|C(u,\lambda)|}-\dfrac{1}{|C(v,\lambda)|}\right)\end{array}\right)\\
&= \frac{1}{\Delta_i}\left(\begin{array}{l}\dfrac{|C(u,\lambda)\setminus C(v,\lambda)|}{|C(u,\lambda)|}+\dfrac{|C(v,\lambda)\setminus C(u,\lambda)|}{|C(v,\lambda)|}\\ \quad+\dfrac{|C(u,\lambda)\cap C(v,\lambda)|(|C(v,\lambda)|-|C(u,\lambda)|)}{|C(u,\lambda)||C(v,\lambda)|}\end{array}\right)\\
&\le \frac{1}{\Delta_i}\left(\begin{array}{l}\dfrac{|C(u,\lambda)\setminus C(v,\lambda)|}{|C(u,\lambda)|}+\dfrac{|C(v,\lambda)\setminus C(u,\lambda)|}{|C(v,\lambda)|}\\ \quad+\dfrac{|C(u,\lambda)||C(v,\lambda)\setminus C(u,\lambda)|}{|C(u,\lambda)||C(v,\lambda)|}\end{array}\right)\\
&= \frac{1}{\Delta_i}\left(\frac{|C(u,\lambda)\setminus C(v,\lambda)|}{|C(u,\lambda)|}+\frac{2|C(v,\lambda)\setminus C(u,\lambda)|}{|C(v,\lambda)|}\right)\\
&\le \frac{2}{\Delta_i}\left(\frac{|C(u,\lambda)\setminus C(v,\lambda)|}{|C(u,\lambda)|}+\frac{|C(v,\lambda)\setminus C(u,\lambda)|}{|C(v,\lambda)|}\right)\\
&= \frac{2}{\Delta_i}\sum_{w\in V_i}\left(\begin{array}{l}\dfrac{[w\in(C(u,\lambda)\setminus C(v,\lambda))]}{|C(u,\lambda)|}\\ \quad+\dfrac{[w\in(C(v,\lambda)\setminus C(u,\lambda))]}{|C(v,\lambda)|}\end{array}\right),
\end{aligned}$$

and this same bound holds when $|C(v,\lambda)|<|C(u,\lambda)|$.

Considering all λ, we see

$$\sum_{w\in V_i}|P_2(w,u)-P_2(w,v)|\le\frac{2}{\Delta_i}\int_0^{\Delta_i}\sum_{w\in V_i}\left(\begin{array}{l}\dfrac{[w\in(C(u,\lambda)\setminus C(v,\lambda))]}{|C(u,\lambda)|}\\ \quad+\dfrac{[w\in(C(v,\lambda)\setminus C(u,\lambda))]}{|C(v,\lambda)|}\end{array}\right)d\lambda.$$

Therefore, our goal is to, for each $w\in V_i$, bound the measure of λ that puts w in exactly one of $C(u,\lambda)$ or $C(v,\lambda)$ 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 $w\in(C(u,\lambda)\setminus C(v,\lambda))$. 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, $w\in C(u,\lambda)$ 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 w, and let $w=w_q$. We have $\{w_1,\ldots,w_q\}\subseteq C(u,\lambda)$, so $|C(u,\lambda)|\ge q$.

2. $C(u,\lambda)\subseteq B_i^j(u)$ but $C(v,\lambda)\subseteq B_i^{j'}(v)$ for some $j'\ne j$: This case occurs when $(\delta_i(u,A_i^j)\le\lambda$ and $\delta_i(v,A_i^j)>\lambda)$ or $(\delta_i(u,A_i^{j+1}>\lambda)$ and $\delta_i(v,A_i^{j+1})\le\lambda)$. As before, the triangle inequality implies $\delta_i(v,A_i^j)-\delta_i(u,A_i^j)$ and $\delta_i(u,A_i^{j+1})-\delta_i(v,A_i^{j+1})$ 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 $2\delta_i(u,v)\lg n < \delta_i(u,v)H_n \lg n$. During those moments, $C(u,\lambda)$ and $C(v,\lambda)$ are completely disjoint, implying the integrand above is equal to 2. Summing across $w \in V_i$, and noting each index q can be used at most once per choice of j, we have

$$\begin{aligned}\sum_{w\in V_i} |P_2(w,u) - P_2(w,v)| &\leq \frac{2}{\Delta_i}\int_0^{\Delta_i} \sum_{w\in V_i} \left(\begin{array}{c} \frac{[w \in (C(u,\lambda) \setminus C(v,\lambda))]}{|C(u,\lambda)|} \\ + \frac{[w \in (C(v,\lambda) \setminus C(u,\lambda))]}{|C(v,\lambda)|} \end{array} \right) d\lambda \\ &< \frac{2}{\Delta_i}\left(2\delta_i(u,v)H_n \lg n + 2\lg n \sum_{q=1}^{n} \frac{\delta_i(u,v)}{q} \right) \\ &= \frac{8\delta_i(u,v)H_n \lg n}{\Delta_i}.\end{aligned}$$

□

We set $\alpha := 200H_n \lg^2 n = O(\log^3 n)$.

Lemma 5.5. *We have* $||Pb||_1 \leq \alpha \cdot \mathrm{OPT}(b)$.

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 $c(f_j) \geq \delta(u,v)$ (in fact, equal). We consider charges based on a_j units in $b(u)$ and $-a_j$ units in $b(v)$.

Consider any V_i where there exists $u' \in V_i$ with $u \in V(u')$ and $\Delta_i \leq 8\delta(u,v)H_n \lg n$. By construction of P_2, $\sum_{w\in V_i} \sum_{w'\in(\cup_{i'>i}V_{i'})} \sum_{u''\in p^{-1}(u')} P_2((w',w),u'') = 1$. Considering all such i and the construction of P as a whole, we see

$$\begin{aligned}\sum_{i|\Delta_i \leq 8\delta(u,v)H_n \lg n} \sum_{w\in V_i} \sum_{w'\in(\cup_{i'>i}V_{i'})} \sum_{u''\in p^{-1}(u')} P((w',w),u') &\leq \sum_{i'|\Delta_{i'} \leq 4\delta(u,v)H_n \lg n} 6\Delta_{i'} \\ &\leq 48\delta(u,v)H_n \lg n \\ &\leq 24\delta(u,v)H_n \lg^2 n.\end{aligned}$$

The above implies that the a_j units of demand from u contribute at most $24a_j\delta(u,v)H_n \lg^2 n$ total among various $|(Pb)((w',w))|$ where $w \in V_i$ with $\Delta_i \leq 8\delta(u,v)H_n \lg n$. We charge $24a_j\delta(u,v)H_n \lg^2 n$ to f_j for these contributions.

Now, consider any i where there exist $u' \in V_i$, $v' \in V_i$, $u \in V(u')$, $v \in V(v')$, and $\Delta_i > 8\delta(u,v)H_n \lg n$.

By Lemma 5.4 and the fact that $|ab - cd| \leq |a - c| + |b - d|$ for $a, b, c, d \in [0, 1]$,

$$\sum_{w \in V_i} \sum_{w' \in (\cup_{i'} V_{i'})} \sum_{u'' \in p^{-1}(u)} \sum_{v'' \in p^{-1}(v)} |P_2((w', w), u'') - P_2((w', w), v'')| =$$

$$\sum_{w \in V_i} \sum_{w' \in (\cup_{i'} V_{i'})} \sum_{u'' \in p^{-1}(u)} \sum_{v'' \in p^{-1}(v)} |P_2((w', w'), u'')P_2((w, w),u') - P_2((w', w'), v'')P_2((w, w), v')|$$

$$\sum_{w \in V_i} \sum_{w' \in (\cup_{i'} V_{i'})} \sum_{u'' \in p^{-1}(u)} \sum_{v'' \in p^{-1}(v)} (|P_2((w', w'), u'') - P_2((w', w'),v'')| + |P_2((w, w), u') - P_2((w, w), v')|)$$

$$\leq \frac{8\delta(u, v)H_n \lg n}{\Delta_i} + \frac{8\delta(u, v)H_n \lg n}{\Delta_i/2}$$

$$\leq \frac{24\delta(u, v)H_n \lg n}{\Delta_i}.$$

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 at most $3\Delta_i$, so we charge $72a_j\delta(u, v)H_n \lg n$ for these contributions. There are at most $\lg n$ different values i of this type, bringing this third set of charges to $72a_j\delta(u, v)H_n \lg^2 n$.

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

$$\sum_{w \in V_0} |P_2((w, w), u') - P_2((w, w), v')| \leq \frac{8\delta(u, v)H_n \lg n}{\Delta_0}.$$

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 Δ_0, we charge $8a_j\delta(u, v)H_n \lg n \leq 4a_j\delta(u, v)H_n \lg^2 n$.

Summing over all three types of charges and then doubling the value for the contributions of v to various $|(Pb)(w', w)|$, we get a total charge of $200a_j\delta(u, v)H_n \lg^2 n \leq \alpha \cdot a_j c(f_j)$ 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 $\sum_j \alpha \cdot a_j c(f_j) = \alpha \cdot c(f^*) = \alpha \cdot \text{OPT}(b)$ in total. □

6 Approximation scheme

We are now ready to present our main theorem.

Theorem 6.1. *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 $c(f) \leq (1 + \varepsilon)\text{OPT}(d)$ 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 $O(m \log^{O(1)} n)$ time total. We construct sparse representations of the matrices P_2 and P_3 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 [Zuz23a, Corollaries 12 and 16] to find an infeasible flow f' of cost $c(f') \leq (1 + \varepsilon/2)\text{OPT}(b)$ where $\text{OPT}(b - I_G f') \leq \text{OPT}(b)/n^2$. Finally, we add in an n-approximate solution that routes $b - I_G f'$ as in [She17] in $O(m \log n)$ additional time. The total cost of the flow returned is $(1 + \varepsilon/2 + 1/n)\text{OPT}(b) \leq (1 + \varepsilon)\text{OPT}(b)$. □

References

[ASZ20] 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.

[BFKL21] 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.

[Bou85] Jean Bourgain. On Lipschitz embedding of finite metric spaces in Hilbert space. *Israel J. Math.*, 52(1–2), 1985.

[CKL+22] 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.

[Dij59] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[DS08] 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.

[FL22] 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.

[FL23a] Emily Fox and Jiashuai Lu. A deterministic near-linear time approximation scheme for geometric transportation. In *Proc. 64th IEEE Ann. Symp. Found. Comput. Sci.*, 2023. To appear.

[FL23b] Emily Fox and Jiashuai Lu. A deterministic near-linear time approximation scheme for geometric transportation. *CoRR*, arXiv:2211.03891v2, 2023.

[FRT04] 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.

[KNP21] Andrey Boris Khesin, Aleksandar Nikolov, and Dmitry Paramonov. Preconditioning for the geometric transportation problem. *J. Comput. Geom.*, 11(2):234–259, 2021.

[Li20] Jason Li. Faster parallel algorithm for approximate shortest path. In *Proc. 52nd Ann. ACM SIGACT Symp. Theory of Comput.*, pages 308–321, 2020.

[Orl93] James B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations Research*, 41(2):338–350, 1993.

[RGH+22] 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.

[RTZ05] 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.

[She17] Jonah Sherman. Generalized preconditioning and undirected minimum-cost flow. In *Proc. 28th Ann. ACM-SIAM Symp. Discrete Algorithms*, pages 772–780, 2017.

[TZ05] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, jan 2005.

[Vil08] Cédric Villani. *Optimal Transport: Old and New*. Springer Science & Business Media, 2008.

[ZGY+22] 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.

[Zuz23a] Goran Zuzic. A simple boosting framework for transshipment. In *Proc. 31st Ann. Europ. Symp. Algorithms*, 2023. To appear.

[Zuz23b] Goran Zuzic. A simple boosting framework for transshipment. *CoRR*, arXiv:2110.11723v2, 2023.