

Near-Optimal Approximate Shortest Paths and Transshipment in Distributed and Streaming Models

Ruben Becker Andreas Karrenbauer Sebastian Krininger
Christoph Lenzen

*Max Planck Institute for Informatics
Saarland Informatics Campus
Saarbrücken, Germany*

Abstract

We present a method for solving the *shortest transshipment* problem—also known as uncapacitated minimum cost flow—up to a multiplicative error of $1 + \varepsilon$ in undirected graphs with polynomially bounded integer edge weights using a tailored gradient descent algorithm. An important special case of the transshipment problem is the single-source shortest paths (SSSP) problem. Our gradient descent algorithm takes ε^{-3} polylog n iterations, and in each iteration it needs to solve the transshipment problem up to a multiplicative error of polylog n , where n is the number of nodes. In particular, this allows us to perform a single iteration by computing a solution on a sparse spanner of logarithmic stretch. As a consequence, we improve prior work by obtaining the following results:

1. *Broadcast congest model*: $(1 + \varepsilon)$ -approximate SSSP using $\tilde{O}(\varepsilon^{-O(1)}(\sqrt{n} + D))$ rounds,¹ where D is the (hop) diameter of the network.
2. *Broadcast congested clique model*: $(1 + \varepsilon)$ -approximate transshipment and SSSP using $\tilde{O}(\varepsilon^{-O(1)})$ rounds.
3. *Multipass streaming model*: $(1 + \varepsilon)$ -approximate transshipment and SSSP using $\tilde{O}(n)$ space and $\tilde{O}(\varepsilon^{-O(1)})$ passes.

The previously fastest algorithms for these models leverage sparse hop sets. We bypass the hop set construction; computing a spanner is sufficient with our method. The above bounds assume non-negative integer edge weights that are polynomially bounded in n ; for general non-negative weights, running times scale with the logarithm of the maximum ratio between non-zero weights. In case of asymmetric costs, running times scale with the maximum ratio between the costs of both directions over all edges.

¹We use $\tilde{O}(\cdot)$ to hide polylogarithmic factors in n .

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 3 |
| 2 | High-level Description of Underlying Ideas | 6 |
| 2.1 | Gradient Descent for Asymmetric Transshipment | 6 |
| 2.2 | Implementation in Various Models of Computation | 10 |
| 3 | Solving the Asymmetric Transshipment Problem | 12 |
| 3.1 | Single-Source Shortest Paths | 17 |
| 3.2 | Finding a Tree Solution | 19 |
| 4 | Applications | 20 |
| 4.1 | Broadcast Congested Clique | 20 |
| 4.2 | Broadcast Congest Model | 24 |
| 4.3 | Multipass Streaming | 25 |
| 5 | Further Related Work | 28 |
| | References | 30 |
| A | Deterministic Spanner Computation in Congested Clique and Multipass Streaming Model | 34 |
| B | Primal-Dual Pair | 36 |
| C | Inequality (11) | 36 |

1 Introduction

Single-source shortest paths (SSSP) is a fundamental and well-studied problem in computer science. Thanks to sophisticated algorithms and data structures [FT87, Tho99, HKT⁺15], it has been known for a long time how to obtain (near-)optimal running time in the RAM model. This is not the case in non-centralized models of computation, which become more and more relevant in a big-data world. In fact, inherent bottlenecks [KR90] seem to block progress for *exact* SSSP algorithms in these models. Thus, the focus has shifted towards efficient approximation schemes. For instance, in the Congest model of distributed computing, the state of the art is an algorithm that computes $(1 + \varepsilon)$ -approximate SSSP in $(\sqrt{n} + D) \cdot 2^{O(\sqrt{\log n / \log(\varepsilon^{-1} \sqrt{\log n})})}$ rounds [HKN16]. Even for constant ε , this exceeds a known lower bound of $\Omega(\sqrt{n} / \log n + D)$ rounds [DHK⁺12] by a super-polylogarithmic factor. As a consequence of the techniques developed in this paper, we make a qualitative algorithmic improvement in this model: we solve the problem in $(\sqrt{n} + D)\varepsilon^{-O(1)}$ polylog n rounds. We thus narrow the gap between upper and lower bound significantly and additionally improve the dependence on ε .

The algorithm of [HKN16] follows the framework developed in [Nan14]: First, it reduces the problem to computing SSSP on an overlay network in which communication is carried out by broadcasting via a global BFS tree. Second, it adds a sparse hop set to the overlay network to speed up an algorithm similar to Bellman-Ford on the overlay network. An (h, ε) -hop set is a set of weighted edges that, when added to the original graph, provides sufficient shortcuts to approximate all pairwise distances using paths with only h edges (“hops”). The running time of [HKN16] mentioned above is achieved by constructing an (h, ε) -hop set of size $n\rho$ where $h \leq 2^{O(\sqrt{\log n / \log(\varepsilon^{-1} \sqrt{\log n})})}$ and $\rho \leq 2^{O(\sqrt{\log n / \log(\varepsilon^{-1} \sqrt{\log n})})}$. Roughly speaking, *both* h and ρ enter the running time of the corresponding SSSP algorithm. Motivated by this application, it is therefore an interesting open problem to find better hop sets where both h and ρ are bounded by $\varepsilon^{-O(1)}$ polylog n . Although the concept of hop sets has been known for over 20 years now, and despite considerable efforts [Coh00, Ber09, HKN14, MPV⁺15, HKN16, EN16], to date no construction is known that is polylogarithmic in both h and ρ . In contrast to these efforts for designing faster approximate SSSP algorithms, our new approach avoids the use of hop sets completely. Instead, it achieves its superior running time by leveraging techniques from continuous optimization.

To this end, in fact we solve a more general problem than SSSP. In the *shortest transshipment* problem, we seek to find a cheapest routing for sending units of a single good from sources to sinks along the edges of a graph meeting the nodes’ demands. Equivalently, we want to find the minimum-cost flow in a graph where edges have unlimited capacity. The special case of SSSP can be modeled as a shortest transshipment problem by setting the demand of the source to $-n + 1$ and the demand of every other node to 1.

Techniques from continuous optimization have been key to recent breakthroughs in the combinatorial realm of graph algorithms [DS08, CKM⁺11, She13, KLO⁺14, Mađ13, LS14, CMS⁺17]. In this paper, we apply this paradigm to computing primal and dual $(1 + \varepsilon)$ -approximate solutions of the shortest transshipment in undirected graphs with non-negative edge weights. We make use of the available pool of techniques, but need to add significant

problem-specific tweaks. Concretely, our chief approach is to perform projected gradient descent for a suitable norm-minimization formulation of the problem, where we approximate the infinity norm by a differentiable soft-max function. This method reduces the problem of computing a $(1 + \varepsilon)$ -approximation to the more relaxed problem of computing, e.g., an $O(\log n)$ -approximation. We then exploit that an $O(\log n)$ -approximation can be computed very efficiently by solving a variant of the problem on a sparse spanner, and that it is well-known how to compute sparse spanners efficiently.

Our method is widely applicable among a plurality of non-centralized models of computation and we obtain the first non-trivial algorithms for approximate undirected shortest transshipment in the broadcast congest,² broadcast congested clique, and multipass streaming models. As a further, arguably more important, consequence, we improve upon prior results for computing approximate SSSP in these models. Our approximate SSSP algorithms are the first to be provably optimal up to polylogarithmic factors.

We note that an approximate (dual) solution to the transshipment problem merely yields distances to the source that are a $(1 + \varepsilon)$ -approximation *on average*. In the special case of SSSP, one typically is interested in obtaining a $(1 + \varepsilon)$ -approximate distance to the source for *each node*. We provide an extension of our algorithm that achieves this per-node guarantee. Interestingly, the generalization to the shortest transshipment problem seems conceptually necessary for our method to be applicable, even if we are only interested in solving the special case of SSSP.

Our Results. By implementing our method in specific models of computation, we obtain the following approximation schemes in graphs with non-negative polynomially bounded³ integer edge weights:

1. *Broadcast congest model:* We obtain a deterministic algorithm that computes a $(1 + \varepsilon)$ -approximate shortest transshipment using $\tilde{O}(\varepsilon^{-3}n)$ rounds. No non-trivial upper bound was known before in this model. We also obtain a deterministic algorithm for computing $(1 + \varepsilon)$ -approximate SSSP using $\tilde{O}(\varepsilon^{-O(1)}(\sqrt{n} + D))$ rounds. This improves upon the previous best upper bound of $(\sqrt{n} + D) \cdot 2^{O(\sqrt{\log n / \log(\varepsilon^{-1}\sqrt{\log n})})}$ rounds [HKN16]. For $\varepsilon^{-1} \in O(\text{polylog } n)$, we match the lower bound of $\Omega(\sqrt{n} / \log n + D)$ [DHK⁺12] (for any $(\text{poly } n)$ -approximation of the distance between two fixed nodes in a weighted undirected graph) up to polylogarithmic factors in n .
2. *Broadcast congested clique model:* We obtain a deterministic algorithm computing a $(1 + \varepsilon)$ -approximate shortest transshipment using $\tilde{O}(\varepsilon^{-3})$ rounds. No non-trivial upper bound was known before in this model. We also obtain a deterministic algorithm for computing $(1 + \varepsilon)$ -approximate SSSP using $\tilde{O}(\varepsilon^{-O(1)})$ rounds. This improves upon the previous best upper bound of $2^{O(\sqrt{\log n / \log(\varepsilon^{-1}\sqrt{\log n})})}$ rounds [HKN16].

²Also known as the node-congest model.

³For general non-negative weights, running times scale by a multiplicative factor of $\log R$, where R is the maximum ratio between non-zero edge weights.

3. *Multipass streaming model:* We obtain a deterministic algorithm for computing a $(1 + \varepsilon)$ -approximate shortest transshipment using $\tilde{O}(\varepsilon^{-3})$ passes and $O(n \log n)$ space. No non-trivial upper bound was known before in this model. We also obtain a deterministic algorithm for computing $(1 + \varepsilon)$ -approximate SSSP using $O(\varepsilon^{-O(1)})$ passes and $O(n \log n)$ space. This improves upon the previous best upper bound of $(2 + 1/\varepsilon)^{O(\sqrt{\log n \log \log n})}$ passes and $O(n \log^2 n)$ space [EN16]. By setting ε small enough we can compute distances up to the value $\log n$ exactly in integer-weighted graphs using $\text{polylog } n$ passes and $O(n \log n)$ space. Thus, up to polylogarithmic factors in n , our result matches a lower bound of $n^{1+\Omega(1/p)}/\text{poly } p$ space for all algorithms that decide in p passes if the distance between two fixed nodes in an unweighted undirected graph is at most $2(p + 1)$ for any $p = O(\log n / \log \log n)$ [GO13].

In the case of shortest transshipment, we can (deterministically) return $(1 + \varepsilon)$ -approximate primal and dual solutions. We can further extend the results to asymmetric weights on undirected edges, where each edge can be used in either direction at potentially different costs. Denoting by $\lambda \geq 1$ the maximum over all edges of the cost ratio between traversing the edge in different directions, our algorithms give the same guarantees if the number of rounds or passes, respectively, is increased by a factor of $\lambda^4 \log \lambda$.

In the case of SSSP, we can deterministically compute a $(1 + \varepsilon)$ -approximation of the distance to the source for every node. Using a *randomized* procedure, we can additionally compute (with high probability within the same asymptotic running times) a tree on which every node has a path to the source that is within a factor of $(1 + \varepsilon)$ of its true distance.

Generally speaking, the meta-theorem behind our results is the following: Whenever we have an algorithm for computing an α -approximate shortest transshipment on an undirected graph, we can extend it to an algorithm for computing a $(1 + \varepsilon)$ -approximation at the cost of an overhead of $\tilde{O}(\varepsilon^{-3}\alpha^2)$.⁴ We can also apply this scheme to the RAM model, for example. As one can simply compute an $O(\log n)$ -approximate shortest transshipment on a near-linear size spanner of stretch $O(\log n)$, and the spanner can in turn be computed in nearly-linear time [RTZ05], we obtain a deterministic $(1 + \varepsilon)$ -approximate shortest transshipment algorithm spending $\tilde{O}(\varepsilon^{-3}n^2)$ time. For dense graphs and a wide range of ε , this is faster than the recent $(1 + \varepsilon)$ -approximate shortest transshipment algorithm of Sherman [She17] with running time $O(\varepsilon^{-2}m^{1+o(1)})$. Furthermore, if Sherman’s algorithm can be adapted to return a dual solution, using it to solve the problem on the spanner would yield a running time of $\tilde{O}(\varepsilon^{-3}(m + n^{1+o(1)}))$, which is faster in any graph that is not very sparse. We do not get any explicit results in the PRAM model, but the task of qualitatively improving Cohen’s long-standing upper bound of $m^{1+o(1)}$ work and $\text{polylog } n$ depth for $(1 + \varepsilon)$ -approximate SSSP [Coh00] now reduces to finding a $(\text{polylog } n)$ -approximate transshipment using $\tilde{O}(m)$ work and $\text{polylog } n$ depth.

Related Work on Shortest Transshipment. Shortest transshipment is a classic problem in combinatorial optimization [KV00, Sch03]. The classic algorithms for directed graphs with

⁴The ‘overhead’ we mean here is model-specific: number of rounds in the Congest model and congested clique, respectively, number of passes in the multipass streaming model, and work and depth, respectively, in the PRAM model.

non-negative edge weights in the RAM model run in time $O(n(m + n \log n) \log n)$ [Or193] and $O((m + n \log n)B)$ [EK72], respectively, where B is the sum of the nodes' demands (when they are given as integers) and the term $m + n \log n$ comes from SSSP computations. If the graph contains negative edge weights, then these algorithms require an additional preprocessing step to compute SSSP in presence of negative edge weights, for example in time $O(mn)$ using the Bellman-Ford algorithm [Bel58, For56] or in time $O(m\sqrt{n} \log N)$ using Goldberg's algorithm [Gol95].⁵ The weakly polynomial running time was improved to $\tilde{O}(m\sqrt{n} \text{polylog } R)$ in a recent breakthrough for minimum-cost flow [LS14], where R is the ratio between the largest and the smallest edge weight. Independent of our research, very recently Sherman [She17] obtained a randomized algorithm for computing a $(1 + \varepsilon)$ -approximate shortest transshipment in undirected graphs with non-negative edge weights in time $O(\varepsilon^{-2}m^{1+o(1)})$ in the RAM model. This algorithm is based on a preconditioning approach for speeding up gradient descent. We are not aware of any non-trivial algorithms for computing (approximate) shortest transshipment in non-centralized models of computation, such as distributed and streaming models. It is unclear whether Sherman's approximation algorithm [She17] can be implemented efficiently in these models. In any event, the $\varepsilon^{-2}n^{o(1)}$ bound on the number of iterations seems inherent in Sherman's approach, whereas our method brings this number down to $\varepsilon^{-3} \text{polylog } n$.

We review further related work on (approximate) SSSP in Section 5.

2 High-level Description of Underlying Ideas

In the following two subsections, we will describe the main ideas underlying our approach from a high-level perspective. The first part covers the gradient descent method for asymmetric transshipment and in the second part we describe how it can be implemented in distributed and streaming models of computation. For a complete description including all proofs see Sections 3 and 4.

2.1 Gradient Descent for Asymmetric Transshipment

Let $G = (V, E)$ be a (w.l.o.g. connected) graph with n nodes, m edges, positive integral forward and backward edge weights $w^+, w^- \in \mathbb{Z}_{\geq 1}^m$, and a demand vector $b \in \mathbb{Z}^n$ on the nodes. We split every edge $e \in E$ into a *forward arc* (u, v) and a *backward arc* (v, u) such that $w^+(u, v) \geq w^-(v, u)$, and denote by E^+ and E^- the sets of forward and backward arcs, respectively. Let us denote $W_+ = \text{diag}(w^+) \in \mathbb{Z}_{\geq 1}^{m \times m}$ and $W_- = \text{diag}(w^-) \in \mathbb{Z}_{\geq 1}^{m \times m}$. Moreover, we will use $W_* := [W_+ \ -W_-]^T$ for the composition of these weight matrices and $R_* := [W_+^{-1} \ -W_-^{-1}]^T$ for the composition of their inverses. Moreover, let us, for a vector $v \in \mathbb{R}^d$, define the asymmetric norms $p(v) := \sum_{i \in [d]} \max\{0, v_i\}$ and $q(v) := \max\{0, \max\{v_i : i \in [d]\}\}$.

The *asymmetric shortest transshipment problem* can then be written as a primal/dual pair of linear programs, see Appendix B,

$$\min\{p(W_*x) : Ax = b\} = \max\{b^T y : q(R_*A^T y) \leq 1\}, \quad (1)$$

⁵Goldberg's running time bound holds for integer-weighted graphs with most negative weight $-N$.

where $A \in \mathbb{R}^{n \times m}$ is the node-arc incidence matrix of G with respect to the orientation of the edges such that $w^+ \geq w^-$. Let us denote $\lambda := \|W_-^{-1}w_+\|_\infty \geq 1$. Note that, for instance, for the s - t shortest path problem b is equal to $\mathbf{1}_t - \mathbf{1}_s$, and for the single-source shortest path problem b is equal to $\mathbf{1} - n\mathbf{1}_s$. We remark that the *symmetric undirected shortest transshipment problem*, which is in fact a 1-norm-minimization problem, can, for given weights $w \in \mathbb{Z}_{\geq 1}^m$, be written as

$$\min\{\|Wx\|_1 : Ax = b\} = \max\{b^T y : \|W^{-1}A^T y\|_\infty \leq 1\}, \quad (2)$$

where $W = \text{diag}(w) \in \mathbb{R}^{m \times m}$. This is in fact a special case of the asymmetric shortest transshipment problem, where $w^+ = w^-$ and thus $\lambda = 1$.

In this section, we will describe a gradient descent method that, given an oracle that computes α -approximate solutions to the *symmetric undirected* shortest transshipment problem, returns primal and dual feasible solutions x and y to the *asymmetric* shortest transshipment problem that are $1 + \varepsilon$ close to optimal, i.e., fulfill $p(W_*x) \leq (1 + \varepsilon)b^T y$, with $O(\varepsilon^{-3}\lambda^4\alpha^2 \log \alpha \log n)$ calls to the oracle.

As our first step, we relate the dual of the asymmetric shortest transshipment problem to another linear program that normalizes the objective to 1 and seeks to minimize $q(R_*A^T y)$:

$$\min\{q(R_*A^T \pi) : b^T \pi = 1\}. \quad (3)$$

We will denote by π^* an optimal solution to this problem, whereas y^* will denote an optimal solution to the dual of the original problem (1). Notice that feasible solutions π of (3) that satisfy $q(R_*A^T \pi) > 0$ are mapped to feasible solutions of the dual program in (1) via $f(\pi) := \pi/q(R_*A^T \pi)$. Similarly, feasible solutions y of the dual program in (1) that satisfy $b^T y > 0$ are mapped to feasible solutions of (3) via $g(y) := y/b^T y$. Moreover, the map $f(\cdot)$ preserves the approximation ratio. Namely, for any $\varepsilon > 0$, if π is a solution of (3) within factor $1 + \varepsilon$ of the optimum, then $f(\pi)$ is feasible for (1) and within factor $1 + \varepsilon$ of the optimum. In particular, $f(\pi^*)$ is an optimal solution of (1).

We remark that there is also a clear relation between the asymmetric and the symmetric version of the problem. Namely, an $\alpha\lambda$ -approximate solution to the asymmetric shortest transshipment problem can be obtained from an α -approximate solution to a specific symmetric transshipment problem, namely $\min\{\|W_-^{-1}A^T \bar{\pi}\|_\infty : b^T \bar{\pi} = 1\}$, which we call the *symmetrized problem* for the given asymmetric instance.

Corollary 2.1. *Let us denote with $\bar{\pi}^*$ the optimal solution of $\min\{\|W_-^{-1}A^T \bar{\pi}\|_\infty : b^T \bar{\pi} = 1\}$ and let $\bar{\pi}$ be a feasible solution to that problem such that $\|W_-^{-1}A^T \bar{\pi}\|_\infty \leq \alpha \cdot \|W_-^{-1}A^T \bar{\pi}^*\|_\infty$. Then, $\bar{\pi}$ is feasible for (3) and moreover $q(R_*A^T \bar{\pi}) \leq \alpha\lambda \cdot q(R_*A^T \bar{\pi}^*)$.*

One of our contributions lies in showing that given a primitive that gives a very bad approximation, say logarithmic, to the *symmetric* shortest transshipment problem, a $1 + \varepsilon$ -approximate solution even to the *asymmetric* shortest transshipment problem can be obtained with a small number of iterations of a gradient descent method. Applying gradient descent to (3) directly is however not feasible, since the objective is not differentiable, thus we change the problem one more time by using the so-called soft-max (a.k.a. log-sum-exp or lse for

short), which is a suitable approximation for the infinity norm $\|\cdot\|_\infty$. It is defined for vectors $v \in \mathbb{R}^d$ as $\text{lse}_\beta(v) := \frac{1}{\beta} \ln \left(\sum_{i \in [d]} e^{\beta v_i} \right)$, where $\beta > 0$ is a parameter that controls the accuracy of the approximation of the infinity-norm at the expense of smoothness. This enables us to define a sufficiently smooth and accurate *potential function* $\Phi_\beta(\pi) := \text{lse}_\beta(R_* A^T \pi)$, for some parameter β that will be adapted during the course of the gradient descent method. The algorithm takes as argument a starting-solution π that is an α -approximate solution to the symmetrized problem, i.e., the symmetric shortest transshipment problem with weights w^- , see Corollary 2.1, an initial β that is appropriate for π , and the desired approximation guarantee ε . The algorithm in every iteration updates the iterate using an α -approximate solution to a symmetric shortest transshipment problem with a demand vector depending on the gradient and weights w^- . The algorithm, see Algorithm `gradient_ust` for a pseudo-code implementation, returns potentials $\pi \in \mathbb{R}^n$ and the constant β (this will be of interest only later when we consider the special case of the single source shortest path problem).

Algorithm 1: `gradient_ust` ($G, b, \pi, \beta, \varepsilon$)

```

//  $\pi$  is  $\alpha$ -approximation to symmetrized problem,  $\beta$  such that  $\Phi_\beta(\pi) \in \left[ \frac{4 \ln(2m)}{\varepsilon \beta}, \frac{5 \ln(2m)}{\varepsilon \beta} \right]$ 
1 repeat // Invariant:  $b^T \pi = 1$ 
2   while  $\frac{4 \ln(4m)}{\varepsilon \beta} \geq \Phi_\beta(\pi)$  do  $\beta \leftarrow \frac{5}{4} \beta$ .
3    $P := [I - \pi b^T]$ ,  $\tilde{b} := P^T \nabla \Phi_\beta(\pi)$ 
4   Determine  $\tilde{h}$  with  $\|W_-^{-1} A^T \tilde{h}\|_\infty = 1$  and  $\tilde{b}^T \tilde{h} \geq \frac{1}{\alpha} \max\{\tilde{b}^T h : \|W_-^{-1} A^T h\|_\infty \leq 1\}$ 
   //  $\tilde{h}$  can be obtained as an  $\alpha$ -approximate solution of the transshipment problem
   // with weights  $w^-$  and demand vector  $P^T \nabla \Phi_\beta(\pi)$ .
5   Let  $\delta := \frac{\tilde{b}^T \tilde{h}}{\|R_* A^T P \tilde{h}\|_\infty}$ 
6   if  $\delta > \frac{\varepsilon}{8\alpha\lambda^2}$  then  $\pi \leftarrow \pi - \frac{\delta}{2\beta\|R_* A^T P \tilde{h}\|_\infty} P \tilde{h}$ .
7 until  $\delta \leq \frac{\varepsilon}{8\alpha\lambda^2}$ 
8 return  $\pi, \beta$ 

```

We will now show that a primal-dual pair that is $(1 + \varepsilon)$ -close to optimal in (1) can be constructed from the output potentials π and α -approximate primal and dual solutions to the symmetrized problem that was solved in the last iteration of the algorithm. Note that if one is only interested in a dual solution to (1), then also the α -approximate dual solution is enough and thus also an oracle that provides a dual solution is sufficient.

Lemma 2.2 (Correctness). *Let $\pi \in \mathbb{R}^n$ with $b^T \pi = 1$ be output by Algorithm `gradient_ust`. Let $x_1 \in \mathbb{R}^m$ such that $Ax_1 = \nabla \Phi_\beta(\pi)$ and $p(W_* x_1) \leq 1 + \varepsilon/8$ and let $x_2 \in \mathbb{R}^m$ such that $Ax_2 = \tilde{b}$ and $\|W_- x_2\|_1 \leq \alpha \tilde{b}^T \tilde{h}$ for some $\tilde{h} \in \mathbb{R}^n$ with $\|W_-^{-1} A^T \tilde{h}\|_\infty \leq 1$. Then $x := \frac{x_1 - x_2}{\pi^T \nabla \Phi_\beta(\pi)}$, $y := \frac{\pi}{q(R_* A^T \pi)}$ satisfy $Ax = b$, $q(R_* A^T y) \leq 1$ and $p(W_* x) \leq (1 + \varepsilon)b^T y$.*

Proof. First note that $Ax = \frac{\nabla \Phi_\beta(\pi) - \tilde{b}}{\pi^T \nabla \Phi_\beta(\pi)} = b$ and $q(R_* A^T y) = 1$, which establishes feasibility. Now note that convexity of $\Phi_\beta(\cdot)$ and the choice of β yield $\pi^T \nabla \Phi_\beta(\pi) \geq (1 - \varepsilon/4)\Phi_\beta(\pi) \geq$

$(1 - \varepsilon/4)q(R_*A^T\pi) \geq 0$, see (12) in Section 3 for a detailed derivation of this estimate. Thus

$$p(W_*x) \leq \frac{1 + \varepsilon/8 + \lambda\|W_-x_2\|_1}{\pi^T\nabla\Phi_\beta(\pi)} \leq \frac{1 + \varepsilon/8 + \alpha\lambda\tilde{b}^T\tilde{h}}{\pi^T\nabla\Phi_\beta(\pi)}.$$

With the definition $\delta := \frac{\tilde{b}^T\tilde{h}}{\|R_*A^TP\tilde{h}\|_\infty}$ from Algorithm 1, this yields

$$p(W_*x) \leq \frac{1 + \varepsilon/8 + \alpha\lambda\delta\|R_*A^TP\tilde{h}\|_\infty}{\pi^T\nabla\Phi_\beta(\pi)} \leq \frac{1 + \varepsilon/8 + \alpha\lambda^2\delta(1 + q(R_*A^T\pi)b^Ty^*)}{(1 - \frac{\varepsilon}{4})q(R_*A^T\pi)},$$

where the last inequality uses triangle inequality, $\|R_*A^T\pi\|_\infty \leq \lambda q(R_*A^T\pi)$ and $\pi^T\nabla\Phi_\beta(\pi) \geq (1 - \frac{\varepsilon}{4})q(R_*A^T\pi)$. Moreover, since $b^Ty^* \leq p(W_*x)$ and $b^Ty = 1/q(R_*A^T\pi)$, we conclude that $p(W_*x) \leq (1 + \frac{\varepsilon}{4})/(1 - \frac{\varepsilon}{2})b^Ty \leq (1 + \varepsilon)b^Ty$ whenever $\delta \leq \frac{\varepsilon}{8\alpha\lambda^2}$. \square

Observe that a suitable choice for x_1 appears when applying the chain rule to determine $\nabla\Phi_\beta(\pi)$, i.e., we may set $x_1 := R_*^T\nabla\text{lse}_\beta(R_*A^T\pi)$, which satisfies $Ax_1 = \nabla\Phi_\beta(\pi)$ and $p(W_*x_1) \leq 1$ by (11). We can also obtain a tree solution x_T that routes $\nabla\Phi_\beta(\pi)$ by sampling from x_1 ; we get x_T such that $p(W_*x_T) \leq 1 + \varepsilon/8$ with high probability in $O(\frac{\log n}{\varepsilon})$ trials. Note that, unless we want a primal solution, it is not necessary to have x_1 explicitly; knowing Ax_1 is sufficient. Lemma 2.2 then guarantees that the spanner augmented with the tree must contain a $(1 + \varepsilon)$ -approximate tree solution for the original problem. Moreover, we can compute such a tree solution locally.

It remains to show a bound on the number of iterations that the algorithm needs. We will obtain this bound by showing that the potential function decreases by a multiplicative factor in each iteration.

Corollary 2.3 (Multiplicative Decrement of Φ_β). *Given $\pi \in \mathbb{R}^n$ and $\tilde{h} \in \mathbb{R}^n$ such that $\|W_*^{-1}A^T\tilde{h}\|_\infty \leq 1$ and suppose that β is such that $\varepsilon\beta\Phi_\beta(\pi) \leq 5\ln(2m)$. Then for $\delta := \frac{\tilde{b}^T\tilde{h}}{\|R_*A^TP\tilde{h}\|_\infty}$, it holds that*

$$\Phi_\beta\left(\pi - \frac{\delta}{2\beta\|R_*A^TP\tilde{h}\|_\infty}P\tilde{h}\right) \leq \left[1 - \frac{\varepsilon\delta^2}{20\ln(2m)}\right]\Phi_\beta(\pi).$$

The proof, which uses properties of the potential function $\Phi_\beta(\cdot)$ such as convexity and Lipschitz smoothness, can be found in Section 3. The above corollary is sufficient in order to show the following bound on the number of iterations.

Lemma 2.4 (Number of Iterations). *Suppose that $0 < \varepsilon \leq 1/2$. Then Algorithm `gradient_ust` terminates within $O([\varepsilon^{-3} + \log \lambda + \log \alpha]\lambda^4\alpha^2 \log n)$ iterations.*

We conclude this paragraph with the following theorem summarizing our results so far.

Theorem 2.5. *Given an oracle that computes an α -approximate solutions to the undirected transshipment problem, using Algorithm `gradient_ust`, we can compute primal and dual solutions x, y to the asymmetric shortest transshipment problem satisfying $p(W_*x) \leq (1 + \varepsilon)b^Ty$ with $\tilde{O}(\varepsilon^{-3}\alpha^2\lambda^4)$ oracle calls.*

Single-Source Shortest Paths. In the special case of single-source shortest paths, we have $b_v = 1$ for all $v \in V \setminus \{s\}$ and $b_s = 1 - n$ for the source s . In fact, it is the combination of $n - 1$ shortest s - t -path problems. Thus, the approximation guarantee from the previous section only holds on average over all sink-nodes. However, we show how to use Algorithm `gradient_ust` to obtain the distance from the source within a factor of $1 + \varepsilon$ for *each* node. To this end, we use a little more precision so that we can recognize the nodes for which we know the distance with sufficient accuracy using the tools we proposed above. We then concentrate on the other nodes by adapting the demand vector b accordingly so that $b_v = 0$ for all good nodes v , i.e., the nodes for which we already know the distance within a factor of $1 + \varepsilon$. We iterate until all nodes are good. This yields the following theorem, which is proven in Section 3.

Theorem 2.6. *Let $y^* \in \mathbb{R}^n$ denote the distances of all nodes from the source node s . There is an algorithm that computes a vector $y \in \mathbb{R}^n$ with $q(R_* A^T y) \leq 1$ such that for each $v \in V$ it holds that $y_v^*/(1 + \varepsilon) \leq y_v \leq y_v^*$ with $\text{polylog}(n, \|w\|_\infty)$ calls to Algorithm `gradient_ust`.*

2.2 Implementation in Various Models of Computation

Common to all our implementations is the use of sparse *spanners*.

Definition 2.7 (Spanner). *Given a weighted graph $G = (V, E, w)$ and $\alpha > 1$, an α -spanner of G is a subgraph $(V, E', w|_{E'})$, $E' \subseteq E$, in which distances are at most by factor α larger than in G .*

In other words, a spanner removes edges from G while approximately preserving distances. This implies that an optimal solution of an instance of the shortest transshipment problem on an α -spanner of the input graph is an α -approximate solution to the original problem.

Broadcast Congested Clique. In the *broadcast congested clique* model, the system consists of n fully connected nodes labeled by unique $O(\log n)$ -bit identifiers. Computation proceeds in synchronous rounds, where in each round, nodes may perform arbitrary local computations, broadcast (send) an $O(\log n)$ -bit message to the other nodes, and receive the messages from other nodes. The input is distributed among the nodes. The first part of the input of every node consists of its incident edges (given by their endpoints' identifiers) and their weights. The second part of the input is problem specific: for the transshipment problem, every node v knows its demand b_v and for SSSP v knows whether or not it is the source s . In both cases, every node knows $0 < \varepsilon \leq 1$ as well. Each node needs to compute its part of the output. For shortest transshipment every node needs to know a $(1 + \varepsilon)$ approximation of the optimum value, and for SSSP every node needs to know a $(1 + \varepsilon)$ -approximation of its distance to the source. The complexity of the algorithm is measured in the worst-case number of rounds until the computation is complete.

Implementing our approach in this model is straightforward. The key observations are:

- Every node can locally aggregate information about its incident edges (e.g. concerning the “stretches” under the potential of the current solution π) and make it known to all

other nodes in a single communication round. Thus, given $\beta > 0$ and $\pi \in \mathbb{R}^n$, it is rather straightforward to evaluate $\Phi_\beta(\pi)$ and $\nabla\Phi_\beta(\pi)$ in a constant number of rounds.

- An $O(\log n)$ -spanner of the input graph can be computed and made known to all nodes quickly (see Appendix A).
- Local computation then suffices to solve (sub)problems on the spanner optimally. In particular, $O(\log n)$ -approximate solutions to (constrained) transshipment problems can be computed easily, where it suffices to communicate the demand vector.

This leads to the following main results; for details we refer to Section 4.1.

Theorem 2.8. *For any $0 < \varepsilon \leq 1$, in the broadcast congested clique model a deterministic $(1 + \varepsilon)$ -approximation to the shortest transshipment problem in undirected graphs with non-negative weights can be computed in ε^{-3} polylog n rounds.*

Theorem 2.9. *For any $0 < \varepsilon \leq 1$, in the broadcast congested clique model a deterministic $(1 + \varepsilon)$ -approximation to single-source shortest paths in undirected graphs with non-negative weights can be computed in ε^{-9} polylog n rounds.*

If in addition to the approximate dual solution to the transshipment problem, we also want to obtain an approximate primal solution, we can do the following. After the last iteration of the gradient descent algorithm (with gradient $\nabla\Phi_\beta(\pi)$), we first locally compute for every node the value of $x_1 := R_*^T \nabla \text{lse}_\beta(R_* A^T \pi)$ for its incident edges. We then compute a tree solution x_2 to the shortest transshipment on the spanner with demands $\tilde{b} := P^T \nabla\Phi_\beta(\pi)$ and broadcast it to all nodes. Now every node can compute, for its incident edges, the value of x from the values of x_1 and x_2 as specified in Lemma 2.2.

To obtain a primal tree solution we sample a tree from x_1 and make it known to all nodes. Since we can use the gradient to orient the edges according to x_1 , obtaining a DAG, the sampling can be performed locally at every node. We then compute a tree solution in the graph combining the trees from x_1 and the edges of the initial spanner locally at every node. To obtain an approximate tree solution in the case of single-source shortest paths, we repeat the sampling after every call of the gradient descent algorithm. We can then find the approximate shortest path tree in the graph combining all sampled edges and the initial spanner. Since the number of calls to the gradient descent algorithm is polylog n , the resulting graph is still small enough so that we can make it known to every node and thus perform this computation locally at every node.

Broadcast Congest Model. The *broadcast congest* model differs from the broadcast congested clique in that communication is restricted to edges that are present in the input graph. That is, node v receives the messages sent by node w if and only if $\{v, w\} \in E$. All other aspects of the model are identical to the broadcast congested clique. We stress that this restriction has significant impact, however: Denoting the hop diameter⁶ of the input graph by D , it is

⁶That is, the diameter of the unweighted graph $G = (V, E)$.

straightforward to show that $\Omega(D)$ rounds are necessary to solve the transshipment problem. Moreover, it has been established that $\Omega(\sqrt{n}/\log n)$ rounds are required even on graphs with $D \in O(\log n)$ [DHK⁺12]. Both of these bounds apply to randomized approximation algorithms (unless the approximation ratio is not polynomially bounded in n).

Our main result for this model is that we can match the above lower bounds for approximate single-source shortest paths computation. The solution is based on combining a known reduction to an overlay network on $\tilde{\Theta}(\sqrt{n})$ nodes, simulating the broadcast congested clique on this overlay, and applying Theorem 2.9. Simulating a round of the broadcast congested clique for k nodes is done by pipelining each of the k messages over a breadth-first search tree of the underlying graph, taking $O(D + k)$ rounds. See Section 4.2 for details.

Corollary 2.10. *For any $0 < \varepsilon \leq 1$, in the broadcast congest model a deterministic $(1 + \varepsilon)$ -approximation to single-source shortest paths in undirected graphs with non-negative weights can be computed in $\tilde{O}(\varepsilon^{-O(1)}(\sqrt{n} + D))$ rounds.*

Multipass Streaming In the *streaming* model the input graph is presented to the algorithm edge by edge as a “stream” without repetitions and the goal is to design algorithms that use as little space as possible. In the *multipass streaming* model, the algorithm is allowed to make several such passes over the input stream and the goal is to design algorithms that need only a small number of passes (and again little space). For graph algorithms, the usual assumption is that the edges of the input graph are presented to the algorithm in arbitrary order.

The main observation is that we can apply the same approach as before with $O(n \log n)$ space: this enables us to store a spanner throughout the entire computation, and we can keep track of intermediate (node) state vectors. Any computations on the spanner are thus “free,” while $\Phi_\beta(\pi)$ and $\nabla\Phi_\beta(\pi)$ can be evaluated in a single pass by straightforward aggregation. With this in mind, it is straightforward to show that $\varepsilon^{-O(1)}$ polylog n passes suffice for completing the computation.

Theorem 2.11. *For any $0 < \varepsilon \leq 1$, in the multipass streaming model a deterministic $(1 + \varepsilon)$ -approximation to the shortest transshipment problem in undirected graphs with non-negative weights can be computed in ε^{-3} polylog n passes with $O(n \log n)$ space.*

Theorem 2.12. *For any $0 < \varepsilon \leq 1$, in the multipass streaming model, deterministic $(1 + \varepsilon)$ -approximation to single-source shortest paths in undirected graphs with non-negative weights can be computed in ε^{-9} polylog n passes with $O(n \log n)$ space.*

3 Solving the Asymmetric Transshipment Problem

In this section we give a complete presentation of the gradient descent method described in Section 2.1. We consider $(1 + \varepsilon)$ -approximation algorithms for the *asymmetric shortest transshipment problem* without capacity constraints on a (w.l.o.g. connected) undirected graph $G = (V, E)$ with n nodes, m edges, and positive integer forward and backward edge weights $w^+, w^- \in \mathbb{Z}_{\geq 1}^m$. Note that excluding 0 as edge-weight is a mild restriction because we can

always generate new weights w' with $w'_e = 1 + \lceil n/\varepsilon \rceil \cdot w_e$ while preserving at least one of the shortest paths between each pair of nodes as well as $(1 + \varepsilon)$ -approximations. We split every edge $e \in E$ into a *forward arc* (u, v) and a *backward arc* (u, v) such that $w^+(u, v) \geq w^-(v, u)$, and denote by E^+ and E^- the sets of forward and backward arcs, respectively. Let us denote $W_+ = \text{diag}(w^+) \in \mathbb{R}^{m \times m}$ and $W_- = \text{diag}(w^-) \in \mathbb{R}^{m \times m}$ and

$$W_* := \begin{bmatrix} W_+ \\ -W_- \end{bmatrix} \quad \text{and} \quad R_* := \begin{bmatrix} W_+^{-1} \\ -W_-^{-1} \end{bmatrix}.$$

Let us, w.l.o.g. fix an orientation of the edges such that $w^+ \geq w^-$ and denote $\lambda := \|W_-^{-1}w_+\|_\infty \geq 1$ the maximal ratio between a forward and a backward arc.

Moreover, let us, for a vector $v \in \mathbb{R}^d$, define the following asymmetric norms

$$p(v) := \sum_{i \in [d]} \max\{0, v_i\} \quad \text{and} \quad q(v) := \max\{0, \max\{v_i : i \in [d]\}\}.$$

The asymmetric transshipment problem can then be written as a primal/dual pair of linear programs (see Appendix B):

$$\min\{p(W_*x) : Ax = b\} = \max\{b^T y : q(R_*A^T y) \leq 1\}, \quad (1)$$

where $A \in \mathbb{R}^{n \times m}$ is the node-arc incidence matrix of G with respect to the orientation of the edges picked as described above and $b \in \mathbb{Z}^n$ is the demand vector. For instance, b is equal to $\mathbf{1}_t - \mathbf{1}_s$ for the s - t shortest path problem and $\mathbf{1} - n\mathbf{1}_s$ for the single-source shortest paths problem. We remark that this is different from the symmetric version of the transshipment problem, which is in fact a 1-norm-minimization problem and can, for given weights $w \in \mathbb{Z}_{\geq 1}^m$ be written as

$$\min\{\|Wx\|_1 : Ax = b\} = \max\{b^T y : \|W^{-1}A^T y\|_\infty \leq 1\}, \quad (2)$$

where $W = \text{diag}(w) \in \mathbb{R}^{m \times m}$.

Note that by weak duality for the linear program pair in (1), we obtain that for any $x \in \mathbb{R}^m$ with $Ax = b$ and $y \in \mathbb{R}^n$ with $q(R_*A^T y) \neq 0$, it holds that

$$b^T \frac{y}{q(R_*A^T y)} \leq p(W_*x), \quad \text{and thus} \quad x^T A^T y \leq p(W_*x)q(R_*A^T y). \quad (4)$$

We remark that the above inequality holds for an arbitrary right hand side $b \in \mathbb{R}^n$. Moreover, $q(R_*A^T h)$ can be bounded in terms of $\|R_*A^T h\|_\infty = \|W_-A^T h\|_\infty$ as follows

$$\frac{\|W_-A^T h\|_\infty}{\lambda} \leq q(R_*A^T h) \leq \|W_-A^T h\|_\infty. \quad (5)$$

In the following, we consider G and b to be fixed, and denote by y^* an optimal solution of the dual program, i.e., $b^T y^* = \max\{b^T y : q(R_*A^T y) \leq 1\}$. W.l.o.g., we restrict to feasible and non-trivial instances, i.e., $b^T \mathbf{1} = 0$ and $b \neq 0$. As our first step, we relate the dual program

to another linear program that normalizes the objective to 1 and seeks to minimize $q(R_*A^T y)$ instead:

$$\min\{q(R_*A^T \pi) : b^T \pi = 1\}.$$

We will denote by π^* an optimal solution to this problem.

Observation 3.1. *1. Feasible solutions π of (3) that satisfy $q(R_*A^T \pi) > 0$ are mapped to feasible solutions of the dual in (1) via $f(\pi) := \frac{\pi}{q(R_*A^T \pi)}$. Feasible solutions y of the dual in (1) that satisfy $b^T y > 0$ are mapped to feasible solutions of (3) via $g(y) := \frac{y}{b^T y}$.*

2. The map $f(\cdot)$ preserves the approximation ratio. Namely, for any $\varepsilon > 0$, if π is a solution of (3) within factor $1 + \varepsilon$ of the optimum, then $f(\pi)$ is feasible for (1) and within factor $1 + \varepsilon$ of the optimum. In particular, $f(\pi^)$ is an optimal solution of (1).*

In other words, it is sufficient to determine a $(1 + \varepsilon)$ -approximation to (3) in order to obtain a $(1 + \varepsilon)$ -approximation to (1). Another observation is that an $\alpha\lambda$ -approximate solution to the asymmetric transshipment problem can be obtained from an α -approximate solution to a specific symmetric transshipment problem, which we call the symmetrized problem.

Corollary (Restatement of Corollary 2.1). *Let us denote with $\bar{\pi}^*$ the optimal solution of $\min\{\|W^{-1}A^T \pi\|_\infty : b^T \pi = 1\}$ and let $\bar{\pi}$ be a feasible solution to that problem such that $\|W^{-1}A^T \bar{\pi}\|_\infty \leq \alpha \cdot \|W^{-1}A^T \bar{\pi}^*\|_\infty$. Then, $\bar{\pi}$ is feasible for (3) and moreover $q(R_*A^T \bar{\pi}) \leq \alpha\lambda \cdot q(R_*A^T \bar{\pi}^*)$.*

Proof. Feasibility is trivial. For the approximation ratio consider the following estimate, which uses that $\|R_*A^T \pi\|_\infty = \|W^{-1}A^T \pi\|_\infty$ for any π and the inequalities in (5):

$$\begin{aligned} q(R_*A^T \bar{\pi}) &\leq \|R_*A^T \bar{\pi}\|_\infty = \|W^{-1}A^T \bar{\pi}\|_\infty \leq \alpha \|W^{-1}A^T \bar{\pi}^*\|_\infty \leq \alpha \|W^{-1}A^T \pi^*\|_\infty \\ &= \alpha \|R_*A^T \pi^*\|_\infty \leq \alpha\lambda \cdot q(R_*A^T \pi^*). \end{aligned} \quad \square$$

One of our contributions, lies in showing that given a primitive that gives a very bad approximation, say logarithmic, to the *symmetric* transshipment problem, a $1 + \varepsilon$ -approximate solution even to the *asymmetric* transshipment problem can be obtained with poly-logarithmically many iterations of a gradient descent method. However, the objective of (3) is not differentiable, so we change the problem one more time by using the so-called soft-max (a.k.a. log-sum-exp or lse for short), which is a suitable approximation for $\|\cdot\|_\infty$. It is defined for vectors $v \in \mathbb{R}^d$ as

$$\text{lse}_\beta(v) := \frac{1}{\beta} \ln \left(\sum_{i \in [d]} e^{\beta v_i} \right),$$

where $\beta > 0$ is a parameter that controls the accuracy of the approximation of the infinity-norm at the expense of smoothness.⁷ This enables us to define a sufficiently smooth and accurate *potential function*

$$\Phi_\beta(\pi) := \text{lse}_\beta(R_*A^T \pi),$$

⁷Note that for the gradient $\nabla \text{lse}_\beta(x)$ of the soft-max, we have that $\nabla \text{lse}_\beta(x)_i = \frac{e^{\beta x_i}}{\sum_{j \in [d]} e^{\beta x_j}}$ for all $i \in [d]$.

for some parameter β that will be adapted during the course of the gradient descent. The algorithm takes as arguments the graph G , a demand vector b , a starting-solution π that is an α -approximate solution, an initial β that is appropriate for π , and the desired approximation guarantee ε .

Before showing the correctness and the running-time guarantee for the algorithm, we state some known facts (cf. for example [She13]) about $\text{lse}_\beta(\cdot)$. For x, y being d -dimensional vectors and β, t being positive scalars, it holds that

$$\text{lse}_{t,\beta}(x/t) = \text{lse}_\beta(x)/t \quad (6)$$

$$\nabla \text{lse}_{t,\beta}(x/t) = \nabla \text{lse}_\beta(x) \quad (7)$$

$$\|\nabla \text{lse}_\beta(x)\|_1 \leq 1 \quad (8)$$

$$\|\nabla \text{lse}_\beta(x) - \nabla \text{lse}_\beta(y)\|_1 \leq \beta \|x - y\|_\infty, \quad (9)$$

Note that the last inequality means that the gradient of $\text{lse}_\beta(\cdot)$ is Lipschitz continuous with Lipschitz constant β . Moreover, note that since $\Phi_\beta(\pi) := \text{lse}_\beta(R_* A^T \pi)$ is a composition of an linear and a convex function, it is again convex and thus $\Phi_\beta(\pi) \leq \Phi_\beta(0) + \nabla \Phi_\beta(\pi)^T \pi$. Together with duality (4), this yields

$$\begin{aligned} \Phi_\beta(\pi) &\leq \Phi_\beta(0) + \nabla \Phi_\beta(\pi)^T \pi = \frac{\ln(2m)}{\beta} + \nabla \text{lse}_\beta(R_* A^T \pi)^T R_* A^T \pi \\ &\leq \frac{\ln(2m)}{\beta} + p(W_* R_*^T \nabla \text{lse}_\beta(R_* A^T \pi)) \cdot q(R_* A^T \pi) \leq \frac{\ln(2m)}{\beta} + q(R_* A^T \pi), \end{aligned} \quad (10)$$

where the latter inequality follows from

$$p(W_* R_*^T \nabla \text{lse}_\beta(R_* A^T \pi)) \leq 1, \quad (11)$$

which is the asymmetric analogue of (8). A derivation can be found in Appendix C.

On the other hand convexity, and our choice of β , namely $\Phi_\beta(\pi) \in [\frac{4 \ln(2m)}{\varepsilon \beta}, \frac{5 \ln(2m)}{\varepsilon \beta}]$, gives

$$\nabla \Phi_\beta(\pi)^T \pi \geq \Phi_\beta(\pi) - \frac{\ln(2m)}{\beta} \geq \left(1 - \frac{\varepsilon}{4}\right) \Phi_\beta(\pi) \geq \left(1 - \frac{\varepsilon}{4}\right) q(R_* A^T \pi) \geq 0, \quad (12)$$

where the second to last inequality follows since

$$\text{lse}_\beta(R_* A^T \pi) = \frac{1}{\beta} \ln \left(\sum_{a \in A} e^{\beta(\pi_w - \pi_v)/w_a^+} + e^{\beta(\pi_v - \pi_w)/w_a^-} \right) \geq \frac{1}{\beta} \ln e^{\beta q(R_* A^T \pi)} = q(R_* A^T \pi).$$

Inequality (12) is what we used in the proof of Lemma 2.2 in Section 2.1, where we have shown that one can obtain a primal-dual $(1 + \varepsilon)$ -approximately optimal pair from the output potentials π of Algorithm `gradient_ust`. However, for the bound on the iterations, we left some proofs open that we will provide in the rest of this section.

We will first show that the potential function decreases by a multiplicative factor in each iteration. Notice that the gradient⁸ of the potential function $\Phi_\beta(\pi)$ takes the form

$$\nabla \Phi_\beta(\pi) = A R_*^T \nabla \text{lse}_\beta(R_* A^T \pi) = A R_*^T \gamma_\beta(\pi),$$

⁸We remark that $\nabla f(x)$ is short term for $\nabla_x f(x)$ for any function f , i.e., the gradient is w.r.t. the argument of the function following it, if the subscript is omitted.

where we denote $\gamma_\beta(\pi) := \nabla \text{lse}_\beta(R_* A^T \pi) \in \mathbb{R}^{2m}$.

Lemma 3.2. *Given $\pi \in \mathbb{R}^n$, let $P := [I - \pi b^T]$. For any $h \in \mathbb{R}^n$, it holds that*

$$\Phi_\beta(\pi - Ph) - \Phi_\beta(\pi) \leq -\tilde{b}^T h + \beta \|R_* A^T Ph\|_\infty^2,$$

where $\tilde{b} = P^T \nabla \Phi_\beta(\pi)$.

Proof. By convexity of the potential function, we have

$$\begin{aligned} \Phi_\beta(\pi - Ph) - \Phi_\beta(\pi) &\leq -\nabla \Phi_\beta(\pi - Ph)^T Ph \\ &= -\nabla \Phi_\beta(\pi)^T Ph + \left[\nabla \Phi_\beta(\pi) - \nabla \Phi_\beta(\pi - Ph) \right]^T Ph \\ &= -\tilde{b}^T h + \left[\gamma_\beta(\pi) - \gamma_\beta(\pi - Ph) \right]^T R_* A^T Ph. \end{aligned}$$

Using Hölder's inequality, we conclude that

$$\begin{aligned} \Phi_\beta(\pi - Ph) - \Phi_\beta(\pi) &\leq -\tilde{b}^T h + \|\gamma_\beta(\pi) - \gamma_\beta(\pi - Ph)\|_1 \|R_* A^T Ph\|_\infty \\ &\stackrel{(9)}{\leq} -\tilde{b}^T h + \beta \|R_* A^T Ph\|_\infty^2. \quad \square \end{aligned}$$

Corollary (Restatement of Corollary 2.3). *Given $\pi \in \mathbb{R}^n$ with $b^T \pi = 1$ and $\tilde{h} \in \mathbb{R}^n$ such that $\|W_-^{-1} A^T \tilde{h}\|_\infty \leq 1$ and suppose that β is such that $\varepsilon \beta \Phi_\beta(\pi) \leq 5 \ln(2m)$. Then for $\delta := \frac{\tilde{b}^T \tilde{h}}{\|R_* A^T P \tilde{h}\|_\infty}$, it holds that*

$$\Phi_\beta\left(\pi - \frac{\delta}{2\beta \|R_* A^T P \tilde{h}\|_\infty} P \tilde{h}\right) \leq \left[1 - \frac{\varepsilon \delta^2}{20 \ln(2m)}\right] \Phi_\beta(\pi).$$

Proof. Applying Lemma 3.2 and $\|R_* A^T P \tilde{h}\|_\infty \leq 1$, yields

$$\Phi_\beta\left(\pi - \frac{\delta}{2\beta \|R_* A^T P \tilde{h}\|_\infty} P \tilde{h}\right) - \Phi_\beta(\pi) \leq -\frac{\delta \tilde{b}^T \tilde{h}}{2\beta \|R_* A^T P \tilde{h}\|_\infty} + \frac{\delta^2}{4\beta} = -\frac{\delta^2}{4\beta}$$

By the choice of β , it follows that $-\frac{\delta^2}{4\beta} \leq -\frac{\varepsilon \delta^2}{20 \ln(2m)} \Phi_\beta(\pi)$. \square

Lemma (Restatement of Lemma 2.4). *Suppose that $0 < \varepsilon \leq 1/2$. Then Algorithm `gradient_ust` terminates within $O([\varepsilon^{-3} + \log \lambda + \log \alpha] \lambda^4 \alpha^2 \log n)$ iterations.*

Proof. Note that for all $x \in \mathbb{R}^n$, $\nabla_\beta \text{lse}_\beta(x) \leq 0$, i.e., lse_β is decreasing as function of β and thus the while-loop that scales β up does not increase $\Phi_\beta(\pi)$. Denoting π_0 and β_0 the initial values of β and π , respectively and π and β the values at termination, it follows by Corollary 2.3, that the potential decreases by a factor of $1 - \frac{\varepsilon \delta^2}{20 \ln(2m)} \leq 1 - \frac{\varepsilon^3}{640 \alpha^2 \lambda^4 \ln(2m)}$. Thus the number of iterations k can be bounded by

$$k \leq \log\left(\frac{\Phi_\beta(\pi)}{\Phi_{\beta_0}(\pi_0)}\right) \left[\log\left(1 - \frac{\varepsilon^3}{640 \alpha^2 \ln(2m)}\right)\right]^{-1} \leq \log\left(\frac{\Phi_{\beta_0}(\pi_0)}{\Phi_\beta(\pi)}\right) \frac{640 \alpha^2 \lambda^4 \ln(2m)}{\varepsilon^3}.$$

It remains to bound the quotient $\frac{\Phi_{\beta_0}(\pi^0)}{\Phi_{\beta}(\pi)}$. We will use our oracle for the symmetrized problem to produce a starting solution that is an $\alpha\lambda$ -approximate solution for the asymmetric problem. Using that the algorithm chooses β_0 such that $4\ln(2m) \leq \varepsilon\beta_0\Phi_{\beta_0}(\pi^0)$ shows that

$$\Phi_{\beta_0}(\pi^0) = \text{lse}_{\beta_0}(R_*A^T\pi^0) \stackrel{(10)}{\leq} q(R_*A^T\pi^0) + \frac{\ln(2m)}{\beta_0} \leq \alpha\lambda q(R_*A^T\pi^*) + \frac{\varepsilon\Phi_{\beta_0}(\pi^0)}{4}$$

and thus $\Phi_{\beta_0}(\pi^0) \leq \frac{\alpha\lambda q(R_*A^T\pi^*)}{1-\varepsilon/4}$. On the other hand $\Phi_{\beta}(\pi) \geq q(R_*A^T\pi) \geq q(R_*A^T\pi^*)$ and thus $\frac{\Phi_{\beta_0}(\pi^0)}{\Phi_{\beta}(\pi)} \leq \frac{\alpha\lambda}{1-\varepsilon/4} = O(\alpha\lambda)$. Since we can first run our gradient descent with $\varepsilon = 1/2$ until we obtain a constant factor approximation before we use the desired accuracy, it follows that the total number of iterations is in $O([\varepsilon^{-3} + \log \lambda + \log \alpha]\lambda^4\alpha^2 \log n)$. \square

3.1 Single-Source Shortest Paths

In the special case of single-source shortest paths, we have $b_v = 1$ for all $v \in V \setminus \{s\}$ and $b_s = 1 - n$ for the source s . In fact, it is the combination of $n - 1$ shortest s - t -path problems. Thus, the approximation guarantee from the previous section only holds on average over all sink nodes. In the following, we show how to use Algorithm 1 to obtain the distance from the source within a factor of $1 + \varepsilon$ for each node. To this end, we use a little more precision so that we can recognize the nodes for which we know the distance with sufficient accuracy using the tools we proposed above. We then concentrate on the other nodes and adapt the demand-vector b accordingly so that $b_v = 0$ for all good nodes v , i.e., the nodes for which we know the distance within a factor of $1 + \varepsilon$. We iterate until all nodes are good. It remains to show how to recognize good nodes and that a constant fraction of the nodes become good in each iteration.

Let $y^* \in \mathbb{Z}^n$ denote the vector of distances from the source and let $y \in \mathbb{R}^n$ be our current estimates of the distances. A node $v \in V$ is called ε -good if $y_v^*/(1 + \varepsilon) \leq y_v \leq y_v^*$. First, we show that we cannot be too close to the optimum when the stopping criterion is not reached yet.

Lemma 3.3. *If the termination criterion is not met for y yet, i.e., $\delta > \frac{\varepsilon}{4\alpha\lambda^2}$, then*

$$\Phi_{\beta \cdot b^T y}(\frac{y}{b^T y}) > \frac{1 + \varepsilon'}{b^T y^*}, \text{ where } \varepsilon' := \frac{\varepsilon^3}{640\lambda^4\alpha^2 \ln(2m)}.$$

Proof. Let $\pi' \in \mathbb{R}^n$ be the vector obtained by updating $\pi := \frac{y}{b^T y}$ as described in Algorithm `gradient_ust`, i.e., $\pi' = \pi - \delta P\tilde{h}/(2\beta\|R_*A^T P\tilde{h}\|_{\infty})$, where \tilde{h} is an α -approximate solution of the optimization problem $\max\{\tilde{b}^T h : \|W^{-1}A^T h\|_{\infty} \leq 1\}$. Then, using $\delta > \frac{\varepsilon}{4\alpha\lambda^2}$, Corollary 2.3 together with (6), and the fact that $q(R_*A^T\pi^*) = \frac{1}{b^T y^*}$, which we conclude from Observation 3.1, we get

$$(1 - \varepsilon')\Phi_{\beta \cdot b^T y}(\pi) > \left(1 - \frac{\varepsilon\delta^2}{20\ln(2m)}\right)\Phi_{\beta \cdot b^T y}(\pi) \geq \Phi_{\beta \cdot b^T y}(\pi') \geq q(R_*A^T\pi') \geq \frac{1}{b^T y^*}.$$

This implies $\Phi_{\beta \cdot b^T y}(\frac{y}{b^T y}) > \frac{1}{(1-\varepsilon')b^T y^*} \geq \frac{1+\varepsilon'}{b^T y^*}$. \square

Next, we show that a $1 + \varepsilon'$ on average guarantee implies that $b^T y^*$ reduces by at least a constant factor when we set $b_v = 0$ for all $2\varepsilon'$ -good nodes.

Lemma 3.4. *Let $b_v \geq 0$ for all $v \in V \setminus \{s\}$, $y \in \mathbb{R}^n$ with $q(R_* A^T y) = 1$, $y_s = 0$ and $b^T y \geq \frac{b^T y^*}{1 + \varepsilon'}$, and let $X := \{v \in V \setminus \{s\} : y_v < y_v^*/(1 + 2\varepsilon')\}$ for some $\varepsilon' \leq \frac{1}{4}$. Then, $\sum_{v \in X} b_v y_v^* \leq \frac{3}{4} b^T y^*$.*

Proof. We have that

$$\begin{aligned}
\varepsilon' \sum_{v \in X} b_v y_v &\leq \varepsilon' \sum_{v \in X} b_v y_v + (1 + \varepsilon') b^T y - b^T y^* \\
&= (1 + 2\varepsilon') \sum_{v \in X} b_v y_v + (1 + \varepsilon') \sum_{v \in V \setminus X} b_v y_v - b^T y^* \\
&< \sum_{v \in X} b_v y_v^* + (1 + \varepsilon') \sum_{v \in V \setminus X} b_v y_v - b^T y^* \\
&= (1 + \varepsilon') \sum_{v \in V \setminus X} b_v y_v - \sum_{v \in V \setminus X} b_v y_v^* \\
&\leq \varepsilon' \sum_{v \in V \setminus X} b_v y_v,
\end{aligned}$$

where the last inequality follows from the observation that $q(R_* A^T y) = 1$ guarantees that $y_v \leq y_v^*$ for all $v \in V$, the fact that $b_v \geq 0$ for all $v \in V \setminus X \setminus \{s\}$ and the assumption that $y_s = 0$. It follows that $\sum_{v \in V \setminus X} b_v y_v \geq \frac{1}{2} b^T y$ and thus for $\varepsilon' \leq \frac{1}{4}$, it follows that

$$\sum_{v \in X} b_v y_v^* = b^T y^* - \sum_{v \in V \setminus X} b_v y_v^* \leq (1 + \varepsilon') b^T y - \sum_{v \in V \setminus X} b_v y_v < \left(\frac{1}{2} + \varepsilon'\right) b^T y \leq \frac{3}{4} b^T y^*. \quad \square$$

We are now ready to state the algorithm.

Algorithm 2: `single_source_shortest_path` (G, s, ε)

- 1 Compute an α -approximate solution y , let $\hat{y} = 0$ and set $b = \mathbf{1} - n\mathbf{1}_s$
 - 2 Determine β s.t. $4 \ln(2m) < \varepsilon\beta\Phi_\beta(y) \leq 5 \ln(2m)$.
 - 3 **while** $b_s < 0$ **do**
 - 4 $\pi, \beta = \text{gradient_ust}(G, b, \frac{y}{b^T y}, \beta \cdot b^T y, \frac{\varepsilon^3}{640\alpha^2 \ln(2m)})$, $y = \frac{\pi}{q(R_* A^T \pi)}$
 - 5 **for each** $v \in V$ **with** $b_v = 1$ **do**
 - 6 Determine \tilde{h} with $\|W^{-1} A^T \tilde{h}\|_\infty = 1$ and
 $\tilde{b}^T \tilde{h} \geq \frac{1}{\alpha} \max\{\nabla\Phi_\beta(y)^T h : \|W^{-1} A^T h\|_\infty \leq 1\}$, let $\delta := \frac{\nabla\Phi_\beta(y)^T \tilde{h}}{R_* A^T P \tilde{h}}$
 - 7 **if** $\delta \leq \frac{\varepsilon}{8\alpha\lambda^2}$ **then**
 - 8 Set $b_v = 0$, $\hat{y}_v = y_v - y_s$ and $b_s \leftarrow b_s + 1$
 - 9 **return** \hat{y}
-

We show that the above algorithm outputs $1 + \varepsilon$ -approximate distances for all nodes:

Theorem (Restatement of Theorem 2.6). *Algorithm `single_source_shortest_path` computes y such that for each $v \in V$ it holds that $y_v^*/(1 + \varepsilon) \leq y_v \leq y_v^*$ with $\text{polylog}(n, \|w\|_\infty)$ calls to Algorithm `gradient_ust`.*

Proof. Consider one iteration of Algorithm `single_source_shortest_path`. W.l.o.g., we assume that $y_s = 0$. From Lemma 2.2 and weak duality, we get that $b^T y \geq b^T y^*/(1 + \varepsilon')$ for the returned solution y of Algorithm `gradient_ust`. Then, by Lemma 3.4, we conclude that there is a set of “good nodes” $U := \{v \in V : y_v^* \geq y_v \geq y_v^*/(1 + 2\varepsilon')\}$ such that $\sum_{v \in U} b_v y_v^* \geq \frac{1}{4} b^T y^*$ and it follows that

$$(\mathbf{1}_v - \mathbf{1}_s)^T y = y_v \geq \frac{y_v^*}{1 + 2\varepsilon'} = \frac{(\mathbf{1}_v - \mathbf{1}_s)^T y^*}{1 + 2\varepsilon'} \quad \text{for every } v \in U.$$

Since $\Phi_\beta(\frac{y}{b^T y}) \leq \frac{1 + \varepsilon'/4}{b^T y} \leq \frac{(1 + \varepsilon'/4)(1 + \varepsilon')}{b^T y^*} \leq \frac{1 + 2\varepsilon'}{b^T y^*}$, it follows with Lemma 3.3 that for an \tilde{h} such that $\|W_-^{-1} A^T \tilde{h}\|_\infty = 1$ and $\tilde{b}^T \tilde{h} < \frac{1}{\alpha} \max\{\nabla \Phi_\beta(y)^T h : \|W_-^{-1} A^T h\|_\infty \leq 1\}$, it holds that $\delta \leq \frac{\varepsilon}{4\alpha\lambda^2}$, where $\delta := \frac{\tilde{b}^T \tilde{h}}{\|R_* A^T P \pi\|_\infty}$. Thus we will recognize every node $v \in U$ as good in this iteration.⁹ It follows that, in each iteration, $b^T y^* = O(\|w\|_\infty n^2)$ gets reduced by at least a constant fraction 1/4 and thus after $\text{polylog}(n, \|w\|_\infty)$ many iterations, the while-loop in the algorithm terminates and \hat{y} contains a $(1 + \varepsilon)$ -approximate distance for every vertex $v \in V$. \square

3.2 Finding a Tree Solution

Recall from Lemma 2.2 that, given a near optimal dual solution, we can construct a primal approximate solution from a flow x_1 that routes the gradient and from a flow x_2 that routes the projection of the gradient. The natural choice for x_1 comes from the gradient of the softmax, i.e., $x_1 := R_*^T \nabla \text{lse}_\beta(R_* A^T \pi)$. However, the support of such an x_1 is rather dense in general. So, we cannot afford to make x_1 global knowledge in distributed models of computation or store it with near linear space in streaming models. Hence, we cannot directly transform x_1 to a tree solution using a global view on the problem. We could steer clear from this problem if x_1 had a small support, e.g., like a tree solution. In fact, Lemma 2.2 shows that we can afford to be a little worse in the objective value than the canonical choice for x_1 . This suggests to randomly sample a tree such that the expected flow over an edge equals the value of the canonical choice and thus to obtain the same objective value in expectation. Repetition and Markov’s inequality would then yield a suitable tree in $O(\frac{\log n}{\varepsilon})$ trials with high probability. Note that for this sampling the previously fixed orientation does not matter, so we reorient the edges such that each entry in the vector of the canonical choice for x_1 becomes non-negative for the sake of exposition. W.l.o.g., we may discard edges with zero flow. By the construction of the gradient from potential differences, the resulting graph then becomes acyclic and thus admits a topological order of the nodes. Each node v now samples one of its incoming arcs with probability $x_1(a)/x_1(\delta^{\text{in}}(v))$ for each $a \in \delta^{\text{in}}(v)$. By induction from the last to the first node in the topological order, one can show that the expected flow over arc a is then equal to x_a .

⁹Note that due to (7), it holds that $\nabla \Phi_\beta(y) = \nabla \Phi_{(\mathbf{1}_v - \mathbf{1}_s)^T y \beta}(\frac{y}{(\mathbf{1}_v - \mathbf{1}_s)^T y})$ and in particular the same gradient can be used for all $v \in V \setminus \{s\}$.

Moreover, we thereby obtain a tree because the first node in the topological order does not have any incoming arc. By the linearity of expectation in the equivalent linearized objective function (cf. Appendix B), the expected objective value of the sampled tree solution equals the objective value of the canonical solution for x_1 , which is upper bounded by 1. Any reorientation can be revoked now without changing the result. The edges of the tree can now be stored or communicated, e.g., to augment a spanner such that the resulting graph is guaranteed to have a tree solution that is $(1 + \varepsilon)$ -approximate for the problem with the original demands.

4 Applications

In the following we explain how to implement the gradient descent algorithm for computing $(1 + \varepsilon)$ -approximate shortest transshipment and SSSP in distributed and streaming models. Common to all these implementations is the use of sparse *spanners*. We restate the definition here.

Definition 2.7 (Spanner). *Given a weighted graph $G = (V, E, w)$ and $\alpha > 1$, an α -spanner of G is a subgraph $(V, E', w|_{E'})$, $E' \subseteq E$, in which distances are at most by factor α larger than in G .*

In other words, a spanner removes edges from G while approximately preserving distances. This implies that an optimal solution of an instance of the shortest transshipment problem on an α -spanner of the input graph is an α -approximate solution to the original problem. We give deterministic algorithms for computing spanners in distributed and streaming models in Appendix A.

4.1 Broadcast Congested Clique

Model. In the Broadcast Congested Clique model, the system consists of n fully connected nodes labeled by unique $O(\log n)$ -bit identifiers. Computation proceeds in synchronous rounds, where in each round, nodes may perform arbitrary local computations, broadcast (send) an $O(\log n)$ -bit message to the other nodes, and receive the messages from other nodes. The input is distributed among the nodes. The first part of the input of every node consists of its incident edges (given by their endpoints' identifiers) and their weights. The second part of the input is problem specific: for the transshipment problem, every node v knows its demand b_v and for SSSP v knows whether or not it is the source s . In both cases, every node knows $0 < \varepsilon \leq 1/2$ as well. Each node needs to compute its part of the output. For shortest transshipment every node needs to know a $(1 + \varepsilon)$ approximation of the optimum value, and for SSSP every node needs to know a $(1 + \varepsilon)$ -approximation of its distance to the source. The complexity of the algorithm is measured in the worst-case number of rounds until the computation is complete.

Implementing Algorithm `gradient_ust`. In the following, we explain how to implement our gradient descent algorithm for approximating the shortest transshipment.

Theorem 2.8. *For any $0 < \varepsilon \leq 1$, in the broadcast congested clique model a deterministic $(1 + \varepsilon)$ -approximation to the shortest transshipment problem in undirected graphs with non-negative weights can be computed in ε^{-3} polylog n rounds.*

The rest of this subsection is devoted to proving the theorem. In the following description of the algorithm we assume that we can use the following two algorithmic primitives. We will argue after the algorithm's description that they can be carried out within the stated running time bounds.

- (A) Given $\beta > 0$ and $\pi \in \mathbb{R}^n$ known to every node, compute, and make known to every node, $\Phi_\beta(\pi)$ and $\nabla\Phi_\beta(\pi)$ using a constant number of rounds.
- (B) Given an α -spanner G' of $G^- = (V, E, w^-)$ and $q \in \mathbb{R}^n$ known to every node, compute, at every node, $\tilde{h} \in \mathbb{R}^n$ such that $\|W_-^{-1}A^T\tilde{h}\|_\infty = 1$ and $q^T\tilde{h} \geq \max \frac{1}{\alpha}\{q^T h : \|W_-^{-1}A^T h\|_\infty \leq 1\}$, using only local computation.

We now describe how to implement the gradient descent algorithm. We will maintain the invariant that each node knows π and the current value of β at the beginning of each iteration of the algorithm.

-
1. Make n , m , b , λ , and the set of node identifiers known to all nodes in a single round by each node v broadcasting its identifier, b_v , degree in the input graph, and maximum ratio between forward and backward cost of incident edges.
 2. Next, construct an α -spanner G' of $G^- = (V, E, w^-)$ with $\alpha = 2\lceil \log n \rceil - 1$ in $O(\log^2 n)$ rounds using the algorithm of Corollary A.1 in Appendix A.
 3. Locally at every node, compute an α -approximate solution π to the symmetrized shortest transshipment problem with demands b using (B) with $q := b$.
 4. Compute and make known to every node $\Phi_\beta(\pi)$ using (A) in a constant number of rounds.
 5. Locally at every node, check if $\frac{4\ln(4m)}{\varepsilon\beta} \geq \Phi_\beta(\pi)$. If not, locally at every node scale up β by $5/4$ and proceed with Step 4.
 6. Compute and make known to every node $\nabla\Phi_\beta(\pi)$ using (A) in a constant number of rounds.
 7. Locally at every node, compute $P = I - \pi b^T$ and $\tilde{b} = P^T \nabla\Phi_\beta(\pi)$.
 8. Locally at every node, determine \tilde{h} such that $\|W_-^{-1}A^T\tilde{h}\|_\infty = 1$ and $\tilde{b}^T\tilde{h} \geq \frac{1}{\alpha} \max\{\tilde{b}^T h : \|W_-^{-1}A^T h\|_\infty \leq 1\}$ using (B) with $q := \tilde{b}$, performing only local computation.
 9. Compute $\|R_*A^TP\tilde{h}\|_\infty$ in a single round. (Every node first locally computes $P\tilde{h}$ and determines from π the maximum value of $(R_*A^TP\tilde{h})_{(u,v)}$ among its incoming forward and backward arcs (u, v) . The nodes then broadcast these values such that every node can determine the maximum among the received values and its own value.)

10. Locally at every node, compute $\delta := \frac{\tilde{b}^T \tilde{h}}{\|R_* A^T P \tilde{h}\|_\infty}$ and check if $\delta \geq \frac{\varepsilon}{8\alpha\lambda^2}$. If yes, locally at every node, compute $\pi := \pi - \frac{\delta}{2\beta\|R_* A^T P \tilde{h}\|_\infty} P \tilde{h}$ and proceed with the next iteration at Step 4.
 11. At termination, compute $f(\pi) := \frac{\pi}{q(R_* A^T \pi)}$ as the output in a single round. (Every node first locally determines from π the maximum value of $(R_* A^T \pi)_{(u,v)}$ among its incoming forward and backward arcs (u,v) . The nodes then broadcast these values such that every node can determine the maximum among the received values, its own value, and 0).
-

Apart from repetitions of Step 4 for scaling up β we need a constant number of rounds per iteration. As there are ε^{-3} polylog n iterations by Lemma 2.4 and we scale up β at most $O(\log n)$ times over the course of the algorithm, we use ε^{-3} polylog n rounds as promised in Theorem 2.8. It remains to show how to perform primitives (A) and (B).

Primitive (A). For every forward arc $(u,v) \in E^+$ define the *forward stretch* under potentials π by

$$s_\pi^+(u,v) := \frac{\pi_v - \pi_u}{w^+(u,v)}$$

and for every backward arc $(u,v) \in E^-$ define the *backward stretch* under potentials π by

$$s_\pi^-(u,v) := \frac{\pi_v - \pi_u}{w^-(u,v)}.$$

Note that $R_* A^T \pi$ is the vector containing first the forward stretches and then the backward stretches of the arcs under potential π . For every node v , let \mathbf{a}_v and \mathbf{b}_v , respectively, be the following sum over its incoming forward and outgoing arcs:

$$\begin{aligned} \mathbf{a}_v &:= \sum_{(u,v) \in E^+} e^{\beta s_\pi^+(u,v)} + \sum_{(v,u) \in E^-} e^{\beta s_\pi^-(v,u)} \\ \mathbf{b}_v &:= \sum_{(u,v) \in E^+} \frac{e^{\beta s_\pi^+(u,v)}}{w^+(u,v)} + \sum_{(u,v) \in E^-} \frac{e^{\beta s_\pi^-(u,v)}}{w^-(u,v)} - \sum_{(v,u) \in E^+} \frac{e^{\beta s_\pi^+(v,u)}}{w^+(v,u)} - \sum_{(v,u) \in E^-} \frac{e^{\beta s_\pi^-(v,u)}}{w^-(v,u)}. \end{aligned}$$

and let \mathbf{s} be the following sum over all arcs

$$\mathbf{s} := \sum_{(u,v) \in E^+} e^{\beta s_\pi^+(u,v)} + \sum_{(v,u) \in E^-} e^{\beta s_\pi^-(v,u)} = \sum_{v \in V} \mathbf{a}_v.$$

Thus,

$$\Phi_\beta(\pi) = \text{lse}_\beta(R_* A^T \pi) = \frac{1}{\beta} \ln \left(\sum_{(u,v) \in E^+} e^{\beta s_\pi^+(u,v)} + \sum_{(v,u) \in E^-} e^{\beta s_\pi^-(v,u)} \right) = \frac{\ln \mathbf{s}}{\beta}$$

and, for every node v ,

$$\begin{aligned} \nabla\Phi_\beta(\pi)_v &= AR_*^T \nabla \text{lse}_\beta(R_* A^T \pi) = \\ & \sum_{(u,v) \in E^+} \frac{e^{\beta s_\pi^+(u,v)}}{w^+(u,v) \cdot \mathbf{s}} + \sum_{(u,v) \in E^-} \frac{e^{\beta s_\pi^-(u,v)}}{w^-(u,v) \cdot \mathbf{s}} - \sum_{(v,u) \in E^+} \frac{e^{\beta s_\pi^+(v,u)}}{w^+(v,u) \cdot \mathbf{s}} - \sum_{(v,u) \in E^-} \frac{e^{\beta s_\pi^-(v,u)}}{w^-(v,u) \cdot \mathbf{s}} = \frac{\mathbf{b}_v}{\mathbf{s}}. \end{aligned}$$

As every node v knows its incident arcs and their respective weights, it can locally compute \mathbf{a}_v and \mathbf{b}_v , respectively. In one round of communication all nodes can learn the value \mathbf{a}_v of every other node v . Once these values are known, each node can locally compute $\Phi_\beta(\pi)$ as well as \mathbf{s} and $\nabla\Phi_\beta(\pi)_v$. Simultaneously for every node v , we send the latter value to all its neighbors in one round such that every node knows $\nabla\Phi_\beta(\pi)_v$ afterwards.

Primitive (B). We use the fact that an α -spanner G' of $G^- = (V, E, w^-)$ is known to every node. In particular, every node can locally construct the node-arc incidence matrix A' and the diagonal weight matrix W' of G' . Thus, each node can locally compute an optimal solution h' to the linear program $\max\{q^T h : \|(W')^{-1}(A')^T h\|_\infty = 1\}$.¹⁰ We now claim that $\tilde{h} := h'/\alpha$ has the desired properties.

Lemma 4.1. *Let $q \in \mathbb{R}^n$ and let G' be an α -spanner of $G^- = (V, E, w^-)$. Let $A, A', W_-,$ and W' denote the node-arc incident matrices and the weight matrices of G^- and G' , respectively. Let h' be an optimal solution to the linear program $\max\{q^T h : \|(W')^{-1}(A')^T h\|_\infty \leq 1\}$ and set $\tilde{h} := h'/\alpha$. Then $\|W_-^{-1} A^T \tilde{h}\|_\infty \leq 1$ and $\alpha q^T \tilde{h} \geq \max\{q^T h : \|W_-^{-1} A^T h\|_\infty \leq 1\}$.*

Proof. As the spanner is a subgraph, $\max\{q^T h : \|(W')^{-1}(A')^T h\|_\infty \leq 1\} \geq \max\{q^T h : \|W_-^{-1} A^T h\|_\infty \leq 1\}$ and thus $\alpha q^T \tilde{h} = q^T h' \geq \max\{q^T h : \|W_-^{-1} A^T h\|_\infty \leq 1\}$. Let $(u, v) \in E^-$ be a backward arc and denote by p a shortest path between u and v in G . Then the stretch of (u, v) in G under h' is

$$\begin{aligned} |(W_-^{-1} A^T h')_{(u,v)}| &= \frac{|h'_v - h'_u|}{w(u,v)} \leq \frac{\sum_{\{u',v'\} \in p} |h'_{v'} - h'_{u'}|}{w(u,v)} \\ &= \frac{\sum_{(u',v') \in p} w(u',v') |(W')^{-1}(A')^T h'_{(u',v')}|}{w(u,v)} \leq \frac{\sum_{(u',v') \in p} w(u',v')}{w(u,v)} \leq \alpha. \end{aligned}$$

Thus, $\|W_-^{-1} A^T h'\|_\infty \leq \alpha$ and consequently $\|W_-^{-1} A^T \tilde{h}\|_\infty \leq 1$ for $\tilde{h} = h'/\alpha$. \square

Note that if $\|W_-^{-1} A^T \tilde{h}\|_\infty < 1$, we can simply “scale up” \tilde{h} to get a solution with $\|W_-^{-1} A^T \tilde{h}\|_\infty = 1$.

Implementing Algorithm `single_source_shortest_path`. After computing an α -spanner, we can implement Algorithm `single_source_shortest_path` using primitives (A) and (B) in a straightforward way. The algorithm internally uses our implementation of `gradient_ust` with an increased precision of $\varepsilon' = \varepsilon^3 / (640\alpha^2 \ln(2m))$. We thus arrive at the following theorem.

¹⁰In particular, we can locally run any shortest transshipment algorithm.

Theorem 2.9. *For any $0 < \varepsilon \leq 1$, in the broadcast congested clique model a deterministic $(1 + \varepsilon)$ -approximation to single-source shortest paths in undirected graphs with non-negative weights can be computed in ε^{-9} polylog n rounds.*

4.2 Broadcast Congest Model

Model. The broadcast congest model differs from the broadcast congested clique in that communication is restricted to edges that are present in the input graph. That is, node v receives the messages sent by node w if and only if $\{v, w\} \in E$. All other aspects of the model are identical to the broadcast congested clique. We stress that this restriction has significant impact, however: Denoting the hop diameter¹¹ of the input graph by D , it is straightforward to show that $\Omega(D)$ rounds are necessary to solve the transshipment problem. Moreover, it has been established that $\Omega(\sqrt{n}/\log n)$ rounds are required even on graphs with $D \in O(\log n)$ [DHK⁺12]. Both of these bounds apply to randomized approximation algorithms (unless the approximation ratio is not polynomially bounded in n).

Solving the Transshipment Problem. A straightforward implementation of our algorithm in this model simply simulates the broadcast congested clique. A folklore method to simulate (global) broadcast is to use “pipelining” on a breadth-first-search (BFS) tree.

Lemma 4.2 (cf. [Pel00]). *Suppose each $v \in V$ holds $m_v \in \mathbb{Z}_{\geq 0}$ messages of $O(\log n)$ bits each, for a total of $M = \sum_{v \in V} m_v$ strings. Then all nodes in the graph can receive these M messages within $O(M + D)$ rounds.*

Proof Sketch. It is easy to construct a BFS tree in $O(D)$ rounds (rooted at, e.g., the node with smallest identifier) and obtain an upper bound $d \in [D, 2D]$ by determining the depth of the tree and multiplying it by 2. By a convergecast, we can determine $|M|$, where each node in the tree determines the total number of messages in its subtree. We define a total order on the messages via lexicographical order on node identifier/message pairs.¹² Then nodes flood their messages through the tree, where a flooding operation for a message may be delayed by those for other messages which are smaller w.r.t. the total order on the messages. On each path, a flooding operation may be delayed once for each flooding operation for a smaller message. Hence, this operation completes within $O(D + |M|)$ rounds, and using the knowledge on d and M , nodes can safely terminate within $O(d + |M|) = O(D + |M|)$ rounds. \square

We obtain the following corollary to Theorem 2.8.

Corollary 4.3. *For any $0 < \varepsilon \leq 1$, in the broadcast congest model a deterministic $(1 + \varepsilon)$ -approximation of (1) can be computed in $\tilde{O}(\varepsilon^{-3}n)$ rounds.*

Proof. Simulate a round on the broadcast congested clique using Lemma 4.2, i.e., with parameters $|M| = n$ and $D \leq n$. Applying Theorem 2.8, the claim follows. \square

¹¹That is, the diameter of the unweighted graph $G = (V, E)$.

¹²W.l.o.g., we assume identifier/message pairs to be different.

Special Case: Single-Source Shortest Paths. The near-linear running time bound of Corollary 4.3 is far from the $\tilde{\Omega}(\sqrt{n} + D)$ lower bound, which also applies to the single-source shortest path problem. However, for this problem there is an efficient reduction to a smaller *skeleton graph*, implying that we can match the lower bound up to polylogarithmic factors. The skeleton graph is given as an overlay network on a subset $V' \subseteq V$ of the nodes, where each node in V' learns its incident edges and their weights.

Theorem 4.4 ([HKN16]). *Given any weighted undirected network G and source node $s \in V$, there is a $\tilde{O}(\sqrt{n})$ -round deterministic distributed algorithm in the broadcast congest model¹³ that computes an overlay network $G' = (V', E')$ with edge weights $w' : E' \rightarrow \{1, \dots, \text{poly } n\}$ and some additional information for every node with the following properties.*

- $|V'| = \tilde{O}(\varepsilon^{-1}\sqrt{n})$ and $s \in V'$.
- For $\varepsilon' := \varepsilon/7$, each node $v \in V$ can infer a $(1 + \varepsilon)$ -approximation of its distance to s from $(1 + \varepsilon')$ -approximations of the distances between s and each $t \in V'$.

This reduces the problem to a graph of roughly \sqrt{n} nodes, to which we can apply the previous simulation approach.

Corollary 2.10. *For any $0 < \varepsilon \leq 1$, in the broadcast congest model a deterministic $(1 + \varepsilon)$ -approximation to single-source shortest paths in undirected graphs with non-negative weights can be computed in $\tilde{O}(\varepsilon^{-O(1)}(\sqrt{n} + D))$ rounds.*

Proof. We apply Theorem 4.4. Subsequently, we use Lemma 4.2 to simulate rounds of the broadcast congested clique on the overlay network, taking $\tilde{O}(\varepsilon^{-O(1)}\sqrt{n} + D)$ rounds per simulated round. Using Corollary 4.3 for $\varepsilon' \in \Theta(\varepsilon)$, we obtain a $(1 + \varepsilon')$ -approximation to the distances from each $t \in V'$ to s in the overlay. After broadcasting these distances using Lemma 4.2 again, all nodes can locally compute a $(1 + \varepsilon)$ -approximation of their distance to s . The total running time is $\tilde{O}(\varepsilon^{-O(1)}(\sqrt{n} + D))$. \square

4.3 Multipass Streaming

Model. In the *Streaming* model the input graph is presented to the algorithm edge by edge as a “stream” without repetitions and the goal is to design algorithms that use as little space as possible. In the *Multipass Streaming* model, the algorithm is allowed to make several such passes over the input stream and the goal is to design algorithms that need only a small number of passes (and again little space). For graph algorithms, the usual assumption is that the edges of the input graph are presented to the algorithm in arbitrary order.

¹³All communication of the algorithm in [HKN16] meets the constraint that in each round, each node sends the same message to all neighbors (which is the difference between the broadcast congest and the standard congest model used in [HKN16]).

Implementing Algorithm `gradient_ust`. In the following we explain how to implement our gradient descent algorithm for approximating the shortest transshipment.

Theorem 2.11. *For any $0 < \varepsilon \leq 1$, in the multipass streaming model a deterministic $(1 + \varepsilon)$ -approximation to the shortest transshipment problem in undirected graphs with non-negative weights can be computed in ε^{-3} polylog n passes with $O(n \log n)$ space.*

The rest of this subsection is devoted to proving the theorem. In the algorithm we will use space $O(n \log n)$ to permanently store certain information. We will perform all operations of the algorithm within an additional “temporary space” of size $O(n \log n)$, i.e., at any time the sum of the permanent space and the temporary space is $O(n \log n)$. In the following description of the algorithm we assume that we can use the following two algorithmic primitives. We will argue after the algorithm’s description that they can be carried out within the stated running time bounds.

- (A) Given stored $\beta > 0$ and $\pi \in \mathbb{R}^n$, compute $\Phi_\beta(\pi)$ and $\nabla \Phi_\beta(\pi)$ using a single pass and $O(n)$ temporary space.
- (B) Given a stored α -spanner G' of $G^- = (V, E, w^-)$ and stored $q \in \mathbb{R}^n$, compute $\tilde{h} \in \mathbb{R}^n$ such that $\|W_-^{-1} A^T \tilde{h}\|_\infty = 1$ and $q^T \tilde{h} \geq \frac{1}{\alpha} \max\{q^T h : \|W_-^{-1} A^T h\|_\infty \leq 1\}$ internally (i.e. without additional passes) with $O(n \log n)$ temporary space

We reserve space $O(n \log n)$ to permanently store the following information throughout the algorithm:

- An $O(\log n)$ -spanner G' of $G^- = (V, E, w^-)$ of size $O(n \log n)$.
- An n -dimensional vector π for the solution maintained by the gradient descent algorithm.
- An n -dimensional vector s for partial sums of edge weights.
- An n -dimensional vector b for the input demands.
- Scalars $\alpha, \beta, \varepsilon, m$ and n .

We now describe how to implement Algorithm 1 in the multipass streaming model.

1. Compute and store n, m, b, λ in a single pass with $O(1)$ temporary space.
2. Construct and store an α -spanner G' of $G^- = (V, E, w^-)$ of size $O(n \log n)$ with $\alpha = 2\lceil \log n \rceil - 1$ in $O(\log n)$ passes with $O(n \log n)$ temporary space using the algorithm of Corollary A.2.
3. Internally, compute and store an α -approximate solution π to the min-cost transshipment problem with demands b using (B) with $q := b$ with $O(n \log n)$ temporary space.
4. Compute and store $\Phi_\beta(\pi)$ using (A) in a single pass with $O(n)$ temporary space.

5. Internally, check if $\frac{4\ln(4m)}{\varepsilon\beta} \geq \Phi_\beta(\pi)$. If not, scale up β by 5/4 and proceed with Step 3.
 6. Compute and store $\nabla\Phi_\beta(\pi)$ using (A) in a single pass with $O(n)$ temporary space.
 7. Internally, compute and store $\tilde{b} := P^T\nabla\Phi_\beta(\pi) = \nabla\Phi_\beta(\pi) - b(\pi^T\nabla\Phi_\beta(\pi))$ (where $P = I - \pi b^T$) using $O(n)$ temporary space.
 8. Internally, compute and store \tilde{h} such that $\|W_-^{-1}A^T\tilde{h}\|_\infty = 1$ and $\tilde{b}^T\tilde{h} \geq \frac{1}{\alpha} \max\{\tilde{b}^T h : \|W_-^{-1}A^T h\|_\infty \leq 1\}$ using (B) with $q := \tilde{b}$ with $O(n \log n)$ temporary space.
 9. Internally, compute and store $r := P\tilde{h} = \tilde{h} - \pi(\tilde{b}^T\tilde{h})$ (where $P = I - \pi b^T$) using $O(n)$ temporary space.
 10. Compute $\|R_*A^TP\tilde{h}\|_\infty = \|R_*A^Tr\|_\infty$ in a single pass with $O(1)$ temporary space. (Keep a temporary variable for computing the maximum, initialized to $-\infty$. Every time an edge is read, determine its forward arc (u, v) and its backward arc (v, u) , and compute $(R_*A^Tr)_{(u,v)}$ and $(R_*A^Tr)_{(v,u)}$, respectively, from the stored π . Then store the larger of these two values if it exceeds the temporary maximum.)
 11. Internally, compute $\delta := \frac{\tilde{b}^T\tilde{h}}{\|R_*A^TP\tilde{h}\|_\infty}$ and check if $\delta \geq \frac{\varepsilon}{8\alpha\lambda^2}$ with $O(1)$ temporary space. If yes, internally compute $\pi := \pi - \frac{\delta}{2\beta\|R_*A^TP\tilde{h}\|_\infty}P\tilde{h} = \pi - \frac{\delta}{2\beta\|R_*A^Tr\|_\infty}r$ with $O(n)$ temporary space and proceed with the next iteration at Step 4.
 12. At termination, compute $f(\pi) := \frac{\pi}{q(R_*A^T\pi)}$ as the output in a single pass with $O(1)$ temporary space. (Keep a temporary variable for computing the maximum, initialized to 0. Every time an edge is read, determine its forward arc (u, v) and its backward arc (v, u) , and compute $(R_*A^T\pi)_{(u,v)}$ and $(R_*A^T\pi)_{(v,u)}$, respectively, from the stored π . Then store the larger of these two values if it exceeds the temporary maximum.)
-

Using the algorithmic primitives (A) and (B), it is clear that we never exceed the promised space bound of $O(n \log n)$. Apart from repetitions of Step 4 for scaling up β we need a constant number of passes per step in the algorithm above. As there are ε^{-3} polylog n iterations by Lemma 2.4 and we scale up β at most $O(\log n)$ times over the course of the algorithm, we use ε^{-3} polylog n passes as promised in Theorem 2.11. It remains to show how to perform primitives (A) and (B).

Primitive (A). We use the notation introduced in the description of Primitive (A) in Section 4.1. We again exploit that

$$\Phi_\beta(\pi) = \frac{\ln \mathfrak{s}}{\beta}$$

and, for every node v ,

$$\nabla\Phi_\beta(\pi)_v = \frac{\mathfrak{b}_v}{\mathfrak{s}}.$$

To implement the computation of $\Phi_\beta(\pi)$ and $\nabla\Phi_\beta(\pi)$ in a single pass with $O(n)$ temporary space, it is sufficient to compute \mathfrak{s} , and, for every node v , \mathfrak{b}_v in a single pass. For this purpose we keep a variable for each of these values that is initialized to 0 before the pass. Every time we read an edge with forward and backward weights from the stream, we first determine its forward arc (u, v) and its backward arc (v, u) . We then compute $s_\pi^+(u, v)$ and $s_\pi^-(v, u)$ and add $e^{\beta s_\pi^+(u, v)} + e^{\beta s_\pi^-(v, u)}$ to the variable for \mathfrak{s} , $\frac{e^{\beta s_\pi^+(u, v)}}{w^+(u, v)} - \frac{e^{\beta s_\pi^-(v, u)}}{w^-(v, u)}$ to the variable for \mathfrak{b}_v , and $\frac{e^{\beta s_\pi^-(v, u)}}{w^-(v, u)} - \frac{e^{\beta s_\pi^+(u, v)}}{w^+(u, v)}$ to the variable for \mathfrak{b}_u . After the pass is finished, the variables have the desired value and we can internally compute $\Phi_\beta(\pi)$ and $\nabla\Phi_\beta(\pi)$ with $O(n)$ temporary space.

Primitive (B). We use the fact that we have stored an α -spanner G' of $G^- = (V, E, w^-)$. We internally construct the node-arc incidence matrix A' and the diagonal weight matrix W' of G' and compute an optimal solution h' to the linear program $\max\{q^T h : \|(W')^{-1}(A')^T h\|_\infty = 1\}$. We can compute a shortest transshipment using $O(n \log n)$ temporary space by enumerating all feasible solutions and checking for optimality.¹⁴ By Lemma 4.1, $\tilde{h} := h'/\alpha$ has the desired properties.

Implementing Algorithm `single_source_shortest_path` After computing an α -spanner, we can implement Algorithm `single_source_shortest_path` using primitives (A) and (B) in a straightforward way. The algorithm internally uses our implementation of `gradient_ust` with an increased precision of $\varepsilon' = \varepsilon^3/(640\alpha^2 \ln(2m))$. We thus arrive at the following theorem.

Theorem 2.12. *For any $0 < \varepsilon \leq 1$, in the multipass streaming model, deterministic $(1 + \varepsilon)$ -approximation to single-source shortest paths in undirected graphs with non-negative weights can be computed in ε^{-9} polylog n passes with $O(n \log n)$ space.*

5 Further Related Work

In the following we review further results on (approximate) SSSP in the non-centralized models considered in this paper.

In the *congest model* of distributed computing, SSSP on unweighted graphs can be computed exactly in $O(D)$ rounds by distributed breadth-first search [Pel00]. For weighted graphs, the only non-trivial algorithm known is the distributed version of Bellman-Ford [Bel58, For56] using $O(n)$ rounds. In terms of approximation, Elkin [Elk06] showed that computing an α -approximate SSSP tree requires $\Omega((n/\alpha)^{1/2}/\log n + D)$ rounds. Together with many other lower bounds, this was strengthened in [DHK⁺12] by showing that computing a (poly n)-approximation of the distance between two fixed nodes in a weighted undirected graph requires $\Omega(\sqrt{n}/\log n + D)$ rounds. The lower bounds were complemented by two SSSP algorithms: a randomized $O(\alpha \log \alpha)$ -approximation using $\tilde{O}(n^{1/2+1/\alpha} + D)$ rounds [LPS13] and a randomized $(1 + o(1))$ -approximation using $\tilde{O}(n^{1/2}D^{1/4} + D)$ rounds [Nan14]. Both results were improved upon in [HKN16] by a deterministic algorithm that computes $(1 + o(1))$ -approximate SSSP

¹⁴Alternatively, we could run one of the classic linear-space algorithms for the shortest transshipment problem.

in $n^{1/2+o(1)} + D^{1+o(1)}$ rounds. A recent hop set construction of Elkin and Neiman [EN16] improves the running time for computing shortest paths from multiple sources.

The *congested clique* model [LPP⁺05] has seen increasing interest in the past years as it highlights the aspect of limited bandwidth in distributed computing, yet excludes the possibility of explicit bottlenecks (e.g., a bridge that would limit the flow of information between the parts of the graph it connects to $O(\log n)$ bits per round in the congest model). For weighted graphs, SSSP can again be computed exactly in $O(n)$ rounds. The first approximation was given by Nanongkai [Nan14] with a randomized algorithm for computing $(1 + o(1))$ -approximate SSSP in $\tilde{O}(\sqrt{n})$ rounds. All-pair shortest paths on the congested Clique can be computed deterministically in $\tilde{O}(n^{1/3})$ rounds for an exact result and in $O(n^{0.158})$ rounds for a $(1 + o(1))$ -approximation [CKK⁺15]. The time for computing all-pairs shortest paths exactly has subsequently been improved to $O(n^{0.2096})$ rounds [LG16]. The hop set construction of [HKN16] gives a deterministic algorithm for computing $(1 + o(1))$ -approximate SSSP in $n^{o(1)}$ rounds. A recent hop set construction of Elkin and Neiman [EN16] improves the running time for computing shortest paths from multiple sources. We note that both, the approaches of [HKN16] and [EN16], and our approach can actually operate in the more restricted *broadcast congested clique* and *broadcast congest* models, in which in each round, each node sends the *same* $O(\log n)$ bits to all other nodes or all its neighbors, respectively.

In the *streaming model*, two approaches were known for (approximate) SSSP before the algorithm of [HKN16]. First, shortest paths up to distance d can be computed using d passes and $O(n)$ space in unweighted graphs by breadth-first search. Second, approximate shortest paths can be computed by first obtaining a sparse spanner and then computing distances on the spanner without additional passes [FKM⁺05, EZ06, FKM⁺08, Bas08, Elk11]. This leads, for example, to a randomized $(2k - 1)$ -approximate all-pairs shortest paths algorithm using 1 pass and $\tilde{O}(n^{1+1/k})$ space for any integer $k \geq 2$ in unweighted undirected graphs, which can be extended to a $(1 + \varepsilon)(2k - 1)$ -approximation in weighted undirected graphs for any $\varepsilon > 0$ at the cost of increasing the space by a factor of $O(\varepsilon^{-1} \log R)$. In unweighted undirected graphs, the spanner construction of [EZ06] can be used to compute $(1 + o(1))$ -approximate SSSP using $O(1)$ passes and $O(n^{1+o(1)})$ space. The hop set construction of [HKN16] gives a deterministic algorithm for computing $(1 + o(1))$ -approximate SSSP in weighted undirected graphs using $n^{o(1)} \log R$ passes and $n^{1+o(1)} \log R$ space. Using randomization, this was improved to $n^{o(1)}$ passes and $O(n \log^2 n)$ space by Elkin and Neiman [EN16]. These upper bounds are complemented by a lower bound of $n^{1+\Omega(1/p)}/(\text{poly } p)$ space for all algorithms that decide in p passes if the distance between two fixed nodes in an unweighted undirected graph is at most $2(p + 1)$ for any $p = O(\log n / \log \log n)$ [GO13]. Note that this lower bound in particular applies to all algorithms that provide a $1 + \varepsilon$ approximation for $\varepsilon < 1/2(p + 1)$ in integer weighted graphs, as this level of precision requires to compute shortest paths for small enough distances exactly.

References

- [Bas08] Surender Baswana. “Streaming algorithm for graph spanners - single pass and constant processing time per edge”. In: *Information Processing Letters* 106.3 (2008), pp. 110–114 (cit. on pp. 29, 35).
- [Bel58] Richard Bellman. “On a Routing Problem”. In: *Quarterly of Applied Mathematics* 16.1 (1958), pp. 87–90 (cit. on pp. 6, 28).
- [Ber09] Aaron Bernstein. “Fully Dynamic (2 + epsilon) Approximate All-Pairs Shortest Paths with Fast Query and Close to Linear Update Time”. In: *Symposium on Foundations of Computer Science (FOCS)*. 2009, pp. 693–702 (cit. on p. 3).
- [BS07] Surender Baswana and Sandeep Sen. “A Simple and Linear Time Randomized Algorithm for Computing Sparse Spanners in Weighted Graphs”. In: *Random Structures & Algorithms* 30.4 (2007). Announced at ICALP’03, pp. 532–563 (cit. on pp. 34, 35).
- [CKK⁺15] Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. “Algebraic Methods in the Congested Clique”. In: *Symposium on Principles of Distributed Computing (PODC)*. 2015, pp. 143–152 (cit. on p. 29).
- [CKM⁺11] Paul Christiano, Jonathan A. Kelner, Aleksander Mądry, Daniel A. Spielman, and Shang-Hua Teng. “Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs”. In: *Symposium on Theory of Computing (STOC)*. 2011, pp. 273–282 (cit. on p. 3).
- [CMS⁺17] Michael B. Cohen, Aleksander Mądry, Piotr Sankowski, and Adrian Vladu. “Negative-Weight Shortest Paths and Unit Capacity Minimum Cost Flow in $\tilde{O}(m^{10/7} \log W)$ Time”. In: *Symposium on Discrete Algorithms (SODA)*. 2017 (cit. on p. 3).
- [Coh00] Edith Cohen. “Polylog-Time and Near-Linear Work Approximation Scheme for Undirected Shortest Paths”. In: *Journal of the ACM* 47.1 (2000). Announced at STOC’94, pp. 132–166 (cit. on pp. 3, 5).
- [DHK⁺12] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. “Distributed Verification and Hardness of Distributed Approximation”. In: *SIAM Journal on Computing* 41.5 (2012). Announced at STOC’11, pp. 1235–1265 (cit. on pp. 3, 4, 12, 24, 28).
- [DS08] Samuel I. Daitch and Daniel A. Spielman. “Faster approximate lossy generalized flow via interior point algorithms”. In: *Symposium on Theory of Computing (STOC)*. 2008, pp. 451–460 (cit. on p. 3).
- [EK72] Jack Edmonds and Richard M. Karp. “Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems”. In: *Journal of the ACM* 19.2 (1972), pp. 248–264 (cit. on p. 6).

- [Elk06] Michael Elkin. “An Unconditional Lower Bound on the Time-Approximation Trade-off for the Distributed Minimum Spanning Tree Problem”. In: *SIAM Journal on Computing* 36.2 (2006). Announced at STOC’04, pp. 433–456 (cit. on p. 28).
- [Elk11] Michael Elkin. “Streaming and Fully Dynamic Centralized Algorithms for Constructing and Maintaining Sparse Spanners”. In: *ACM Transactions on Algorithms* 7.2 (2011). Announced at ICALP’07, 20:1–20:17 (cit. on p. 29).
- [EN16] Michael Elkin and Ofer Neiman. “Hopsets with Constant Hopbound, and Applications to Approximate Shortest Paths”. In: *Symposium on Foundations of Computer Science (FOCS)*. 2016 (cit. on pp. 3, 5, 29).
- [EZ06] Michael Elkin and Jian Zhang. “Efficient algorithms for constructing $(1 + \epsilon, \beta)$ -spanners in the distributed and streaming models”. In: *Distributed Computing* 18.5 (2006), pp. 375–385 (cit. on p. 29).
- [FKM⁺05] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. “On graph problems in a semi-streaming model”. In: *Theoretical Computer Science* 348.2-3 (2005). Announced at ICALP’04, pp. 207–216 (cit. on p. 29).
- [FKM⁺08] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. “Graph Distances in the Data-Stream Model”. In: *SIAM Journal on Computing* 38.5 (2008). Announced at SODA’05, pp. 1709–1727 (cit. on p. 29).
- [For56] Lester R. Ford. *Network Flow Theory*. Tech. rep. P-923. The Rand Corporation, 1956 (cit. on pp. 6, 28).
- [FT87] Michael L. Fredman and Robert Endre Tarjan. “Fibonacci heaps and their uses in improved network optimization algorithms”. In: *Journal of the ACM* 34.3 (1987). Announced at FOCS’84, pp. 596–615 (cit. on p. 3).
- [GO13] Venkatesan Guruswami and Krzysztof Onak. “Superlinear Lower Bounds for Multipass Graph Processing”. In: *Conference on Computational Complexity (CCC)*. 2013, pp. 287–298 (cit. on pp. 5, 29).
- [Gol95] Andrew V. Goldberg. “Scaling Algorithms for the Shortest Paths Problem”. In: *SIAM Journal on Computing* 24.3 (1995). Announced at SODA’93, pp. 494–504 (cit. on p. 6).
- [HKN14] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. “Decremental Single-Source Shortest Paths on Undirected Graphs in Near-Linear Total Update Time”. In: *Symposium on Foundations of Computer Science (FOCS)*. 2014, pp. 146–155 (cit. on p. 3).
- [HKN16] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. “A deterministic almost-tight distributed algorithm for approximating single-source shortest paths”. In: *Symposium on Theory of Computing (STOC)*. 2016, pp. 489–498 (cit. on pp. 3, 4, 25, 28, 29).

- [HKT⁺15] Thomas Dueholm Hansen, Haim Kaplan, Robert Endre Tarjan, and Uri Zwick. “Hollow Heaps”. In: *International Colloquium on Automata, Languages and Programming (ICALP)*. 2015, pp. 689–700 (cit. on p. 3).
- [KLO⁺14] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. “An Almost-Linear-Time Algorithm for Approximate Max Flow in Undirected Graphs, and its Multicommodity Generalizations”. In: *Symposium on Discrete Algorithms (SODA)*. 2014, pp. 217–226 (cit. on p. 3).
- [KR90] Richard M. Karp and Vijaya Ramachandran. “Parallel Algorithms for Shared-Memory Machines”. In: *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*. The MIT Press, 1990, pp. 869–942 (cit. on p. 3).
- [KV00] Bernhard Korte and Jens Vygen. *Combinatorial Optimization*. Springer, 2000 (cit. on p. 5).
- [LG16] François Le Gall. “Further Algebraic Algorithms in the Congested Clique Model and Applications to Graph-Theoretic Problems”. In: *International Symposium on Distributed Computing (DISC)*. 2016, pp. 57–70 (cit. on p. 29).
- [LPP⁺05] Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg. “Minimum-Weight Spanning Tree Construction in $O(\log \log n)$ Communication Rounds”. In: *SIAM Journal on Computing* 35.1 (2005). Announced at SPAA’03, pp. 120–131 (cit. on p. 29).
- [LPS13] Christoph Lenzen and Boaz Patt-Shamir. “Fast Routing Table Construction Using Small Messages”. In: *Symposium on Theory of Computing (STOC)*. 2013, pp. 381–390 (cit. on p. 28).
- [LS14] Yin Tat Lee and Aaron Sidford. “Path Finding Methods for Linear Programming: Solving Linear Programs in $\tilde{O}(\sqrt{\text{rank}})$ Iterations and Faster Algorithms for Maximum Flow”. In: *Symposium on Foundations of Computer Science (FOCS)*. 2014, pp. 424–433 (cit. on pp. 3, 6).
- [Mađ13] Aleksander Mađry. “Navigating Central Path with Electrical Flows: From Flows to Matchings, and Back”. In: *Symposium on Foundations of Computer Science (FOCS)*. 2013, pp. 253–262 (cit. on p. 3).
- [MPV⁺15] Gary L. Miller, Richard Peng, Adrian Vladu, and Shen Chen Xu. “Improved Parallel Algorithms for Spanners and Hopsets”. In: *Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 2015, pp. 192–201 (cit. on p. 3).
- [Nan14] Danupon Nanongkai. “Distributed Approximation Algorithms for Weighted Shortest Paths”. In: *Symposium on Theory of Computing (STOC)*. 2014, pp. 565–573 (cit. on pp. 3, 28, 29).
- [Orl93] James B. Orlin. “A Faster Strongly Polynomial Minimum Cost Flow Algorithm”. In: *Operations Research* 41.2 (1993). Announced at STOC’88, pp. 338–350 (cit. on p. 6).

- [Pel00] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Philadelphia, PA: SIAM, 2000 (cit. on pp. 24, 28).
- [RTZ05] Liam Roditty, Mikkel Thorup, and Uri Zwick. “Deterministic Constructions of Approximate Distance Oracles and Spanners”. In: *International Colloquium on Automata, Languages and Programming (ICALP)*. 2005, pp. 261–272 (cit. on pp. 5, 35).
- [Sch03] Alexander Schrijver. *Combinatorial Optimization*. Springer, 2003 (cit. on p. 5).
- [She13] Jonah Sherman. “Nearly Maximum Flows in Nearly Linear Time”. In: *Symposium on Foundations of Computer Science (FOCS)*. 2013, pp. 263–269 (cit. on pp. 3, 15).
- [She17] Jonah Sherman. “Generalized Preconditioning and Network Flow Problems”. In: *Symposium on Discrete Algorithms (SODA)*. 2017 (cit. on pp. 5, 6).
- [Tho99] Mikkel Thorup. “Undirected Single-Source Shortest Paths with Positive Integer Weights in Linear Time”. In: *Journal of the ACM* 46.3 (1999). Announced at FOCS’97, pp. 362–394 (cit. on p. 3).

A Deterministic Spanner Computation in Congested Clique and Multipass Streaming Model

For $k \in \mathbb{Z}_{>0}$, a simple and elegant randomized algorithm computing a $(2k - 1)$ -spanner with $O(kn^{1+1/k})$ edges in expectation was given by Baswana and Sen [BS07]. For the sake of completeness, we restate it here.

1. Initially, each node is a singleton *cluster*: $R_1 := \{\{v\} \mid v \in V\}$.
 2. For $i = 1, \dots, k - 1$ do:
 - (a) Each cluster from R_i is *marked* independently with probability $n^{-1/k}$. R_{i+1} is defined to be the set of clusters marked in phase i .
 - (b) If v is a node in an unmarked cluster:
 - i. Define Q_v to be the set of edges that consists of the lightest edge from v to each cluster in R_i it is adjacent to.
 - ii. If v is not adjacent to any marked cluster, all edges in Q_v are added to the spanner.
 - iii. Otherwise, let u be the closest neighbor of v in a marked cluster. In this case, v adds to the spanner the edge $\{v, u\}$ and all edges $\{v, w\} \in Q_v$ with $w(v, w) < w(v, u)$ (break ties by neighbor identifiers). Also, let X be the cluster of u . Then $X := X \cup \{v\}$, i.e., v *joins* the cluster of u .
 3. Each $v \in V$ adds, for each $X \in R_k$ it is adjacent to, the lightest edge connecting it to X to the spanner.
-

It is easy to see that the algorithm selects $O(kn^{1+1/k})$ expected edges into the spanner: In each iteration, each node v sorts its incident clusters in order of ascending weight of the lightest edge to them and elects for each cluster, up to the first sampled one, the respective lightest edge into the spanner. Because this order is independent of the randomness used in this iteration, v selects $O(n^{1/k})$ edges in expectation and $O(n^{1/k} \log n)$ edges with high probability.¹⁵ The same bound applies to the final step, as $|R_k| \in O(n^{1/k})$ in expectation and $|R_k| \in O(n^{1/k} \log n)$ with high probability. Moreover, this observation provides a straightforward derandomization of the algorithm applicable in our model: Instead of picking R_{i+1} in iteration i randomly, we consider the union E_i over all nodes v of the lightest $O(n^{1/k} \log n)$ edges in Q_v . By a union bound, with high probability we can select R_{i+1} such that (i) $|R_{i+1}| \leq n^{-1/k}|R_i|$ and (ii) each node selects only $O(n^{1/k} \log n)$ edges into the spanner in this iteration. In particular, such a choice must exist, and it can be computed from R_i and E_i alone. With this argument we deterministically obtain a spanner of size $O(kn^{1+1/k} \log n)$.

¹⁵That is, for any fixed constant choice of $c > 0$, the number of selected edges is bounded by $O(n^{1/k} \log n)$ with probability at least $1 - 1/n^c$.

-
1. Initially, each node is a singleton *cluster*: $R_1 := \{\{v\} \mid v \in V\}$.
 2. For $i = 1, \dots, k - 1$ do for each node v :
 - (a) Define Q_v to be the set of edges that consists of the lightest edge from v to each cluster in R_i it is adjacent to.
 - (b) Broadcast the set Q'_v of the lightest $O(n^{1/k} \log n)$ edges in Q_v .
 - (c) For $w \in V$, denote by $X_w \in R_i$ the cluster so that $v \in X_w$. Locally compute $R_{i+1} \subseteq R_i$ such that (i) $|R_{i+1}| \leq n^{-1/k}|R_i|$ and (ii) for each $w \in V$ for which $Q'_v \neq Q_w$, it holds that $X_w \in R_{i+1}$ or Q'_v contains an edge connecting to some $X \in R_{i+1}$.
 - (d) Update clusters and add edges to the spanner as the original algorithm would, but for the computed choice of R_{i+1} .
 3. Each $v \in V$ adds, for each $X \in R_k$ it is adjacent to, the lightest edge connecting it to X .
-

A slightly stronger bound of $O(kn^{1+1/k})$ on the size of the spanner, matching the expected value from above up to constant factors, can be obtained in this framework by using the technique of deterministically finding *early hitting sets* developed by Roditty, Thorup, and Zwick [RTZ05].

Note that, as argued above, the selection of R_{i+1} in Step 2(c) is always possible and can be done deterministically, provided that R_i is known. Because R_1 is simply the set of singletons and each node computes R_{i+1} from R_i and the Q'_v , this holds by induction. We arrive at the following result.

Corollary A.1 (of [BS07] and [RTZ05]). *For $k \in \mathbb{Z}_{>0}$, in the broadcast congested clique a $(2k - 1)$ -spanner of size $O(kn^{1+1/k})$ can be deterministically computed and made known to all nodes in $O(kn^{1/k} \log n)$ rounds.*

Proof. The bound of $\alpha = 2k - 1$ follows from the analysis in [BS07], which does not depend on the choice of the R_i . For the round complexity, observe that all computations can be performed locally based on knowing Q'_v for all nodes $v \in V$ in each iteration. As $|Q'_v| \in O(n^{1/k} \log n)$ for each iteration and each v , the claim follows. \square

By similar arguments, we can also get an algorithm in the multipass streaming algorithm with comparable guarantees. We remark that, apart from the property of being deterministic, this was already stated in [Bas08] as a simple consequence of [BS07].

Corollary A.2 (of [BS07] and [RTZ05]). *For $k \in \mathbb{Z}_{>0}$, in the multipass streaming model a $(2k - 1)$ -spanner of size $O(kn^{1+1/k})$ can be deterministically computed in k passes using $O((k + \log n)n^{1+1/k})$ space.*

B Primal-Dual Pair

Lemma B.1. *The dual of $\min\{p(W_*x) : Ax = b\}$ is $\max\{b^T y : q(R_*A^T y) \leq 1\}$. Moreover, strong duality holds.*

Proof. Observe that the minimization problem is in fact a linear program, since it can be rewritten as

$$\min\{p(W_*x) : Ax = b\} = \min\{w^{+T} x^+ + w^{-T} x^- : Ax^+ - Ax^- = b, x^+, x^- \geq 0\}.$$

Thus, a dual exists for which strong duality holds. It is given by

$$\max\{b^T y : A^T y \leq w^+, -A^T y \leq w^-\} = \max\{b^T y : q(R_*A^T y) \leq 1\},$$

which proves the claim. □

C Inequality (11)

Observe that, for any vector $x \in \mathbb{R}^m$,

$$p(W_*R_*^T \nabla \text{lse}(R_*x)) = \frac{\sum_{i=1}^m \max\{e^{x_i/w_i^+} - \frac{w_i^+}{w_i^-} e^{-x_i/w_i^-}, -\frac{w_i^-}{w_i^+} e^{x_i/w_i^+} + e^{-x_i/w_i^-}\}}{\sum_{i=1}^m e^{x_i/w_i^+} + e^{-x_i/w_i^-}},$$

which is a convex combination of terms of the form

$$\frac{\max\{e^{x_i/w_i^+} - \frac{w_i^+}{w_i^-} e^{-x_i/w_i^-}, -\frac{w_i^-}{w_i^+} e^{x_i/w_i^+} + e^{-x_i/w_i^-}\}}{e^{x_i/w_i^+} + e^{-x_i/w_i^-}}.$$

Basic calculus shows that these terms are bounded between 0 and 1. Thus, their convex combination is at most 1 as well, which shows that Inequality (11) holds.