A Combinatorial Algorithm for the Semi-Discrete Optimal Transport Problem

Pankaj K. Agarwal[1], **Sharath Raghvendra**[2], **Pouyan Shirzadian**[3], **and Keegan Yao**[1]

[1]Duke University, [2]North Carolina State University, [3]Virginia Tech

Abstract

Optimal Transport (OT, also known as the Wasserstein distance) is a popular metric for comparing probability distributions and has been successfully used in many machine-learning applications. In the semi-discrete 2-Wasserstein problem, we wish to compute the cheapest way to transport all the mass from a continuous distribution μ to a discrete distribution ν in $\mathbb{R}^d$ for $d \geq 1$, where the cost of transporting unit mass between points a and b is $d(a, b) = \|a - b\|^2$. When both distributions are discrete, a simple combinatorial framework has been used to find the exact solution (see e.g. [Orlin, STOC 1988]). In this paper, we propose a combinatorial framework for the semi-discrete OT, which can be viewed as an extension of the combinatorial framework for the discrete OT but requires several new ideas. We present a new algorithm that given μ and ν in $\mathbb{R}^2$ and a parameter $\varepsilon > 0$, computes an ε-additive approximate semi-discrete transport plan in $O(n^4 \log n \log \frac{1}{\varepsilon})$ time (in the worst case), where n is the support-size of the discrete distribution ν and we assume that the mass of μ inside a triangle can be computed in $O(1)$ time. Our algorithm is significantly faster than the known algorithms, and unlike many numerical algorithms, it does not make any assumptions on the smoothness of μ. As an application of our algorithm, we describe a data structure to store a large discrete distribution μ (with support size N) using $O(N)$ space so that, given a query discrete distribution ν (with support size k), an ε-additive approximate transport plan can be computed in $O(k^3 \sqrt{N} \log \frac{1}{\varepsilon})$ time in 2 dimensions. Our algorithm and data structure extend to higher dimensions as well as to p-Wasserstein problem for any $p \geq 1$.

1 Introduction

Optimal Transport (OT) is a powerful metric for comparing probability distributions and is used in many machine-learning applications. The semi-discrete optimal transport problem asks for the cheapest transport plan to transport mass from (a possibly continuous) distribution μ that is stored compactly (say, using a deep neural network) to a discrete distribution ν. In recent years, the semi-discrete OT has been used in data mining [31], image processing [28, 29, 33, 40], computational biology [53], variational inference [7], blue noise generation [21, 51], optics [46], solving PDEs [30, 34], and generative models [8, 9, 19].

More formally, a *semi-discrete transport plan* τ between a continuous probability distribution μ defined over a compact support $A \subset \mathbb{R}^d$ and a discrete distribution ν with a support set B of n points in $\mathbb{R}^d$ is a distribution over $A \times B$ whose marginals are dominated by μ and ν, i.e., $\tau \colon A \times B \to \mathbb{R}_{\geq 0}$ is a *transport plan* between μ and ν where $\sum_{b \in B} \tau(a, b) \leq \mu(a)$ for all $a \subseteq A$ and $\int_A \tau(a, b)\, da \leq \nu(b)$ for all $b \in B$. A transport plan τ is *complete* if τ transports all mass of μ to ν. For any fixed $p \geq 1$, in

---

*The authors are listed in alphabetical order.

the *semi-discrete p-Wasserstein problem*, the goal is to compute a complete transport plan τ between μ and ν minimizing the cost $\mathcal{C}(\tau) := \int_A \sum_{b \in B} \|a - b\|^p \tau(a, b) \, da$. An optimal plan is referred to as a W_p-*OT plan*[1]. For any parameter $\varepsilon > 0$, a transport plan τ is called an ε-*close W_p-OT plan* if $\mathcal{C}(\tau) \leq \mathcal{C}(\tau^*) + \varepsilon$, where τ^* is a W_p-OT plan.

An optimal solution for the semi-discrete OT problem can be compactly represented as the weighted Voronoi diagram (also called the Laguerre diagram) with respect to a weight assignment $y(b)$ on every point $b \in B$ [12]. The choice of weights guarantees that mass at b is equal to the mass of μ inside the Voronoi cell (also called the Laguerre cell) of b and the optimal transport plan simply transports the mass at b to the mass inside its Voronoi cell. For 2 dimensions, the weighted Voronoi diagram under the squared Euclidean distance (also known as the power diagram) can be constructed in $O(n \log n)$ time [24, 57]. For higher dimensions $d > 2$, the construction time would be $O(n^{\lceil (d+1)/2 \rceil})$ [11].

The compact and connected nature of the semi-discrete optimal transport plans makes them attractive for many ML applications; for instance, they can help achieve stability in training GANs and avoid issues such as discontinuities and mode collapse [10, 19, 55], they improve the mapping between continuous latent spaces and discrete data in Variational Autoencoders (VAEs) [9], and also have been used in diffusion-based generative models [42]. However, there are no known exact algorithms for computing semi-discrete optimal transport. Additionally, computing an ε-close transport plan is known to be #P-Hard with respect to d and $\log 1/\varepsilon$, i.e., an algorithm with an execution time that is polynomial in d and $\log 1/\varepsilon$ seems unlikely [58]. Due to the intractability of the semi-discrete optimal transport in high dimensions, researchers have considered taking n samples from the model, which is a continuous distribution μ, and computing an optimal discrete transport plan between the empirical distribution μ_n defined on the n samples and ν [43]. It has been shown that this empirical p-Wasserstein distance converges to the true semi-discrete p-Wasserstein distance at a rate of $n^{-1/2p}$ [22, 35, 50]; note that the rate of convergence does not depend on the dimension. However, the optimal transport plan from samples is not necessarily a good approximation of the semi-discrete transport plan and may cause biased gradients [14, 19].

For the discrete OT problem, there are several near-optimal scalable exact and approximation algorithms [1, 4, 5, 20, 25, 26, 36, 39, 52, 54], some of which extend to very high dimensions. For the semi-discrete OT in low dimensions, despite extensive work, scalable algorithms to find optimal semi-discrete transport plans remain elusive. There are algorithms to compute an ε-close semi-discrete transport plan using numerical solvers [12, 15, 17, 21, 37, 38, 41, 48], entropic regularization [6, 16, 32], and multiscale approaches [40, 44]. The execution time of all these algorithms is exponential in both d and $\log 1/\varepsilon$. Furthermore, their convergence relies on a smoothness parameter of μ. For instance, a notable numerical algorithm by Oliker and Prussner [48] assumes that for a point $b \in B$, a small change in $y(b)$ will change the mass of μ inside the Voronoi cell of b by a proportionately small amount [45, Remark 22]. Under this assumption, their algorithm executes $\text{poly}(n, 1/\varepsilon)$ iterations, where each iteration requires the computation of a weighted Voronoi diagram which takes $n^{O(d)}$ time. Their algorithm slows down when μ is non-smooth and does not even converge when μ is a discrete distribution. Furthermore, these methods approximate the transport cost but the transport plan that they compute may not be an approximation of the optimal weighted Voronoi diagram.

Recently, Agarwal *et al.* [5] described a *cost-scaling* paradigm to compute an ε-close semi-discrete transport plan. Their algorithm executes $\log \frac{\Delta}{\varepsilon}$ scales, where Δ is the diameter of $A \cup B$. Within each scale, they create an instance of the discrete OT problem of size $O(n^5)$ in $\mathbb{R}^2$ (and $n^{O(d)}$ in $\mathbb{R}^d$). Using any existing strongly polynomial primal-dual discrete OT solver, such as the algorithm by Orlin [49], their algorithm computes an exact discrete OT plan for each instance and updates the weights for B. The OT plan computed in the final scale is an ε-close transport plan. They show that the weight assigned to any point $b \in B$ is ε away from the optimal weight assignment. The overall execution time of this algorithm is $O(n^9 \log \frac{\Delta}{\varepsilon})$ in $\mathbb{R}^2$ and $n^{O(d)} \log \frac{\Delta}{\varepsilon}$ in d-dimensions. Furthermore, in the limit, the transport plan of their algorithm converges to the optimal weighted Voronoi diagram. Note that their algorithm does not make any assumptions on μ.

Our Contributions. The following theorem states our main result.

Theorem 1.1. *Let μ be a continuous probability distribution defined on a compact set $A \subset \mathbb{R}^2$, ν a discrete probability distribution with a support $B \subset \mathbb{R}^2$ of size n, and $\varepsilon > 0$ a parameter. Suppose*

---

[1]The p-Wasserstein cost of τ is defined as $\mathcal{C}(\tau)^{1/p}$, and the p-Wasserstein distance between μ and ν is the p-Wasserstein cost of a W_p-OT plan.

there exists an oracle that, given a triangle φ, returns the mass of μ inside φ in Φ time. Then, an ε-close W_2-OT plan between μ and ν can be computed in $O(n^3(\Phi + n \log n) \log \frac{\Delta}{\varepsilon})$ time in the worst-case, where Δ is the diameter of $A \cup B$.

Similar to the algorithm by Agarwal *et al.* [5], our algorithm is also based on a cost-scaling approach and executes $O(\log \frac{\Delta}{\varepsilon})$ scales. However, the algorithm within each scale is different. Agarwal *et al.* create a discrete OT instance with n^4 vertices and n^5 edges and use an exact discrete OT algorithm to solve this instance. Instead, we extend the combinatorial primal-dual framework of discrete OT to the continuous space and present an algorithm to find the desired semi-discrete transport plan in $O(n^4 \log n)$ time in the worst case. There are several challenges in extending the combinatorial framework to semi-discrete settings, and overcoming these challenges is one of the main technical contributions of the paper.

In more detail, in each scale, our algorithm maintains a (partial) transport plan, iteratively computes a set of augmenting paths, and augments the transport plan along such paths until all of the mass is transported. In order to assist in finding augmenting paths, we define a residual graph of size $O(n^3)$ in the continuous space. Algorithms for discrete OT maintain dual weights for all points in $A \cup B$. Unlike in the discrete setting where the vertex set is fixed, the vertex set of our residual graph includes "continuous regions", which evolve over time, and the vertex set of the residual graph changes. Therefore, we are able to maintain weights only for points in B and not for the regions in A. Our primal-dual framework (especially the definition of admissibility in Section 2), as well as our algorithm (in particular Sections 3.2 and 3.4) contain a number of novel ideas, carefully designed to address various challenges that arise due to the dynamically changing continuous regions of the residual graph. Like all existing algorithms, our algorithm also requires access to an oracle that, given a query triangle, returns the mass of μ inside the triangle. We note that Dijkstra's shortest path algorithm has been extended to continuous space [47], but we are not aware of any previous work that extends a combinatorial discrete OT framework to continuous space.

Our algorithm extends to any dimension $d \geq 2$ and any $p \geq 1$ in a straightforward way. For $d > 2$, the algorithm in Theorem 1.1 can be shown to have an execution time of $O(n^{d+1}(\Phi + n \log n) \log \frac{\Delta}{\varepsilon})$. Note that the runtime of Oliker-Prussner's algorithm has a factor $1/\varepsilon$ while ours has only $\log 1/\varepsilon$. Unlike their algorithm, ours does not make any assumptions on the smoothness of μ. Furthermore, similar to Agarwal et al. [5], our transport plan approximates an optimal weighted Voronoi diagram within a small additive factor.

One consequence of our algorithm is that we can use it to design a data structure that answers ε-close optimal transport queries efficiently. More precisely, consider a large distribution μ with support of size N. We can preprocess this distribution into a data structure that can return the total weight of points inside a query triangle in $O(N^{1-1/d})$ time [59]. By using Theorem 1.1, we can report an ε-additive approximate transport plan to any query distribution ν with support of size k in $O(k^{d+1}N^{1-1/d} \log \frac{\Delta}{\varepsilon})$ time. Note that this query time is sub-linear in N. There has been some work on computing 1-Wasserstein distance approximately in sub-linear time and for answering nearest neighbor queries under 1-Wasserstein distance [13], but these algorithms have larger error and do not extend to 2-Wasserstein distance.

Theorem 1.2. *Let μ be a discrete probability distribution with a support $A \subset \mathbb{R}^d$ of size N. The distribution μ can be preprocessed, in $O(N \log N)$ time, into an $O(N)$ size data structure so that for a discrete probability distribution ν with a support $B \subset \mathbb{R}^d$ of size k, and a parameter $\varepsilon > 0$, an ε-close W_2-OT plan between μ and ν can be computed in $O(k^{d+1}(N^{1-1/d} + k \log k) \log \frac{\Delta}{\varepsilon})$ time, where Δ is the diameter of $A \cup B$.*

If the points in the supports of μ and ν have integer coordinates and the masses on them are rational numbers, we can adapt our data structure in Theorem 1.2 to compute an exact W_p-OT plan between μ and ν in $O(k^{d+1}(N^{1-1/d} + k \log k) \log \frac{\Delta}{\varepsilon})$ time for any fixed even value of p.

We also note that in 2-dimensions, for $k < N^{1/4}$, we can use the data structure to compute an exact discrete OT plan for any distribution ν in $O(k^3 N^{1/2} \log \Delta)$ time that is faster than any existing OT algorithm that takes $(Nk)^{1+o(1)}$ time [3, 18].

For simplicity in presentation, we restrict our presentation to $d = 2$ and $p = 2$.
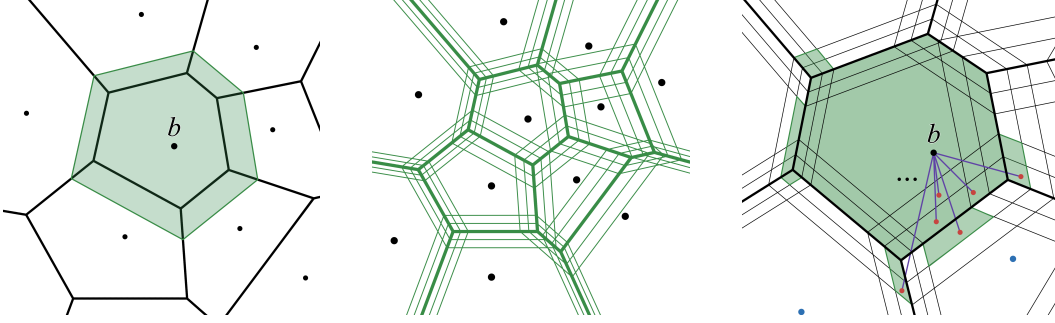
Figure 1: (left) The δ-expanded Voronoi cell V_b^δ of b (green shaded area), (middle) the partitioning $\mathcal{X}_\delta$, and (right) the green region shows the mass of μ that is transported to b, the red points show the representative points of regions, and the purple segments show the compressed transport plan $\hat{\tau}$.

2 Combinatorial Framework

In this section, we extend the combinatorial framework used by discrete OT algorithms to the semi-discrete settings. Let μ be a continuous probability distribution defined over a compact support $A \subset \mathbb{R}^2$ and ν be a discrete distribution with a support set B of n points in $\mathbb{R}^2$.

Weighted Voronoi diagram. For any pair of points $a, b \in \mathbb{R}^2$, let $\mathrm{d}(a, b) = \|a - b\|^2$. Given a weight function $y : B \to \mathbb{R}_{\geq 0}$ on the points in B, the *weighted distance* $d_y(a, b)$ from a point $b \in B$ to any point $a \in \mathbb{R}^2$ is defined as $\mathrm{d}_y(a, b) = \mathrm{d}(a, b) - y(b)$. For a point $b \in B$, the *Voronoi cell* of b is the locus of all points with b as their weighted nearest neighbor; more formally,

$$\mathrm{Vor}_y(b) = \{a \in \mathbb{R}^2 \mid \mathrm{d}_y(a, b) \leq \mathrm{d}_y(a, b'), \forall b' \in B\}.$$

The *weighted Voronoi diagram* $\mathrm{VD}_y(B)$ of the points B with weights $y(\cdot)$ is the decomposition of $\mathbb{R}^2$ induced by Voronoi cells. There exists a weight function $y(\cdot)$ for B such that $\mu(\mathrm{Vor}_y(b)) = \nu(b)$ for every point $b \in B$, and an optimal semi-discrete transport plan transports the mass of ν at each point $b \in B$ to the mass of μ in $\mathrm{Vor}_y(b) \cap A$ [12].

δ**-expanded Voronoi cell.** Consider a weight function $y(\cdot)$ for the points in B. For any point $b \in B$ and a parameter $\delta > 0$, consider the following weight function y_b^δ.

$$y_b^\delta(b') = \begin{cases} y(b') + \delta, & b' = b, \\ y(b'), & b' \neq b. \end{cases} \tag{1}$$

The *δ-expanded Voronoi cell* of b, denoted by V_b^δ, is simply the Voronoi cell of b in the weighted Voronoi diagram $\mathrm{VD}_{y_b^\delta}(B)$ of the point set B with weights $y_b^\delta(\cdot)$. See Figure 1 (left).

δ**-feasibility.** Any (possibly partial) transport plan τ between μ and ν along with a weight function $y(\cdot)$ for the points in B is *δ-feasible* if

(F1) for any pair $(a, b) \in A \times B$ with $\tau(a, b) > 0$, the point a lies inside the 2δ-expanded Voronoi cell $V_b^{2\delta}$.

For any δ-feasible transport plan $\tau, y(\cdot)$, if τ is a complete transport plan between μ and ν, then $\tau, y(\cdot)$ is called a δ-*optimal transport plan*. Recall that any optimal transport plan transports the mass at b to its weighted Voronoi cell. In a δ-optimal transport plan, however, the mass of each point $b \in B$ is transported inside a 2δ-expansion of the Voronoi cell of b. This introduces an additive increase of at most 2δ in the cost of the transport plan.

Lemma 2.1. *Any δ-optimal transport plan $\tau, y(\cdot)$ between μ and ν is 2δ-close.*

See Appendix A for a proof. Next, given a δ-feasible transport plan $\tau, y(\cdot)$, we define a residual graph and an augmenting path. We also introduce the process of augmenting τ along an augmenting path, which allows us to increase the mass transported by τ.

Residual graph. Given a δ-feasible (possibly partial) transport plan $\tau, y(\cdot)$, we construct a residual graph $\mathcal{G}_\delta$ by first partitioning the support A of μ into regions to form the vertex set of $\mathcal{G}_\delta := \mathcal{G}(\tau, y, \delta)$ and then defining a set of directed edges.

Vertex set. For each point $b \in B$, consider the Voronoi cell of b and its δ- and 2δ-expansions V_b^δ and $V_b^{2\delta}$. Let $\mathcal{X}_\delta$ denote the arrangement [2] of these $3n$ cells across all n points of B. See Figure 1 (middle). For each region φ in this arrangement, pick an arbitrary *representative point* r_φ inside φ and assign it a mass of $\mu_{r_\varphi} := \mu(\varphi)$, where $\mu(\varphi) = \int_\varphi \mu(a)\,da$ denotes the mass of μ inside φ. Let A_δ denote the set of representative points of all regions in $\mathcal{X}_\delta$. The vertex set of $\mathcal{G}_\delta$ is the point set $A_\delta \cup B$ along with a source vertex s and a sink vertex t. We refer to any point $b \in B$ whose mass is not fully transported by τ as a *free* point and define its *excess* as the amount of mass of b that is not transported by τ, i.e., $\mathrm{ex}(b) = \nu(b) - \int_{a \in A} \tau(a, b)\,da$. Similarly, any point $r_\varphi \in A_\delta$ is free if τ does not fully transport the mass into the region φ, and its excess is defined as $\mathrm{ex}(r_\varphi) = \mu_{r_\varphi} - \sum_{b' \in B} \tau(\varphi, b')$.

Edge set. For each pair $(r_\varphi, b) \in A_\delta \times B$, if $\tau(\varphi, b) > 0$, we add a *backward edge* directed from r_φ to b in the residual graph. Furthermore, if $r_\varphi \in V_b^{2\delta}$, we add a *forward edge* directed from b to r_φ in $\mathcal{G}_\delta$. Additionally, we add a forward edge from the source s to every free point $b \in B$ and a backward edge directed from every free vertex r_φ to t. This completes the description of the residual graph.

Lemma 2.2. *For any $\delta > 0$ and a δ-feasible transport plan $\hat{\tau}, y(\cdot)$, the residual graph $\mathcal{G}_\delta$ has $O(n^2)$ vertices and $O(n^3)$ edges.*

While describing our algorithm, it is useful to have the definition of weighted distance for all the backward edges, including those incident on t. Therefore, we extend the definition of weighted distance to any edge (r_φ, t) as follows. Let b_φ denote the weighted nearest neighbor of r_φ in B, i.e., $b_\varphi := \min_{b \in B} d_y(r_\varphi, b)$. Define $d_y(r_\varphi, t) := d_y(r_\varphi, b_\varphi) + \delta$.

Compressing a semi-discrete transport plan. Given a semi-discrete transport plan τ that transports mass from B to A, we construct a transport plan $\hat{\tau}$ from B to A_δ as follows. For each pair $(r_\varphi, b) \in A_\delta \times B$, let $\hat{\tau}(r_\varphi, b) := \tau(\varphi, b)$, i.e., we assign the entire mass transported from b to φ to the pair (r_φ, b) (Figure 1 (right)). We refer to the transport plan $\hat{\tau}$ as the *compressed transport plan*.

Lemma 2.3. *For any δ-feasible semi-discrete transport plan $\tau, y(\cdot)$, the compressed transport plan $\hat{\tau}$ along with weights $y(\cdot)$ is also δ-feasible.*

Conversely, consider a transport plan $\hat{\tau}, y(\cdot)$ from B to A_δ. One can compute a semi-discrete transport plan τ that, given a pair $(r_\varphi, b) \in A_\delta \times B$, arbitrarily transports a mass of $\hat{\tau}(r_\varphi, b)$ from b to φ.

Lemma 2.4. *Any δ-feasible transport plan $\hat{\tau}, y(\cdot)$ from B to A_δ can be converted into a δ-feasible semi-discrete transport plan τ from B to A.*

We say that any compressed transport plan $\hat{\tau}$ is a *forest* if the edges transporting a positive mass in $\hat{\tau}$ do not create an undirected cycle.

Augmentation. Given the residual graph $\mathcal{G}_\delta$ for a δ-feasible transport plan $\hat{\tau}, y(\cdot)$, an *alternating path* (resp. *alternating cycle*) is a directed path (resp. directed cycle) in $\mathcal{G}_\delta$. Note that, in any directed path (resp. cycle) in the residual graph, the edges alternate between forward and backward edges. An *augmenting path* $P = \langle s, b_1, r_1, \ldots, r_k, t = b_{k+1} \rangle$ is a directed path from the source vertex s to the sink vertex t in the residual graph. By construction, the vertex b_1 and the vertex r_k are free vertices in the residual graph. One can *augment* $\hat{\tau}$ along P as follows. Define the bottleneck capacity of the augmenting path P as $\mathrm{bc}(P) := \min\{\mathrm{ex}(b_1), \mathrm{ex}(r_k), \min_{i \in [1, k-1]} \hat{\tau}(r_i, b_{i+1})\}$. To augment $\hat{\tau}$ along P, set $\hat{\tau}(r_i, b_i) \leftarrow \hat{\tau}(r_i, b_i) + \mathrm{bc}(P)$ for each forward edge $(b_i, r_i) \in P$ and $\hat{\tau}(r_i, b_{i+1}) \leftarrow \hat{\tau}(r_i, b_{i+1}) - \mathrm{bc}(P)$ for each backward edge $(r_i, b_{i+1}) \in P$.

Lemma 2.5. *The transport plan obtained after augmenting a δ-feasible transport plan $\hat{\tau}, y(\cdot)$ along any augmenting path P in the residual graph is δ-feasible.*

Consider the following straightforward way to augment a semi-discrete transport plan. Given a δ-feasible semi-discrete transport plan $\tau, y(\cdot)$, we can compute an augmenting path P in the residual graph and augment the compressed transport plan $\hat{\tau}$ along P. From Lemmas 2.3 and 2.5, $\hat{\tau}$ remains δ-feasible and from Lemma 2.4, the updated δ-feasible transport plan $\hat{\tau}$ can be converted to a δ-feasible semi-discrete transport plan as desired. To obtain a complete transport plan, one can iteratively apply this procedure until there are no augmenting paths in the residual graph; however, this may result in an unbounded number of iterations. To obtain an efficient algorithm, we iteratively compute a set of
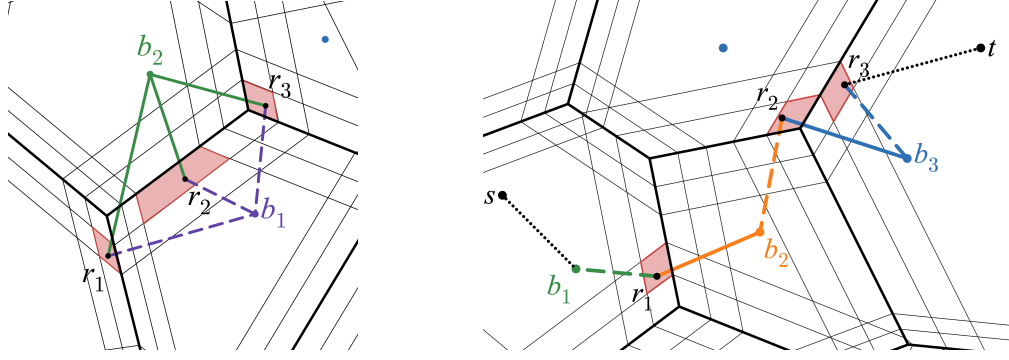
Figure 2: (left) Three admissible triples (b_1, r_1, b_2), (b_1, r_2, b_2), and (b_1, r_3, b_2), where solid (resp. dashed) lines show backward (resp. forward) edges, and (right) an admissible augmenting path.

special augmenting paths called the "admissible" augmenting paths and augment the transport plan along these paths. By doing so, we can reduce the number of iterations to $O(n)$.

Admissibility. Suppose $\hat{\tau}, y(\cdot)$ is a δ-feasible transport plan between μ and ν. For any pair of points $b_1, b_2 \in B$ and any region $\varphi \in \mathcal{X}_\delta$ such that $\hat{\tau}(r_\varphi, b_2) > 0$, the triple (b_1, r_φ, b_2) is *admissible* if $d_y(r_\varphi, b_1) < d_y(r_\varphi, b_2)^2$. See Figure 2 (left). Note that an admissible triple (b_1, r_φ, b_2) forms by a forward edge followed by a backward edge in the residual graph satisfying $d_y(r_\varphi, b_1) < d_y(r_\varphi, b_2)$. Intuitively, for any admissible triple $\langle b_1, r_\varphi, b_2 \rangle$, the mass of μ inside r_φ is transported from b_2 but b_1 is nearer to r_φ than b_2 (with respect to the weighted distances).

We extend the definition of admissibility to augmenting paths and alternating cycles as follows. Any augmenting path (resp. alternating cycle) $P = \langle b_1, r_1, b_2, \ldots, r_k, b_{k+1} \rangle$ is *admissible* if all triples (b_i, r_i, b_{i+1}), $i \in [1, k]$, are admissible. In Figure 2(right), b_2 transports mass to r_1, and b_1 is its weighted nearest neighbor. By augmenting along this admissible path, we increase $\hat{\tau}(r_1, b_1)$ and reduce $\hat{\tau}(r_1, b_2)$, thereby transporting more mass to r_1 from its weighted nearest neighbor.

3 Algorithm

In this section, we present our cost-scaling algorithm that uses the combinatorial framework from Section 2 to compute an ε-close semi-discrete OT plan. Classical discrete OT algorithms assign weights to points in $A \cup B$ and use them to identify a large set of augmenting paths. Their efficiency critically relies on the acyclicity of the "search" graph. In contrast, during the execution of our algorithm, a change in the weights of B creates a new weighted Voronoi diagram, which in turn changes A_δ, the discrete representation of A, and thus the vertex set of the residual graph. Since A_δ may change significantly in each iteration during the execution of our algorithm, we cannot maintain weights for them. This creates significant challenges as the algorithm searches for augmenting paths (See Section 3.2 for details). Furthermore, the updated residual graph may have cycles. We introduce additional steps in our algorithm to eliminate these cycles (See Section 3.4 for details). First, we present an overview of our algorithm and then present the details.

3.1 Overview

The algorithm runs for $O(\log \frac{\Delta}{\varepsilon})$ scales, where Δ is the diameter of $A \cup B$. In each scale δ, our algorithm maintains a transport plan $\hat{\tau}_\delta$ and a weight $y(b)$ for every point $b \in B$. Initially, set $\delta = \Delta^2$ and define $y(b) = 0$ for all $b \in B$. In each scale δ, execute the following steps.

1. *Initialization:* Set τ_δ to be an empty transport plan. Compute the residual graph $\mathcal{G}_\delta$ and the compressed transport plan $\hat{\tau}_\delta$ with respect to $\tau_\delta, y(\cdot)$.

2. *Iterations:* While $\hat{\tau}_\delta$ is not a complete transport plan:

---

[2]Discrete OT algorithms use the weights assigned to $A \cup B$ to define admissible edges. Since A_δ evolves during the execution of our algorithm, we cannot maintain weights for them, forcing us to define admissibility on a sequence of two consecutive edges (b_1, r_φ) and (r_φ, b_2) rather than per edge.

(i) Compute a set of admissible augmenting paths in the residual graph $\mathcal{G}_\delta$ and augment $\hat{\tau}_\delta$ along these paths using the SEARCHANDAUGMENT procedure described in Section 3.2. At the end of this step, there are no admissible augmenting paths in the residual graph.

(ii) Adjust the weights of all points of B that are reachable from the source by admissible paths by δ and recompute the set A_δ, the residual graph $\mathcal{G}_\delta$, and the compressed transport plan $\hat{\tau}_\delta$ using the INCREASEWEIGHTS procedure described in Section 3.3.

(iii) Update the compressed transport plan $\hat{\tau}_\delta$ and the residual graph using the ACYCLIFY procedure described in Section 3.4, so that the transport plan $\hat{\tau}_\delta$ is a forest and the residual graph does not have any admissible cycles.

3. *Scale Update:* Set $\delta \leftarrow \delta/2$.

After the execution of a scale $\delta \leq \varepsilon/2$, our algorithm terminates by returning a complete semi-discrete transport plan τ_δ obtained from the compressed complete transport plan $\hat{\tau}_\delta$ (using Lemma 2.4).

Invariants. As shown in Section 4 below, our algorithm iteratively updates the weights $y(\cdot)$ and the transport plan $\hat{\tau}_\delta$ while maintaining the following invariants:

(I1) The transport plan $\hat{\tau}_\delta, y(\cdot)$ is δ-feasible, and

(I2) At the start of each iteration, the transport plan $\hat{\tau}_\delta$ is a forest and there are no admissible cycles in the residual graphs.

Remark. The algorithm by Agarwal et al. [5] creates a discrete instance in each scale of the algorithm by computing the arrangement of the δ-, 2δ-, ..., and $(4n+1)\delta$-expanded Voronoi cells of each point $b \in B$. Instead of using such a fine partition to create a discrete instance with $O(n^5)$ edges, we work directly with the continuous space, maintain a much smaller residual graph with $O(n^3)$ edges, and use our semi-discrete combinatorial framework to find a transport plan.

3.2 SEARCHANDAUGMENT Procedure

The SEARCHANDAUGMENT procedure executes a partial DFS-style search to identify a set of admissible augmenting paths and augments the transport plan along these paths. The SEARCHAN-DAUGMENT procedure is somewhat similar in style to the blocking flow procedure in Dinic's max-flow algorithm [23] or the partial-DFS procedure in Gabow-Tarjan's algorithm [27], both of which rely on the property that there are no admissible cycles in the residual graph. Unlike these algorithms, in our case, if we augment along an arbitrary admissible augmenting path, we may create an admissible cycle. See Figure 3.

We overcome this challenge by carefully calibrating the search algorithm in two ways. First, we begin our search from the free regions instead of the free points of B. Thus, we reverse the direction of all the edges of the residual graph and begin our search from the sink t. Second, we explore all forward edges incident on a region in the increasing order of their weighted distance. This order of processing edges ensures that no admissible cycles are created and that there are no more admissible augmenting paths in the residual graph after the SEARCHANDAUGMENT procedure terminates. We provide the details below.

Let $\overleftarrow{\mathcal{G}_\delta}$ be the graph formed by reversing the direction of all the edges of $\mathcal{G}_\delta$. We conduct our search starting from the sink t in the graph $\overleftarrow{\mathcal{G}_\delta}$. Initially, mark all points of B and all backward edges as unvisited, define $U := B$ as the set of unvisited points of B, and $Q = \langle t = b_0 \rangle$ as the search path that the procedure grows. Execute the following steps until the search path Q becomes empty.

1. If $Q = \langle t = b_0, r_1, b_1, \ldots, r_i, b_i \rangle$ for some $i \geq 0$,

   (a) If there is an edge from b_i to s in $\overleftarrow{\mathcal{G}_\delta}$ (i.e., b_i is a free point), then $P = \langle s, b_i, r_i, \ldots, r_1, t \rangle$ is an augmenting path in $\mathcal{G}_\delta$. Augment $\hat{\tau}_\delta$ along P and set $Q = \langle t = b_0 \rangle$.

   (b) Assume there is not edge from b_i to s. If there is an unvisited edge (b_i, r) in $\overleftarrow{\mathcal{G}_\delta}$, then add $r = r_{i+1}$ to Q. Otherwise, mark b_i as visited and remove b_i from U and Q.
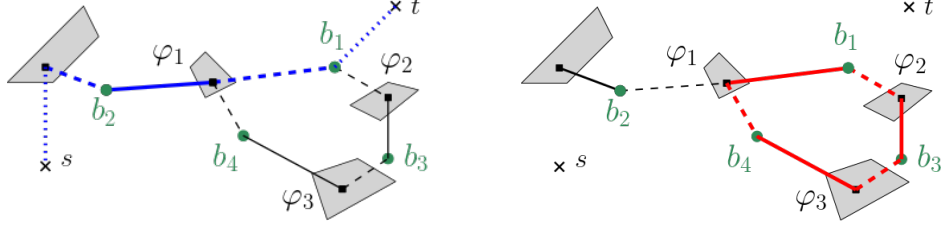
Figure 3: (left) An arbitrary admissible augmenting path $\langle s, b_1, r_{\varphi_1}, b_2 \rangle$ (blue path), and (right) an admissible cycle $\langle b_1, r_{\varphi_2}, b_3, r_{\varphi_3}, b_4, r_{\varphi_1} \rangle$ (red cycle) formed after augmentation.

2. If $Q = \langle t = b_0, r_1, b_1, \ldots, b_i, r_{i+1} \rangle$ for some $i \geq 0$, let $b := \arg\min_{b' \in U} d_y(r_{i+1}, b')$ denote the unvisited point with the minimum weighted distance to r_{i+1}[3].

   (a) If (b, r_{i+1}, b_i) is admissible, i.e., $d_y(r_{i+1}, b) < d_y(r_{i+1}, b_i)$, then add b as b_{i+1} to Q.
   (b) Otherwise, remove r_{i+1} from Q and mark the edge (b_i, r_{i+1}) as visited.

The algorithm terminates when the search path Q becomes empty, i.e., the procedure marked t as visited and removed it from Q. The following lemma shows the properties of the SEARCHANDAUG-MENT procedure.

Lemma 3.1. *Suppose invariants (I1) and (I2) hold at the start of the* SEARCHANDAUGMENT *procedure. Then, during the execution of the* SEARCHANDAUGMENT *procedure,*

   (S1) *the transport plan $\hat{\tau}_\delta, y(\cdot)$ remains δ-feasible,*
   (S2) *any point $b \in B$ (resp. backward edge (r, b)) marked as visited will not form an admissible augmenting path during the execution of the procedure, and*
   (S3) *there are no admissible cycles in the residual graph.*

3.3 INCREASEWEIGHTS Procedure

The INCREASEWEIGHTS procedure adjusts the weights of B leading to the formation of admissible augmenting paths. The INCREASEWEIGHTS procedure performs a DFS on the residual graph $\mathcal{G}_\delta$ to compute a set $\mathcal{K}$ of all vertices of B that are reachable from the source vertex s by admissible paths. It increases the weights of all points in $\mathcal{K}$ by δ (expand their Voronoi cells). Recall that the set A_δ, and thus $\mathcal{G}_\delta$ as well as the compressed transport plan $\hat{\tau}_\delta$ depend on the weights of B. The procedure then recomputes A_δ, $\mathcal{G}_\delta$, and $\hat{\tau}_\delta$ from τ_δ. See Appendix B.2 for complete details of the INCREASEWEIGHTS procedure. The following lemma states the properties of the INCREASEWEIGHTS procedure.

Lemma 3.2. *Suppose invariant (I1) holds at the start of the* INCREASEWEIGHTS *procedure. Then, during the execution of the* INCREASEWEIGHTS *procedure,*

   (W1) *the transport plan $\hat{\tau}_\delta, y(\cdot)$ remain δ-feasible,*
   (W2) *the weight of each free point $b \in B$ increases by δ, and*
   (W3) *the weight of each point $b \in B$ with free regions inside V_b^δ remains unchanged.*

3.4 ACYCLIFY Procedure

The change in the weights of B in the INCREASEWEIGHTS procedure requires us to recompute the residual graph and the compressed transport plan. This recomputation may potentially create a cycle in the transport plan or an admissible cycle in $\mathcal{G}_\delta$. The ACYCLIFY procedure eliminates such cycles and ensures that the invariant (I2) holds at the start of the next iteration. Converting the transport plan into a forest is critical for the efficiency of the SEARCHANDAUGMENT procedure while eliminating admissible cycles is essential for the correctness of the SEARCHANDAUGMENT procedure. The procedure executes the following steps. First, use the dynamic tree structure by Sleator and Tarjan [56] to make the transport plan $\hat{\tau}_\delta$ a forest. Then, execute a partial DFS search from

---

[3]To perform this step efficiently, in the construction of the residual graph, for each $r \in A_\delta$, we store the list of all neighbors of r sorted in increasing order of their weighted distance.

each unvisited point $b \in B$ similar to the one described in the SEARCHANDAUGMENT procedure to detect admissible cycles. Upon finding an admissible cycle, cancel the cycle right away, remove the vertices of the cycle from the search path, and continue the search. When all vertices are visited, no admissible cycles are remaining in the residual graph. Since canceling admissible cycles could have introduced new cycles in the transport plan, repeat the first step to update $\hat{\tau}_\delta$ and make it a forest. See Appendix B.3 for details. The following lemma shows the properties of the ACYCLIFY procedure.

Lemma 3.3. *Suppose invariant (I1) holds at the start of the* ACYCLIFY *procedure. Then, during the execution of the* ACYCLIFY *procedure,*

>   *(A1) the transport plan $\hat{\tau}_\delta, y(\cdot)$ remains δ-feasible, and*
>   *(A2) the transport plan $\hat{\tau}_\delta$ is a forest and there are no admissible cycles in the residual graph.*

This completes the description of our algorithm.

4   Algorithm Analysis

In this section, we first prove the correctness of our algorithm and then analyze its runtime.

Proof of invariants (I1) and (I2). For any scale δ, the initial transport plan $\hat{\tau}_\delta$ is empty. Therefore, $\hat{\tau}_\delta$ along with the weights $y(\cdot)$ is δ-feasible. By properties (S1), (W1), and (A1), the transport plan $\hat{\tau}_\delta, y(\cdot)$ remains δ-feasible in each iteration of our algorithm, and therefore, invariant (I1) holds. The invariant (I2) is a direct consequence of property (A2) in Lemma 3.3.

Proof of Correctness. From Invariant (I1), in each scale δ, our algorithm maintains a δ-feasible transport plan $\hat{\tau}_\delta, y(\cdot)$ during its execution. The while loop in Step 2 breaks when $\hat{\tau}_\delta$ is a complete transport plan. Therefore, $\hat{\tau}_\delta$ along with weights $y(\cdot)$ is δ-optimal. From Lemma 2.4, one can convert $\hat{\tau}_\delta$ into a δ-optimal semi-discrete transport plan τ_δ. Given that our algorithm terminates when $\delta \leq \varepsilon/2$, from Lemma 2.1, the transport plan returned by our algorithm is ε-close, as desired.

Efficiency of the algorithm. The SEARCHANDAUGMENT procedure runs a partial DFS on the residual graph with $O(n^3)$ edges. Upon finding an augmenting path P, the procedure augments the transport plan along P in $O(|P|)$ time. In Lemma C.1, we use invariant (I2) to show that the total length of all augmenting paths found by the procedure is $O(n^3)$. Hence, the running time of the SEARCHANDAUGMENT procedure is $O(n^3)$. The INCREASEWEIGHTS procedure stores a sorted list of neighbors for each region and executes a DFS procedure in the residual graph, which takes $O(n^3 \log n)$ time. After increasing the weight of a subset of points of B, the procedure recomputes the residual graph $\mathcal{G}_\delta$ and the transport plan $\hat{\tau}_\delta$, which can be done in $O(n^2(\Phi + n \log n))$ time. See Lemma C.2 in the appendix for details. In the ACYCLIFY procedure, converting a transport plan into a forest can be done using a dynamic tree data structure [56] in $O(n^3 \log n)$ time. To eliminate the admissible cycles, the ACYCLIFY procedure relies on a partial DFS that runs in $O(n^3)$ time. Therefore, the total time by the ACYCLIFY procedure is $O(n^3 \log n)$ (see Lemma C.3 for details). Combining the running times of all three procedures, each iteration of step 2 of our algorithm takes $O(n^2(\Phi + n \log n))$ time. In the following lemma, we show that in each scale, the total number of iterations of step 2 of our algorithm is at most $O(n)$.

Lemma 4.1. *For each scale δ, the total number of iterations of step 2 of our algorithm is $O(n)$.*

Proof Sketch. Let $\tau_{2\delta}, y_{2\delta}(\cdot)$ denote the 2δ-feasible semi-discrete transport plan computed by our algorithm for scale 2δ, and let $\tau_\delta, y_\delta(\cdot)$ denote a partial semi-discrete transport plan maintained during the execution of step 2 of our algorithm. Let $\mathcal{X}_{2\delta}$ (resp. $\mathcal{X}_\delta$) denote the partitioning of the set A with respect to weights $y_{2\delta}(\cdot)$ (resp. $y_\delta(\cdot)$). Let $\mathcal{Y}$ be the arrangement of all $3n$ cells used to construct $\mathcal{X}_{2\delta}$ with all $3n$ cells used to construct $\mathcal{X}_\delta$. Let $\hat{\tau}_{2\delta}$ (resp. $\hat{\tau}_\delta$) denote the compressed transport plan for $\tau_{2\delta}$ (resp. τ_δ) using the partitioning $\mathcal{Y}$. It is well-known that one can transform $\hat{\tau}_\delta$ to $\hat{\tau}_{2\delta}$ by augmenting $\hat{\tau}_\delta$ along a set of augmenting paths $\mathcal{P}$ on $\mathcal{Y} \times B$ and rearrange the transported mass along a set of cycles $\mathcal{C}$ on $\mathcal{Y} \times B$. Consider an augmenting path $P = \langle r_1, b_1, \ldots, r_k, b_k \rangle$ in $\mathcal{P}$. Since P is a simple path, it contains each point of B at most once and therefore, it has a length at most $2n - 1$. Additionally, for all $i \in [1, k]$, $\hat{\tau}_{2\delta}(r_i, b_i) > 0$ and for each $i \in [2, k]$, $\hat{\tau}_\delta(r_i, b_{i-1}) > 0$. For each edge (r_i, b_i), since $\hat{\tau}_{2\delta}(r_i, b_i) > 0$, by 2δ-feasibility of $\tau_{2\delta}, y_{2\delta}(\cdot)$ (condition (F1)), the point b_i is a 4δ-weighted nearest neighbor of r_i with respect to weights $y_{2\delta}(\cdot)$. Similarly, for each edge (b_{i-1}, r_i), since $\hat{\tau}_\delta(r_i, b_{i-1}) > 0$, by δ-feasibility of $\tau_\delta, y_\delta(\cdot)$ (condition (F1)), the point b_{i-1} is a 2δ-weighted nearest neighbor of r_i with respect to weights $y_\delta(\cdot)$. Since the length of P is at most $2n - 1$ and the

weight of b_1 does not change (by property (W3) in Lemma 3.3), we show that for the free point b_k in P, $y_\delta(b_k) - y_{2\delta}(b_k) \le 6n\delta$. Our algorithm increases the weight of b_k by δ in each iteration (property (W2) in Lemma 3.3), and therefore after $6n$ iterations, the point b_k cannot be free, i.e., after $O(n)$ iterations, all points of B are fully transported in $\hat{\tau}_\delta$. We provide the full proof in Section C.4.

Using Lemma 4.1, the total time spent in step 2 in each scale of our algorithm is $O(n^3(\Phi + n \log n))$. Since there are $O(\log \frac{\Delta}{\varepsilon})$ scales, the overall runtime is $O(n^3(\Phi + n \log n) \log \frac{\Delta}{\varepsilon})$, thereby proving Theorem 1.1.

5 Applications to the Discrete Optimal Transport Problem

In this section, we extend our combinatorial semi-discrete OT algorithm to the discrete OT problem and design a data structure that preprocesses and stores a large discrete distribution μ and efficiently computes an ε-close OT plan between μ and any query distribution ν in sub-linear time relative to the support size of μ. More precisely, given a discrete distribution μ with a (possibly large) support A of N points in $\mathbb{R}^2$, we design a data structure that, given a query discrete distribution ν with a support B of k points, computes an ε-close transport plan between μ and ν in $O(k^3(\sqrt{N} + k \log k) \log \frac{\Delta}{\varepsilon})$ time. Additionally, we show that if the support points have bounded integer coordinates and the masses are rational numbers, our data structure can efficiently compute an exact discrete OT plan.

At a high level, our data structure interprets the large discrete distribution μ as a continuous distribution and uses a simplex range-searching data structure as an oracle to compute the mass of μ inside a query triangle. In this way, for any query distribution ν, one can execute the steps of our combinatorial semi-discrete algorithm to compute an ε-close transport plan between μ and ν. More formally, our data structure preprocesses the distribution μ into a simplex range-searching data structure RS-DS, which takes $O(N)$ space, can be built in $O(N \log N)$ time, and returns the mass of μ inside a query triangle in $\Phi = O(\sqrt{N})$ time [59]. Given a query discrete distribution ν, one can use our algorithm from Section 3 in conjunction with the RS-DS to compute an ε-close transport plan between μ and ν in $O(k^3(\sqrt{N} + k \log k) \log \frac{\Delta}{\varepsilon})$ time leading to Theorem 1.2.

Consider the special case where the points in $A \cup B$ have positive integer coordinates bounded by λ, the mass of μ (resp. ν) on each point $a \in A$ (resp. $b \in B$) is a rational number of form $\frac{x_a}{T}$ (resp. $\frac{x_b}{T}$) for positive integers T and x_a (resp. x_b), and p is an even number. In this case, the p-Wasserstein cost of any transport plan between μ and ν is an integer multiple of $\frac{1}{T}$, and therefore, any $\frac{1}{2T}$-close transport plan between μ and ν would have a minimum cost. Thus, one can compute an exact discrete OT plan between μ and ν by setting $\varepsilon = \frac{1}{2T}$ in our data structure, which would have a query time of $O(k^3(\sqrt{N} + k \log k) \log(\lambda T))$, leading to the following corollary.

Corollary 5.1. *When the points in the supports of the distributions μ and ν have integer coordinates bounded by λ and the mass on each point is a rational number of form $\frac{x}{T}$, our data structure computes, for any even number p, an optimal solution for the p-Wasserstein problem between μ and ν in $O(k^3(\sqrt{N} + k \log k) \log(\lambda T))$ time.*

6 Conclusion

In this paper, we designed a novel combinatorial framework for the semi-discrete optimal transport problem and used it to compute an ε-close semi-discrete transport plan. We also used this framework to design a data structure that stores a discrete distribution μ over a large support of size N and can compute ε-close OT cost between μ and a query discrete distribution ν in a time that is sub-linear in N. We conclude with the following question: Can we use our combinatorial framework to compute an ε-close semi-discrete transport plan between high dimensional distributions in $\text{poly}(n, d, 1/\varepsilon)$ time.

References

[1] Pankaj K. Agarwal and R. Sharathkumar. Approximation algorithms for bipartite matching with metric and geometric costs. In *Proc. 46th Annual ACM Sympos. on Theory of Computing*, STOC '14, pages 555–564, 2014.

[2] Pankaj K Agarwal and Micha Sharir. Arrangements and their applications. In *Handbook of Computational Geometry*, pages 49–119. Elsevier, 2000.

[3] Pankaj K Agarwal, Kyle Fox, Debmalya Panigrahi, Kasturi R Varadarajan, and Allen Xiao. Faster algorithms for the geometric transportation problem. In *33rd Internat. Symposium on Computational Geometry*, 2017.

[4] Pankaj K Agarwal, Sharath Raghvendra, Pouyan Shirzadian, and Rachita Sowle. A higher precision algorithm for computing the 1-Wasserstein distance. In *Proc. 11th Internat. Conference on Learning Representations*, 2023.

[5] Pankaj K Agarwal, Sharath Raghvendra, Pouyan Shirzadian, and Keegan Yao. Fast and accurate approximations of the optimal transport in semi-discrete and discrete settings. In *Proc. 2024 Annual ACM-SIAM Sympos. on Discrete Algorithms*, pages 4514–4529. SIAM, 2024.

[6] Jason M Altschuler, Jonathan Niles-Weed, and Austin J Stromme. Asymptotics for semidiscrete entropic optimal transport. *SIAM Journal of Mathematical Analysis*, 54(2):1718–1741, 2022.

[7] Luca Ambrogioni, Umut Guclu, and Marcel van Gerven. Wasserstein variational gradient descent: From semi-discrete optimal transport to ensemble variational inference. *arXiv preprint arXiv:1811.02827*, 2018.

[8] Luca Ambrogioni, Umut Güçlü, and Marcel van Gerven. k-gans: Ensemble of generative models with semi-discrete optimal transport. *arXiv preprint arXiv:1907.04050*, 2019.

[9] Dongsheng An, Yang Guo, Na Lei, Zhongxuan Luo, Shing-Tung Yau, and Xianfeng Gu. AE-OT: A new generative model based on extended semi-discrete optimal transport. In *Proc. 32nd Internat. Conference on Learning Representations*, 2019.

[10] Dongsheng An, Yang Guo, Min Zhang, Xin Qi, Na Lei, and Xianfang Gu. AE-OT-GAN: Training gans from data specific latent distribution. In *Proc. 16th European Conference on Computer Vision*, pages 548–564. Springer, 2020.

[11] Franz Aurenhammer. Power diagrams: properties, algorithms and applications. *SIAM journal on computing*, 16(1):78–96, 1987.

[12] Franz Aurenhammer, Friedrich Hoffmann, and Boris Aronov. Minkowski-type theorems and least-squares clustering. *Algorithmica*, 20(1):61–76, 1998.

[13] Arturs Backurs, Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Scalable nearest neighbor search for optimal transport. In *Internat. Conference on Machine Learning*, pages 497–506. PMLR, 2020.

[14] Marc G Bellemare, Ivo Danihelka, Will Dabney, Shakir Mohamed, Balaji Lakshminarayanan, Stephan Hoyer, and Rémi Munos. The Cramer distance as a solution to biased Wasserstein gradients. *arXiv preprint arXiv:1705.10743*, 2017.

[15] Jean-David Benamou and Yann Brenier. A computational fluid mechanics solution to the Monge-Kantorovich mass transfer problem. *Numerische Mathematik*, 84(3):375–393, 2000.

[16] Bernard Bercu and Jérémie Bigot. Asymptotic distribution and convergence rates of stochastic algorithms for entropic optimal transportation between probability measures. *Annals of Statistics*, 49(2):968–987, 2021.

[17] Rick Chartrand, Brendt Wohlberg, Kevin Vixie, and Erik Bollt. A gradient descent solution to the Monge-Kantorovich problem. *Applied Math. Sci.*, 3(22):1071–1080, 2009.

[18] Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *Proc. 63rd IEEE Annual Sympos. Foundations of Computer Science*, pages 612–623. IEEE, 2022.

[19] Yucheng Chen, Matus Telgarsky, Chao Zhang, Bolton Bailey, Daniel Hsu, and Jian Peng. A gradual, semi-discrete approach to generative network training via explicit Wasserstein minimization. In *Internat. Conference on Machine Learning*, pages 1071–1080. PMLR, 2019.

[20] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems*, pages 2292–2300, 2013.

[21] Fernando De Goes, Katherine Breeden, Victor Ostromoukhov, and Mathieu Desbrun. Blue noise through optimal transport. *ACM Transactions of Graphics*, 31(6):1–11, 2012.

[22] Eustasio Del Barrio, Alberto González Sanz, and Jean-Michel Loubes. Central limit theorems for semi-discrete Wasserstein distances. *Bernoulli*, 30(1):554–580, 2024.

[23] Efim A Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. In *Soviet Math. Doklady*, volume 11, pages 1277–1280, 1970.

[24] Steven Fortune. Voronoi diagrams and Delaunay triangulations. *In Handbook of Discrete and Computational Geometry*, pages 225–265, 1995.

[25] Emily Fox and Jiashuai Lu. A deterministic near-linear time approximation scheme for geometric transportation. In *Proc. 64th IEEE Annual Sympos. on Foundations of Computer Science*, pages 1301–1315. IEEE, 2023.

[26] Kyle Fox and Jiashuai Lu. A near-linear time approximation scheme for geometric transportation with arbitrary supplies and spread. In *Proc. 36th Annual Sympos. on Computational Geometry*, pages 45:1–45:18, 2020.

[27] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal of Computing*, 18(5):1013–1036, October 1989. ISSN 0097-5397.

[28] Bruno Galerne, Arthur Leclaire, and Julien Rabin. Semi-discrete optimal transport in patch space for enriching Gaussian textures. In *Proc. 3rd Internat. Conference on Geometric Science of Information*, pages 100–108, 2017.

[29] Bruno Galerne, Arthur Leclaire, and Julien Rabin. A texture synthesis model based on semi-discrete optimal transport in patch space. *SIAM Journal of Imaging Sciences*, 11(4):2456–2493, 2018.

[30] Thomas O Gallouët and Quentin Mérigot. A Lagrangian scheme à la Brenier for the incompressible Euler equations. *Foundations of Computational Mathematics*, 18(4):835–865, 2018.

[31] Jing Gao, Peng Li, Asif Ali Laghari, Gautam Srivastava, Thippa Reddy Gadekallu, Sidra Abbas, and Jianing Zhang. Incomplete multiview clustering via semidiscrete optimal transport for multimedia data mining in IoT. *ACM Transactions of Multimedia Computing, Communications and Applications*, 20(6):1–20, 2024.

[32] Aude Genevay, Marco Cuturi, Gabriel Peyré, and Francis Bach. Stochastic optimization for large-scale optimal transport. *Advances in Neural Information Processing Systems*, 29, 2016.

[33] Antoine Houdard, Arthur Leclaire, Nicolas Papadakis, and Julien Rabin. A generative model for texture synthesis based on optimal transport between feature distributions. *Journal of Mathematical Imaging and Vision*, 65(1):4–28, 2023.

[34] Martin Huesmann and Karl-Theodor Sturm. Optimal transport from Lebesgue to poisson. *The Annals of Probability*, 41(4):2426–2478, 2013.

[35] Shayan Hundrieser, Thomas Staudt, and Axel Munk. Empirical optimal transport between different measures adapts to lower complexity. In *Annales de l'Institut Henri Poincare (B) Probabilites et statistiques*, volume 60, pages 824–846, 2024.

[36] Andrey Boris Khesin, Aleksandar Nikolov, and Dmitry Paramonov. Preconditioning for the geometric transportation problem. *arXiv preprint arXiv:1902.08384*, 2019.

[37] Jun Kitagawa. An iterative scheme for solving the optimal transportation problem. *Calculus of Variations and Partial Differential Equations*, 51(1):243–263, 2014.

[38] Jun Kitagawa, Quentin Mérigot, and Boris Thibert. Convergence of a Newton algorithm for semi-discrete optimal transport. *Journal of European Math. Society*, 21(9):2603–2651, 2019.

[39] Nathaniel Lahn, Deepika Mulchandani, and Sharath Raghvendra. A graph theoretic additive approximation of optimal transport. *Advances in Neural Information Processing Systems*, 32, 2019.

[40] Arthur Leclaire and Julien Rabin. A stochastic multi-layer algorithm for semi-discrete optimal transport with applications to texture synthesis and style transfer. *Journal of Mathematical Imaging and Vision*, 63(2):282–308, 2021.

[41] Bruno Lévy and Erica L Schwindt. Notions of optimal transport theory and how to implement them on a computer. *Computers & Graphics*, 72:135–148, 2018.

[42] Zezeng Li, Shenghao Li, Zhanpeng Wang, Na Lei, Zhongxuan Luo, and David Xianfeng Gu. DPM-OT: a new diffusion probabilistic model based on optimal transport. In *Proc. IEEE/CVF Internat. Conference on Computer Vision*, pages 22624–22633, 2023.

[43] Huidong Liu, GU Xianfeng, and Dimitris Samaras. A two-step computation of the exact GAN Wasserstein distance. In *Internat. Conference on Machine Learning*, pages 3159–3168, 2018.

[44] Quentin Mérigot. A multiscale approach to optimal transport. *Computer Graphics Forum*, 30 (5):1583–1592, 2011.

[45] Quentin Merigot and Boris Thibert. Optimal transport: discretization and algorithms. In *Handbook of Numerical Analysis*, volume 22, pages 133–212. Elsevier, 2021.

[46] Jocelyn Meyron. Initialization procedures for discrete and semi-discrete optimal transport. *Computer-Aided Design*, 115:13–22, 2019.

[47] Joseph SB Mitchell, David M Mount, and Christos H Papadimitriou. The discrete geodesic problem. *SIAM Journal of Computing*, 16(4):647–668, 1987.

[48] Vladimir I Oliker and Laird D Prussner. On the numerical solution of the equation $\frac{\partial^2 z}{\partial x^2} \frac{\partial^2 z}{\partial y^2} - \left(\frac{\partial^2 z}{\partial x \partial y} \right) = f$ and its discretizations, i. *Numerische Mathematik*, 54(3):271–293, 1989.

[49] James Orlin. A faster strongly polynomial minimum cost flow algorithm. In *Proc. 20th Annual ACM Sympos. Theory of Computing*, pages 377–387, 1988.

[50] Aram-Alexandre Pooladian, Vincent Divol, and Jonathan Niles-Weed. Minimax estimation of discontinuous optimal transport maps: The semi-discrete case. In *Internat. Conference on Machine Learning*, pages 28128–28150. PMLR, 2023.

[51] Hongxing Qin, Yi Chen, Jinlong He, and Baoquan Chen. Wasserstein blue noise sampling. *ACM Transactions of Graphics*, 36(5):1–13, 2017.

[52] Sharath Raghvendra and Pankaj K Agarwal. A near-linear time ε-approximation algorithm for geometric bipartite matching. *Journal of the ACM*, 67(3):1–19, 2020.

[53] Nathan A Seifert, Kirill Prozument, and Michael J Davis. Computational optimal transport for molecular spectra: The semi-discrete case. *The Journal of Chemical Physics*, 156(13), 2022.

[54] R Sharathkumar and Pankaj K Agarwal. Algorithms for the transportation problem in geometric settings. In *Proc. 23rd Annual ACM-SIAM Sympos. on Discrete Algorithms*, pages 306–317, 2012.

[55] Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Nicolas Papernot, Ross Anderson, and Yarin Gal. Ai models collapse when trained on recursively generated data. *Nature*, 631(8022): 755–759, 2024.

[56] Daniel D Sleator and Robert Endre Tarjan. A data structure for dynamic trees. In *Proc. 13th Annual ACM Sympos. on Theory of Computing*, pages 114–122, 1981.

[57] Kokichi Sugihara. Laguerre Voronoi diagram on the sphere. *Journal for Geometry and Graphics*, 6(1):69–81, 2002.

[58] Bahar Taşkesen, Soroosh Shafieezadeh-Abadeh, and Daniel Kuhn. Semi-discrete optimal transport: Hardness, regularization and numerical solution. *Mathematical Programming*, 199 (1):1033–1106, 2023.

[59] Emo Welzl. Partition trees for triangle counting and other range searching problems. In *Proc. 4th Annual Sympos. Computational Geometry*, pages 23–33, 1988.

A   Missing Proofs of Section 2

In this section, we present the missing proofs and details of our combinatorial framework described in Section 2.

Lemma 2.1. *Any δ-optimal transport plan $\tau, y(\cdot)$ between μ and ν is 2δ-close.*

Proof. Define the weighted cost of a transport plan τ' as $\phi_y(\tau') := \sum_{b \in B} \int_A d_y(a,b) \tau'(a,b) \, da$. For any complete transport plan τ',

$$\phi_y(\tau') = \sum_{b \in B} \int_A d_y(a,b) \tau'(a,b) \, da = \sum_{b \in B} \int_A \big(d(a,b) - y(b)\big) \tau'(a,b) \, da = \phi(\tau') - \sum_{b \in B} y(b)\nu(b). \tag{2}$$

For any point $a \in A$, let b_a denote the weighted nearest neighbor of a, i.e., $b_a := \arg\min_{b \in B} d_y(a,b)$. By property (F1) in the definition of δ-feasibility,

$$\phi_y(\tau) = \sum_{b \in B} \int_A d_y(a,b)\tau(a,b) \, da \le \sum_{b \in B} \int_A \big(d_y(a,b_a) + 2\delta\big)\tau(a,b) \, da$$

$$= \sum_{b \in B} \int_A d_y(a,b_a)\tau(a,b) \, da + 2\delta. \tag{3}$$

Let τ^* denote an optimal transport plan from μ to ν. Then,

$$\phi_y(\tau^*) = \sum_{b \in B} \int_A d_y(a,b)\tau^*(a,b) \, da \ge \sum_{b \in B} \int_A d_y(a,b_a)\tau^*(a,b) \, da. \tag{4}$$

Combining Equations (3) and (4) and plugging τ and τ^* in Equation (2),

$$\phi(\tau) = \phi_y(\tau) + \sum_{b \in B} y(b)\nu(b) \le \left[\sum_{b \in B} \int_A d_y(a,b_a)\tau(a,b) \, da + 2\delta \right] + \sum_{b \in B} y(b)\nu(b)$$

$$\le \phi_y(\tau^*) + 2\delta + \sum_{b \in B} y(b)\nu(b) = \phi(\tau^*) + 2\delta.$$

Therefore, $\phi(\tau) \le \phi(\tau^*) + 2\delta$ and τ is 2δ-close. $\qquad\square$

Residual graph.   Next, we show that for any $\delta > 0$ and any δ-feasible transport plan $\hat{\tau}, y(\cdot)$, the residual graph $\mathcal{G}_\delta$ has $O(n^2)$ vertices and $O(n^3)$ edges, leading to Lemma 2.2. To do so, we show below that the partitioning $\mathcal{X}_\delta$ consists of $O(n^2)$ regions. We then conclude that the number of vertices of $\mathcal{G}_\delta$ is $O(n^2)$. Furthermore, since the residual graph is a bipartite graph between set B of size n and set A_δ of size $O(n^2)$, the number of edges of $\mathcal{G}_\delta$ would be at most $O(n^3)$.

Recall that the partitioning $\mathcal{X}_\delta$ is constructed as the arrangement of all Voronoi cells, δ-expansions, and 2δ-expansions of the Voronoi cells of all points in B. Let $\mathcal{V}$ denote the set of vertices of this arrangement. Since the arrangement is a planar graph, the number of faces (i.e., regions) in $\mathcal{X}_\delta$ is $O(|\mathcal{V}|)$. Therefore, to show that $\mathcal{X}_\delta$ has $O(n^2)$ regions, we show that the number of vertices of this arrangement is $O(n^2)$.

For each point $b \in B$, let V_b^0 (resp. V_b^δ, $V_b^{2\delta}$) denote the Voronoi cell (resp. δ-expanded Voronoi cell, 2δ-expanded Voronoi cell) of b, and let y_b^0 (resp. $y_b^\delta, y_b^{2\delta}$) denote the weights of B used to compute the cell. Note that each Voronoi cell V_b^i for each $b \in B$ and $i \in \{0, \delta, 2\delta\}$ has at most n vertices. Hence, the total number of Voronoi vertices in the arrangement $\mathcal{X}_\delta$ is $O(n^2)$. Next, we count the number of intersection points of these Voronoi cells. For each pair of points $b_1, b_2 \in B$, consider the pair of cells $V_{b_1}^i$ and $V_{b_2}^j$, for some $i, j \in \{0, \delta, 2\delta\}$. We show that $V_{b_1}^i$ and $V_{b_2}^j$ intersect each other in at most two points.

Define the *weighted bisector* of two points b and b' with respect to weights $y(\cdot)$ as the locus of points that have the same weighted distance to b and b', i.e., all points $x \in \mathbb{R}^2$ such that $d_y(x,b) = d_y(x,b')$. Note that under the squared Euclidean distances, the weighted bisector of two points is a straight line. Let v denote an intersection point of $V_{b_1}^i$ and $V_{b_2}^j$, and suppose v lies on the segment of $V_{b_1}^i$
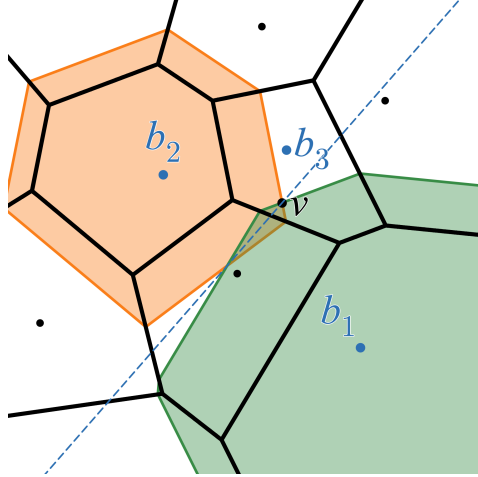
Figure 4: For two cells $V_{b_1}^i$ (green cell) and $V_{b_2}^j$ (orange cell), any intersection point v lies on their weighted bisector.

corresponding to the weighted bisector of b_1 and a point $b_3 \in B$ (See Figure 4). Note that if $b_3 = b_2$, then $V_{b_1}^i$ and $V_{b_2}^j$ share a segment containing v, which means that

$$\mathrm{d}_y(v, b_1) - i = \mathrm{d}_{y_{b_1}^i}(v, b_1) = \mathrm{d}_{y_{b_1}^i}(v, b_2) = \mathrm{d}_y(v, b_2),$$

and also

$$\mathrm{d}_y(v, b_2) - j = \mathrm{d}_{y_{b_2}^j}(v, b_2) = \mathrm{d}_{y_{b_2}^j}(v, b_1) = \mathrm{d}_y(v, b_1).$$

Therefore, in this case, $i = j = 0$, and the two endpoints of the shared segment is counted toward the number of Voronoi vertices of the arrangement. Hence, we assume $b_3 \neq b_2$. In this case,

$$\mathrm{d}_{y_{b_1}^i}(v, b_1) = \mathrm{d}_{y_{b_1}^i}(v, b_3) = \min_{b \in B} \mathrm{d}_{y_{b_1}^i}(v, b). \tag{5}$$

By the construction of the weight function $y_{b_1}^i(\cdot)$ in Equation (1), for any point $b \in B \setminus \{b_1, b_2\}$, $y_{b_1}^i(b) = y_{b_2}^j(b) = y(b)$; thus, using Equation (5),

$$\mathrm{d}_{y_{b_2}^j}(v, b_3) = \mathrm{d}_{y_{b_1}^i}(v, b_3) = \min_{b \in B \setminus \{b_1, b_2\}} \mathrm{d}_{y_{b_1}^i}(v, b) = \min_{b \in B \setminus \{b_1, b_2\}} \mathrm{d}_{y_{b_2}^j}(v, b). \tag{6}$$

Using Equation (6), the point v also lies in the segment of $V_{b_2}^j$ that corresponds to the weighted bisector of b_2 and b_3 with respect to weights $y_{b_2}^j$. Thus,

$$\mathrm{d}_{y_{b_1}^i}(v, b_1) = \mathrm{d}_{y_{b_1}^i}(v, b_3) = \mathrm{d}_{y_{b_2}^j}(v, b_3) = \mathrm{d}_{y_{b_2}^j}(v, b_2).$$

In other words, the point v satisfies

$$\mathrm{d}(v, b_1) - (y(b_1) + i) = \mathrm{d}(v, b_2) - (y(b_2) + j),$$

i.e., the point v lies on the weighted bisector of b_1 and b_2 (the blue dashed line in Figure 4). Since the weighted bisector is a straight line, it intersects the convex polygon $V_{b_1}^i$ in at most 2 points; hence, the two Voronoi cells $V_{b_1}^i$ and $V_{b_2}^j$ intersect each other in at most 2 points. Since the are $O(n^2)$ pairs of such Voronoi cells, each intersecting each other in at most 2 points, the total number of intersection points is at most $O(n^2)$, as claimed.

Lemma 2.2. *For any $\delta > 0$ and a δ-feasible transport plan $\hat{\tau}, y(\cdot)$, the residual graph $\mathcal{G}_\delta$ has $O(n^2)$ vertices and $O(n^3)$ edges.*

Next, we extend our bounds for the number of vertices and edges of the residual graph in Lemma 2.2 to any dimension $d \geq 3$ and show that for any pair of d-dimensional distributions μ and ν and any δ-feasible transport plan $\hat{\tau}, y(\cdot)$, the residual graph $\mathcal{G}_\delta$ has $O(n^d)$ vertices and $O(n^{d+1})$ edges. We prove our bounds by showing that the number of vertices of the arrangement $\mathcal{X}_\delta$ is $O(n^d)$. Using this

bound, we get that the number of regions in the arrangement is upper-bounded by $O(n^d)$; therefore, $|A_\delta| = O(n^d)$ and the residual graph has $O(n^d)$ vertices. Furthermore, since $\mathcal{G}_\delta$ is a bipartite graph between set B with n points and set A_δ with $O(n^d)$ points, the number of edges would be bounded by $O(n^{d+1})$.

For each point $b \in B$ and each $i \in \{0, \delta, 2\delta\}$, the Voronoi cell V_b^i has $O(n^{\lceil d/2 \rceil})$ vertices; since $3n$ such cells are used in the construction of the arrangement, the total number of Voronoi vertices in the arrangement is $O(n^{\lceil d/2 \rceil + 1})$. Next, we bound the number of intersection points of these Voronoi cells. For each d-tuple $(V_{b_1}^{i_1}, V_{b_2}^{i_2}, \ldots, V_{b_d}^{i_d})$ of Voronoi cells, for d distinct points $b_1, \ldots, b_d$ and d values $i_1, \ldots, i_d \in \{0, \delta, 2\delta\}$, let v denote a point in the intersection of these cells. Similar to our analysis for 2-dimensional distributions, we show that v lies on the weighted bisector of $b_1, \ldots, b_d$, where the weight of each point b_j is $y(b_j) + i_j$. Since this weighted bisector is a straight line, and since a straight line intersects a convex polytope in at most 2 points, each d-tuple of Voronoi cells intersect in at most 2 points, where the number of such d-tuples are $O(n^d)^4$. Therefore, we conclude that the number of intersection points is $O(n^d)$, as desired.

Let v be an intersection point of the d Voronoi cells $V_{b_1}^{i_1}, V_{b_2}^{i_2}, \ldots,$ and $V_{b_d}^{i_d}$. Suppose v lies on the $(d-1)$-dimensional hyperplane of $V_{b_1}^{i_1}$ that is shared between b_1 and a point $b_{d+1} \in B$. In this case,

$$d_{y_{b_1}^{i_1}}(v, b_1) = d_{y_{b_1}^{i_1}}(v, b_{d+1}) = \min_{b \in B} d_{y_{b_1}^{i_1}}(v, b). \tag{7}$$

We claim that for any $k \in \{1, \ldots, d\}$, $y_{b_k}^{i_k}(b_{d+1}) = y(b_{d+1})$. Consider the following two cases:

- If $b_{d+1} = b_j$ for some $j \in \{2, \ldots, d\}$, then

$$d_y(v, b_1) - i_1 = d_{y_{b_1}^{i_1}}(v, b_1) = d_{y_{b_1}^{i_1}}(v, b_j) = d_y(v, b_j),$$

  and also

$$d_y(v, b_j) - i_j = d_{y_{b_j}^{i_j}}(v, b_j) = d_{y_{b_j}^{i_j}}(v, b_1) = d_y(v, b_1).$$

  Therefore, $i_1 = i_j = 0$ and for any $k \in \{1, \ldots, d\}$, $y_{b_k}^{i_k}(b_{d+1}) = y(b_{d+1})$.

- Otherwise, $b_{d+1} \notin \{b_1, \ldots, b_d\}$ and by the construction of $y_{b_k}^{i_k}(\cdot)$, for any $k \in \{1, \ldots, d\}$, $y_{b_k}^{i_k}(b_{d+1}) = y(b_{d+1})$.

Therefore, from Equation (7),

$$d_y(v, b_{d+1}) = d_{y_{b_1}^{i_1}}(v, b_{d+1}) = \min_{b \in B} d_{y_{b_1}^{i_1}}(v, b) = \min_{b \in B} d_y(v, b), \tag{8}$$

and for any $k \in \{1, \ldots, d\}$, since v lies on a $(d-1)$-dimensional hyperplane of $V_{b_k}^{i_k}$, it has to lie on the hyperplane of $V_{b_k}^{i_k}$ that is shared between b_k and b_{d+1}; therefore, $d(v, b_{d+1}) - y(b_{d+1}) = d(v, b_k) - (y(b_k) + i_k)$. Thus,

$$d(v, b_1) - (y(b_1) + i_1) = d(v, b_2) - (y(b_2) + i_2) = \ldots = d(v, b_d) - (y(b_d) + i_d). \tag{9}$$

From Equation (9), the point v lies on the weighted bisector of $b_1, \ldots, b_d$ with weights $y(b_1) + i_1, \ldots, y(b_d) + i_d$, and the d-tuple $(V_{b_1}^{i_1}, V_{b_2}^{i_2}, \ldots, V_{b_d}^{i_d})$ intersect each other in at most 2 points.

Lemma A.1. *For any dimension $d \geq 2$, a parameter $\delta > 0$, and a δ-feasible transport plan $\hat{\tau}, y(\cdot)$, the residual graph $\mathcal{G}_\delta$ has $O(n^d)$ vertices and $O(n^{d+1})$ edges.*

Compressing the transport plan. Recall that for any semi-discrete transport plan τ from B to A, a set of weights $y(\cdot)$, and a parameter $\delta > 0$, the compressed transport plan $\hat{\tau}$ is a discrete transport plan from B to A_δ. Note that the definition of δ-feasibility (and more precisely, condition (F1)) is applicable for the discrete OT as well; the discrete transport plan $\hat{\tau}$ along with weights $y(\cdot)$ for B is δ-feasible if $\hat{\tau}$ transports the mass of each point $b \in B$ to points of A_δ within its 2δ-expanded Voronoi cell. Any complete transport plan $\hat{\tau}$ from B to A_δ that is δ-feasible along with weights $y(\cdot)$ is called δ-optimal.

---

[4] Here, the constant is $O(n^d)$ hides a factor of 3^d.

Lemma A.2. *Any δ-optimal compressed transport plan $\hat{\tau}, y(\cdot)$ from B to A_δ is 2δ-close.*

Proof. For any transport plan τ' from B to A_δ define $\cent_y(\tau') := \sum_{(r,b)\in A_\delta \times B} \mathrm{d}_y(r,b)\tau'(r,b)$. Then,

$$\cent_y(\tau') = \sum_{(r,b)\in A_\delta \times B} \mathrm{d}_y(r,b)\tau'(r,b) = \sum_{(r,b)\in A_\delta \times B} \big(\mathrm{d}(r,b) - y(b)\big)\tau'(r,b) = \cent(\tau') - \sum_{b\in B} y(b)\nu(b).$$

$$(10)$$

For any point $r \in A_\delta$, let b_r denote the weighted nearest neighbor of r, i.e., $b_r := \arg\min_{b\in B} \mathrm{d}_y(r,b)$. By the definition of δ-feasibility, for any pair (r,b) with $\hat{\tau}(r,b) > 0$, we have $\mathrm{d}_y(r,b) \le \mathrm{d}_y(r,b_r)+2\delta$. Therefore,

$$\cent_y(\hat{\tau}) = \sum_{(r,b)\in A_\delta \times B} \mathrm{d}_y(r,b)\hat{\tau}(r,b) \le \sum_{(r,b)\in A_\delta \times B} \big(\mathrm{d}_y(r,b_r) + 2\delta\big)\hat{\tau}(r,b)$$

$$= \sum_{(r,b)\in A_\delta \times B} \mathrm{d}_y(r,b_r)\hat{\tau}(a,b) + 2\delta. \tag{11}$$

Let τ^* denote an optimal transport plan from B to A_δ. Then,

$$\cent_y(\tau^*) = \sum_{(r,b)\in A_\delta \times B} \mathrm{d}_y(r,b)\tau^*(r,b) \ge \sum_{(r,b)\in A_\delta \times B} \mathrm{d}_y(r,b_r)\tau^*(r,b). \tag{12}$$

Combining Equations (11) and (12) and plugging $\hat{\tau}$ and τ^* in Equation (10),

$$\cent(\hat{\tau}) = \cent_y(\hat{\tau}) + \sum_{b\in B} y(b)\nu(b) \le \left[\sum_{(r,b)\in A_\delta \times B} \mathrm{d}_y(r,b_r)\hat{\tau}(r,b) + 2\delta \right] + \sum_{b\in B} y(b)\nu(b)$$

$$\le \cent_y(\tau^*) + 2\delta + \sum_{b\in B} y(b)\nu(b) = \cent(\tau^*) + 2\delta.$$

Therefore, $\cent(\hat{\tau}) \le \cent(\tau^*) + 2\delta$ and $\hat{\tau}$ is 2δ-close. $\qquad\square$

The following observation, which is straightforward from the construction of the partitioning $\mathcal{X}_\delta$, helps in proving Lemmas 2.3 and 2.4.

Observation A.3. *For any region $\varphi \in \mathcal{X}_\delta$ and any point $b \in B$, the region φ either completely lies inside $V_b^{2\delta}$ or it is completely outside $V_b^{2\delta}$.*

Lemma 2.3. *For any δ-feasible semi-discrete transport plan $\tau, y(\cdot)$, the compressed transport plan $\hat{\tau}$ along with weights $y(\cdot)$ is also δ-feasible.*

Proof. Consider any point $b \in B$ and any region $\varphi \in \mathcal{X}_\delta$ such that $\tau(\varphi, b) > 0$, i.e., in the compressed graph, $\hat{\tau}(r_\varphi, b) > 0$. To prove this lemma, we show that $r_\varphi \in V_b^{2\delta}$. Since $\tau(\varphi, b) > 0$, there exists a point $a \in \varphi$ such that $\tau(a,b) > 0$, and by the δ-feasibility of $\tau, y(\cdot)$, we have $a \in V_b^{2\delta}$; therefore, using Observation A.3, the region φ has to completely lie inside $V_b^{2\delta}$, and the representative point r_φ, which is a point inside φ also lies inside $V_b^{2\delta}$. Hence, $\hat{\tau}, y(\cdot)$ is δ-feasible. $\qquad\square$

Lemma 2.4. *Any δ-feasible transport plan $\hat{\tau}, y(\cdot)$ from B to A_δ can be converted into a δ-feasible semi-discrete transport plan τ from B to A.*

Proof. Consider any transport plan τ from B to A such that for any point $b \in B$ and any region $\varphi \in \mathcal{X}_\delta$, transports a mass of $\hat{\tau}(r_\varphi, b)$ from b to the mass of μ inside the region φ. One such construction is to assign $\tau(a,b) = \frac{\hat{\tau}(r_\varphi, b)}{\mu(\varphi)}\mu(a)$ for each point $a \in \varphi$. We next show that the transport plan τ is δ-feasible.

Consider any point $b \in B$ and any point $a \in A$ such that $\tau(a,b) > 0$. We prove this lemma by showing that $a \in V_b^{2\delta}$. Suppose $\varphi \in \mathcal{X}_\delta$ is the region containing the point a. Since $\tau(a,b) > 0$, we should have $\hat{\tau}(r_\varphi, b) > 0$, and by the δ-feasibility of $\hat{\tau}, y(\cdot)$, we have $r_\varphi \in V_b^{2\delta}$. By Observation A.3, the whole region φ has to lies inside $V_b^{2\delta}$ and therefore, the point a also lies inside $V_b^{2\delta}$. Hence, $\tau, y(\cdot)$ is δ-feasible. $\qquad\square$

Augmentation. In this section, we first show that upon augmenting a transport plan $\hat{\tau}$ along an augmenting path P, the transport plan remains valid. We then show that upon augmentation, either a backward edge gets removed from the transport plan or a free point will become fully transported. We finally prove Lemma 2.5.

For an augmenting path $P = \langle s, b_1, r_1, \ldots, r_k, t \rangle$ from a free point b_1 to a free point r_k, recall that the bottleneck capacity of P is defined as

$$\mathrm{bc}(P) := \min\{\mathrm{ex}(b_1), \mathrm{ex}(r_k), \min_{i \in [1, k-1]} \hat{\tau}(r_i, b_{i+1})\},$$

where $\mathrm{ex}(b_1)$ (resp. $\mathrm{ex}(r_k)$) denotes the excess mass of b_1 (resp. r_k). To show that the transport plan after augmentation is a valid one, we show that no point v in $\mathcal{G}_\delta$ transports more mass than the mass at v and that each edge transports a non-negative amount of mass.

In the augmentation process, for any $i \in \{1, \ldots, k-1\}$, we decrease $\hat{\tau}(r_i, b_{i+1})$ by $\mathrm{bc}(P)$, where by definition, $\mathrm{bc}(P) \le \hat{\tau}(r_i, b_{i+1})$; hence, $\hat{\tau}(r_i, b_{i+1}) \ge 0$ after augmentation. Furthermore, for any forward edge (b_i, r_i), $i \in \{1, \ldots, k\}$, we increases $\hat{\tau}(r_i, b_i)$ by $\mathrm{bc}(P)$, and $\hat{\tau}(r_i, b_i)$ remains non-negative.

For any $i \in \{2, \ldots, k\}$, we increase $\hat{\tau}(b_i, r_i)$ (resp. decrease $\hat{\tau}(r_{i-1}, b_i)$) by $\mathrm{bc}(P)$; hence, the total amount of mass transported from b_i remains unchanged. Similarly, for each $i \in \{1, \ldots, k-1\}$, we increase $\hat{\tau}(b_i, r_i)$ (resp. decrease $\hat{\tau}(r_i, b_{i+1})$) by $\mathrm{bc}(P)$ and the total amount of mass transported into r_i remains unchanged. For the endpoint b_1 (resp. r_k), we only increase the amount of mass transported from b_1 (resp. into r_k) by $\mathrm{bc}(P)$, where by definition, $\mathrm{bc}(P) \le \mathrm{ex}(b_1)$ (resp. $\mathrm{bc}(P) \le \mathrm{ex}(r_k)$). Therefore, the total mass transport from b_1 (resp. into r_k) after augmentation is at most $\nu(b)$ (resp. μ_{r_k}), as desired.

Note that if a backward edge $(r, b) \in P$ determines the bottleneck capacity of the augmenting path P, then $\hat{\tau}(r, b) = 0$ after augmentation and the backward edge is removed from the residual graph. Otherwise, if the endpoint b_1 (reps. r_k) determines the bottleneck capacity of P, then b_1 (resp. r_k) will be fully transported after augmentation.

Lemma 2.5. *The transport plan obtained after augmenting a δ-feasible transport plan $\hat{\tau}, y(\cdot)$ along any augmenting path P in the residual graph is δ-feasible.*

Proof. Let $P = \langle s, b_1, r_1, \ldots, b_k, r_k, t \rangle$ denote an augmenting path in the residual graph. When augmenting the transport plan along P, we increase the mass transportation on forward edges (b_i, r_i) for each $i \in [1, k]$ and decrease the mass transportation on the backward edges (r_i, b_{i+1}) for each $i \in [1, k-1]$. Therefore, any pair (r, b) that transports mass after augmentation but was not transporting mass before augmentation has to be a forward edge of P. Since we only add forward edges from the point b to the representative points in $V_b^{2\delta} \cap A_\delta$, the edge (r, b) satisfies the δ-feasibility condition (F1), and $\hat{\tau}, y(\cdot)$ remains feasible after augmentation. $\square$

B   Missing Details of Section 3

In this section, we provide the missing details of the implementation of the INCREASEWEIGHTS and ACYCLIFY procedures and also prove the properties of the three procedures.

B.1   Missing Proofs of the SEARCHANDAUGMENT Procedure

Lemma 3.1. *Suppose invariants (I1) and (I2) hold at the start of the* SEARCHANDAUGMENT *procedure. Then, during the execution of the* SEARCHANDAUGMENT *procedure,*

- (S1) *the transport plan $\hat{\tau}_\delta, y(\cdot)$ remains δ-feasible,*
- (S2) *any point $b \in B$ (resp. backward edge (r, b)) marked as visited will not form an admissible augmenting path during the execution of the procedure, and*
- (S3) *there are no admissible cycles in the residual graph.*

Proof. We prove the properties separately in the following.

Property (S1). By the construction of the search path, any augmenting path computed by the SEARCHANDAUGMENT procedure is an admissible augmenting path. From Lemma 2.5, augmenting
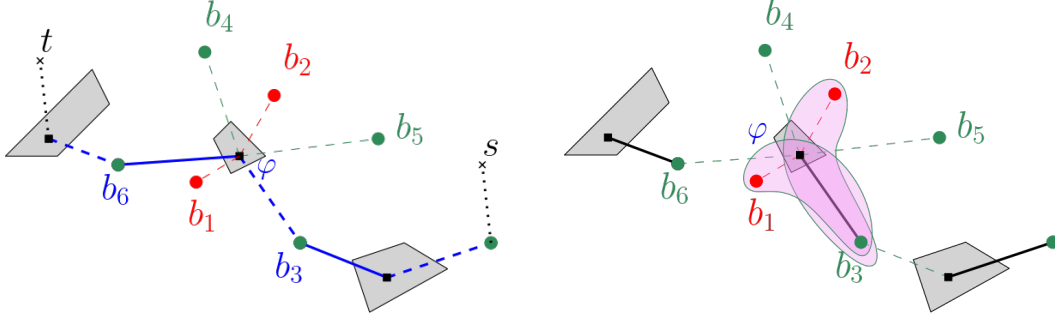
19

Figure 5: (left) An augmenting path found by the SEARCHANDAUGMENT procedure (blue path), and (right) two admissible triples (highlighted in pink) formed after augmentation.

$\hat{\tau}_\delta$ along an admissible augmenting path does not violate the δ-feasibility condition; hence, the transport plan $\hat{\tau}_\delta, y(\cdot)$ is a δ-feasible transport plan during the execution of the SEARCHANDAUGMENT procedure and (S1) holds.

Next, we present an overview of a new property of the SEARCHANDAUGMENT procedure, which we formally state and prove in Lemma B.1 below. Using that, we first prove (S3) and then prove (S2).

In Figure 5 (left), suppose the blue edges show an admissible augmenting path found by the SEARCHANDAUGMENT procedure, and suppose the green (resp. red) points show the unvisited (resp. visited) points of B. For the region φ, suppose $\langle b_1, b_2, \ldots, b_6 \rangle$ denote the set of neighbors of r_φ in $\mathcal{G}_\delta$, sorted in increasing order of their weighted distance to r_φ. When the partial DFS procedure processes r_φ, the two weighted nearest neighbors of φ (i.e., b_1 and b_2) are already marked as visited, leading the procedure to add b_3 to the search path. After augmentation (Figure 5 (right)), for the newly created backward edge (r_φ, b_3), the only admissible triples containing (r_φ, b_3) are (b_1, r_φ, b_3) and (b_2, r_φ, b_3) (the triples highlighted in pink), where b_3 is unvisited and both b_1 and b_2 are visited. More formally, as shown in Lemma B.1, for an augmenting path P found by the procedure, assuming that (S3) holds before augmentation along P, for any newly created admissible triples (b, r_φ, b') after augmenting the transport plan along P, the point b (resp. b') is marked as visited (resp. unvisited).

Property (S3). We begin by presenting an overview of our proof. Consider any augmenting path P found by the SEARCHANDAUGMENT procedure. Assuming that (S3) holds before augmentation along P, all vertices that are reachable from a visited point b by an admissible path in the procedure are also visited, since those points were also added to the search path, did not lead to an admissible augmenting path, marked as visited and removed from the search path. Hence, all points having an admissible path to the visited points (i.e., all points that are reachable from the visited points in our backward DFS) are also visited. Therefore, there are no admissible paths from an unvisited point to a visited point in the residual graph. After augmenting along P, by Lemma B.1, for any newly formed admissible triple (b, r_φ, b'), the point b (resp. b') is visited (resp. unvisited), and since there are no admissible paths from any unvisited point to any visited point, the newly formed admissible triple (b, r_φ, b') does not form a cycle of admissible triples. Hence, (S3) holds after augmentation as well. We provide the details of the proof below.

Let $P^1, \ldots, P^k$ denote the sequence of augmenting paths computed by the SEARCHANDAUGMENT procedure, and let $\hat{\tau}_\delta^0, \hat{\tau}_\delta^1, \ldots, \hat{\tau}_\delta^k$ denote the sequence of transport plans computed by the procedure, i.e., $\hat{\tau}_\delta^0$ is the transport plan computed in the previous iteration, and for each $i \in [1, k]$, $\hat{\tau}_\delta^i$ is obtained by augmenting $\hat{\tau}_\delta^{i-1}$ along P^i. Let $\mathcal{G}^i$ denote the residual graph corresponding to $\hat{\tau}_\delta^i$ for each $i \in [0, k]$. Let V^i (resp. U^i) denote the set of visited (resp. unvisited) points in B when the procedure augments $\hat{\tau}_\delta^{i-1}$ along P^i.

Initially, from invariant (I2), there are no cycles of admissible triples in the residual graph, and (S3) holds for $\mathcal{G}^0$. For any $i \in [1, k]$, assuming $\mathcal{G}^0, \ldots, \mathcal{G}^{i-1}$ satisfies (S3), we show that $\mathcal{G}^i$ also satisfies (S3). Suppose (b, r, b') is any admissible triple in $\mathcal{G}^i$ formed after augmenting $\hat{\tau}_\delta^{i-1}$ along P^i, i.e., the triple (b, r, b') is admissible in $\mathcal{G}^i$ but not in $\mathcal{G}^{i-1}$. We show that the triple (b, r, b') does not participate in any admissible cycles; hence, using property (S3) on $\mathcal{G}^{i-1}$, there are no admissible cycles in $\mathcal{G}^i$ and property (S3) holds for $\mathcal{G}^i$ as well.

Since (S3) holds for $\mathcal{G}^0, \ldots, \mathcal{G}^{i-1}$, for any visited point $b \in V^{i-1}$, the point b has been added to the search path Q by the SEARCHANDAUGMENT procedure, did not lead to the computation of an augmenting path, marked as visited and removed from the path. In this case, any point b'' that is reachable from b by an admissible path in $\overleftarrow{\mathcal{G}^{i-1}}$ (and therefore, is reachable by our partial DFS procedure from b) would have also been added to the path, marked as visited and removed from Q, i.e., any point $b'' \in B$ that has an admissible path to the visited point b in the residual graph $\mathcal{G}^{i-1}$ is also visited. By this observation, there are no admissible paths from any unvisited point in B to any visited point in $\mathcal{G}^{i-1}$. From Lemma B.1, for any newly formed admissible triple (b, r, b'), we have $b \in V^{i-1}$ and $b' \in U^{i-1}$. Thus, all admissible triples formed after augmenting $\hat{\tau}_\delta^{i-1}$ along P^i are from a visited point to an unvisited point, while there are no admissible paths from any unvisited point to any visited point; therefore, the newly formed admissible triples do not form any admissible cycles and (S3) holds for $\mathcal{G}^i$ as well.

Property (S2). We use the property (S3) to show that (S2) holds. For any point $b \in B$ that is marked as visited, as discussed above, if (S3) holds, all vertices that are reachable from b by an admissible path in our backward DFS (i.e., all points having an admissible path to b in the residual graph) are also visited. Since any free point $b_f \in B$ is unvisited, b_f does not have an admissible path to any visited point $b \in B$ and therefore, the visited points do not participate in an admissible augmenting path. Furthermore, the procedure marks a backward edge (r, b) as visited if, for each admissible triple (b', r, b), the point b' is visited. Since the point b' cannot be included in an admissible augmenting path, the visited backward edge (r, b) also does not form an admissible augmenting path.

$\square$

Lemma B.1. *During the execution of the* SEARCHANDAUGMENT *procedure, suppose P is an admissible augmenting path found by the procedure, and let $\mathcal{G}$ (resp. $\mathcal{G}'$) denote the residual graph before (resp. after) augmenting the transport plan along P. Let (b, r, b') denote an admissible triple in $\mathcal{G}'$ that is not admissible in $\mathcal{G}$. Assuming that (S3) holds prior to augmentation along P, the point b is marked as visited, and b' is marked as unvisited.*

Proof. Consider any triple (b, r, b') that is admissible in $\mathcal{G}'$ but not in $\mathcal{G}$. Recall that by the definition of the admissible triples, (r, b') is a backward edge in $\mathcal{G}'$ and $d_y(r, b) > d_y(r, b')$. Since the SEARCHANDAUGMENT procedure does not change the weights $y(\cdot)$, the only case where (b, r, b') is not admissible in $\mathcal{G}$ is when (r, b') is not a backward edge in $\mathcal{G}$, i.e., the pair (b', r) is in P as a forward edge, and augmenting $\hat{\tau}_\delta$ along P results in transporting mass from b' to r. On the other hand, by step 2(a) of the SEARCHANDAUGMENT procedure, a forward edge (b', r) will be added to the search path only if b' is the weighted nearest unvisited neighbor of r; in other words, since $d_y(r, b) > d_y(r, b')$ and the procedure added b' to the search path (instead of b), the point b was marked as visited by the procedure. Therefore, for any newly formed admissible triple (b, r, b'), point b (resp. b') is marked as visited (resp. unvisited). $\square$

B.2 Missing Details and Proofs of the INCREASEWEIGHTS Procedure

After the execution of the SEARCHANDAUGMENT procedure, no admissible augmenting paths remain in the residual graph, i.e., there are no admissible paths from the source vertex s to the sink vertex t. The INCREASEWEIGHTS procedure increases the weights of the subset of points in B that are reachable from s by admissible paths to expand their Voronoi cells and to create new admissible triples in the residual graph. For instance, in Figure 6 (left), the path from s to t is not admissible (the triple (b_2, r_2, b_3) is not admissible, as b_3 has a lower weighted distance to r_2 than b_2). As shown in Figure 6 (right), the INCREASEWEIGHTS procedure then increases the weight of the points b_1 and b_2 (which are reachable from s), leading to the formation of an admissible augmenting path (note that upon updating the weights, the regions corresponding to r_1 and r_2 have slightly changed). The details of the INCREASEWEIGHTS procedure are described below.

For each point $r \in A_\delta$, let $\mathcal{N}(r) \subseteq B$ denote the set of points $b \in B$ with $\hat{\tau}_\delta(r, b) > 0$, sorted in decreasing order of their weighted distance to r, i.e., $\mathcal{N}(r) = \{b_1, \ldots, b_k\}$ where $d_y(r, b_1) \geq d_y(r, b_2) \geq \ldots d_y(r, b_k)$. Mark all points $b \in B$ and all forward edges (b, r) as unvisited and set $\mathcal{K} = \emptyset$, let $U = B$ denote the set of unvisited points of B, and define $Q := \langle s \rangle$ as the search path that the algorithm grows. Execute the following steps until Q becomes empty.
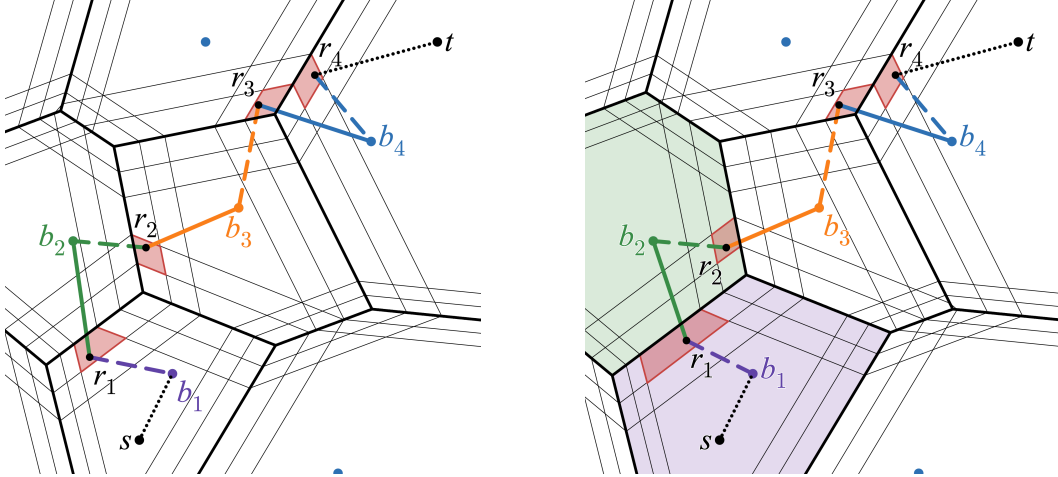
Figure 6: (left) After the execution of the SEARCHANDAUGMENT procedure, there are no admissible augmenting paths in $\mathcal{G}_\delta$, and (right) by increasing the weights of the points that are reachable from s by augmenting paths (points b_1 and b_2), new admissible triples are created (e.g. (b_2, r_2, b_3)), which might lead to the formation of admissible augmenting paths.

1. If $Q = \langle s \rangle$, then if there exists an unvisited point $b \in U$ such that $(s, b) \in \mathcal{G}_\delta$, then add b to Q as b_1. Otherwise, remove s from Q.

2. If $Q = \langle s, b_1, r_1, \ldots, b_i \rangle$ for some $i \geq 1$,

   (a) If there exists an unvisited forward edge (b_i, r) in $\mathcal{G}_\delta$, add r to Q as r_{i+1}.

   (b) Otherwise, mark b_i as visited, remove b_i from U, add b_i to $\mathcal{K}$, and remove b_i from Q.

3. If $Q = \langle s, b_1, r_1, \ldots, b_i, r_i \rangle$ for some $i \geq 1$, let $b := \arg\min_{b' \in U \cap \mathcal{N}(r)} d_y(r_i, b')$ denote the unvisited point of b with the minimum weighted distance to r among all points of B that transport mass to r.

   (a) If (b_i, r_i, b) is admissible, i.e., $d_y(r_i, b) > d_y(r_i, b_i)$, then add b as b_{i+1} to Q.

   (b) Otherwise, remove r_i from Q and mark (b_i, r_i) as visited.

After the DFS procedure terminates, for each point $b \in \mathcal{K}$, set $y(b) \leftarrow y(b) + \delta$. This completes the description of the DFS step. We next describe how to recompute the residual graph and the compressed transport plan with respect to the updated weights.

Let $y(\cdot)$ (resp. $y'(\cdot)$) denote the weights of the points in B after (resp. before) the weight updates, let $\mathcal{X}_\delta$ (resp. $\mathcal{X}'_\delta$) denote the partitioning of the set A with respect to weights $y(\cdot)$ (resp. $y'(\cdot)$), and let A_δ (resp. A'_δ) denote the set of representative points of the regions in $\mathcal{X}_\delta$ (resp. $\mathcal{X}'_\delta$). Furthermore, let $\hat{\tau}'_\delta$ denote the transport plan maintained by the algorithm for partitioning $\mathcal{X}'_\delta$. To compute the new transport plan $\hat{\tau}_\delta$ for the point set A_δ, the INCREASEWEIGHTS procedure first computes the arrangement $\mathcal{Y}$ of all $3n$ cells used to construct $\mathcal{X}'_\delta$ with all $3n$ cells used to construct $\mathcal{X}_\delta$, i.e., $\mathcal{Y}$ is the arrangement of Voronoi cell, δ-expanded Voronoi cell, and 2δ-expanded Voronoi cell of each point $b \in B$ both before and after weight updates. See Figure 7. For each region $\varphi \in \mathcal{X}'_\delta \cup \mathcal{X}_\delta$, let $\mathcal{C}(\varphi) \subseteq \mathcal{Y}$ denote the set of regions of $\mathcal{Y}$ that lie inside φ. For each region $\varrho \in \mathcal{Y}$, pick an arbitrary representative point r_ϱ inside ϱ. We denote the set of all representative points of the regions in $\mathcal{Y}$ by Y.

The INCREASEWEIGHTS procedure first converts $\hat{\tau}'_\delta$ to a transport plan $\hat{\tau}$ over the finer partitioning $Y \times B$ by simply splitting each region φ' in $\mathcal{X}'_\delta$ (and its mass transportation) to fine regions of $\mathcal{Y}$ inside φ'. The procedure then uses $\hat{\tau}$ to construct a transport plan $\hat{\tau}_\delta$ over $A_\delta \times B$ by merging the regions of $\mathcal{Y}$ (and their mass transportation) to regions of $\mathcal{X}_\delta$. The details are provided next.

For each point $b \in B$, each region $\varphi' \in \mathcal{X}'_\delta$ with $\hat{\tau}'_\delta(r_{\varphi'}, b) > 0$, and each $\varrho' \in \mathcal{C}(\varphi')$, set $\hat{\tau}(\varrho', b) = \frac{\mu(\varrho')}{\mu(\varphi')} \hat{\tau}'_\delta(r_{\varphi'}, b)$. This completes the description of the split step and the construction of $\hat{\tau}$. Next, the procedure constructs $\hat{\tau}_\delta$ by setting, for each point $b \in B$ and each region $\varphi \in \mathcal{X}_\delta$, $\hat{\tau}_\delta(r_\varphi, b) := \sum_{\varrho \in \mathcal{C}(\varphi)} \hat{\tau}(\varrho, b)$. The transport plan $\hat{\tau}_\delta$ is defined over $A_\delta \times B$.
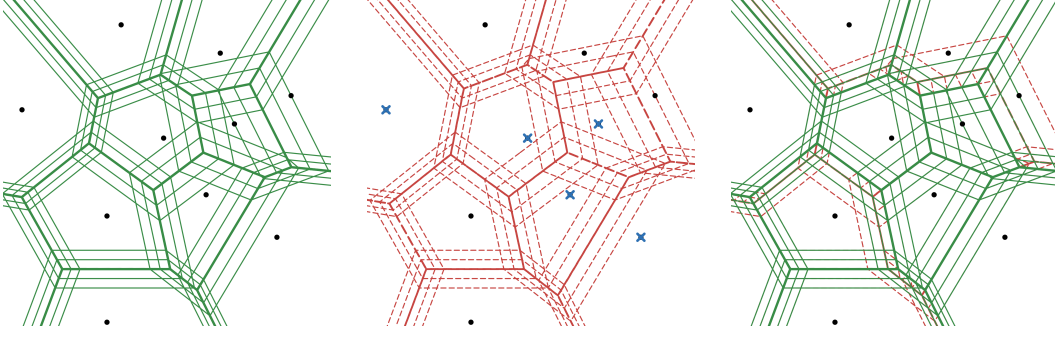
Figure 7: (left) The partitioning $\mathcal{X}'_\delta$ before updating weights, (middle) the partitioning $\mathcal{X}_\delta$ after updating the weights of the points of B that are reachable from s by admissible paths (the blue cross points), and (right) the combined partitioning $\mathcal{Y}$.

Finally, for each region $\varphi \in \mathcal{X}_\delta$, our algorithm stores a list $\mathcal{N}(r_\varphi)$ of all points $b \in B$ with $r_\varphi \in V_b^{2\delta}$, sorted in increasing order of their weights distance to r_φ, i.e., $\mathcal{N}(r_\varphi) = \langle b_1, \ldots, b_k \rangle, r_\varphi \in V_{b_i}^2 \delta$ for each $i \in [1, k]$, and $\mathrm{d}_y(r_\varphi, b_i) \leq \mathrm{d}_y(r_\varphi, b_j)$ for each $1 \leq i < j \leq k$. This completes the description of the INCREASEWEIGHTS procedure.

Lemma 3.2. *Suppose invariant (I1) holds at the start of the* INCREASEWEIGHTS *procedure. Then, during the execution of the* INCREASEWEIGHTS *procedure,*

(W1) the transport plan $\hat{\tau}_\delta, y(\cdot)$ remain δ-feasible,

(W2) the weight of each free point $b \in B$ increases by δ, and

(W3) the weight of each point $b \in B$ with free regions inside V_b^δ remains unchanged.

Proof. Let $y(\cdot)$ (resp. $y'(\cdot)$) denote the weights of the points after (resp. before) the execution of the INCREASEWEIGHTS procedure, and let $\mathcal{X}_\delta$ (resp. $\mathcal{X}'_\delta$) be the partitioning with respect to weights $y(\cdot)$ (resp. $y'(\cdot)$). For any point $a \in A$, let φ_a (resp. φ'_a) be the region in $\mathcal{X}_\delta$ (resp. $\mathcal{X}'_\delta$) that contains a, and let b_a (resp. b'_a) denote the weighted nearest neighbor of a in B with respect to weights $y(\cdot)$ (resp. $y'(\cdot)$). For any pair of points $(a, b) \in A \times B$ with $\tau'_\delta(a, b) > 0$, by the δ-feasibility of $\tau'_\delta, y'(\cdot)$, we have $\mathrm{d}_{y'}(a, b) - 2\delta \leq \mathrm{d}_{y'}(a, b'_a)$. To prove property (W1), we show that $\mathrm{d}_y(a, b) - 2\delta \leq \mathrm{d}_y(a, b_a)$.

Recall that the INCREASEWEIGHTS procedure finds the subset $\mathcal{K} \subset B$ of points that have admissible paths from the source vertex s of the residual graph and increases the weights of all points in $\mathcal{K}$ by δ. Consider the following cases:

- If $b \in \mathcal{K}$ is among the points whose weights are increased by the procedure, then

$$\mathrm{d}_y(a, b) = \mathrm{d}_{y'}(a, b) - \delta \leq \mathrm{d}_{y'}(a, b'_a) + \delta \leq \mathrm{d}_{y'}(a, b_a) + \delta \leq \mathrm{d}_y(a, b_a) + 2\delta,$$

  where the second inequality holds from δ-feasibility of $\tau'_\delta, y'(\cdot)$, the third inequality holds since b_a is the weighted nearest neighbor of a with respect to $y'(\cdot)$, and the last inequality holds since $y(b_a) \leq y'(b_a) + \delta$. Consequently, $\mathrm{d}_y(a, b) - 2\delta \leq \mathrm{d}_y(a, b_a)$.

- Otherwise, $b \notin \mathcal{K}$ and $\mathrm{d}_y(a, b) = \mathrm{d}_{y'}(a, b)$.

  - If $b_a \in \mathcal{K}$, then $\mathrm{d}_{y'}(r_{\varphi_a}, b) \leq \mathrm{d}_{y'}(r_{\varphi_a}, b_a)$ (since otherwise, the triple (b_a, r_{φ_a}, b) would have been an admissible triple and $b_a \in \mathcal{K}$ would have resulted in $b \in \mathcal{K}$). In this case, since the weighted nearest neighbor of a and r_{φ_a} are the same,

$$\mathrm{d}_{y'}(r_{\varphi_a}, b'_a) \leq \mathrm{d}_{y'}(r_{\varphi_a}, b) \leq \mathrm{d}_{y'}(r_{\varphi_a}, b_a) \leq \mathrm{d}_{y'}(r_{\varphi_a}, b'_a) + \delta,$$

    where the last inequality holds since increasing the weight of b'_a by δ made it the weighted nearest neighbor of a. Therefore, the region φ_a and consequently, the point a lie inside V_b^δ and $V_{b_a}^\delta$. Thus,

$$\mathrm{d}_y(a, b) = \mathrm{d}_{y'}(a, b) \leq \mathrm{d}_{y'}(a, b_a) + \delta = \mathrm{d}_y(a, b_a) + 2\delta.$$

  - Otherwise, if b_a is also not in $\mathcal{K}$, then $\mathrm{d}_y(a, b_a) = \mathrm{d}_{y'}(a, b_a)$ and,

$$\mathrm{d}_y(a, b) - 2\delta = \mathrm{d}_{y'}(a, b) - 2\delta \leq \mathrm{d}_{y'}(a, b_a) = \mathrm{d}_y(a, b_a).$$
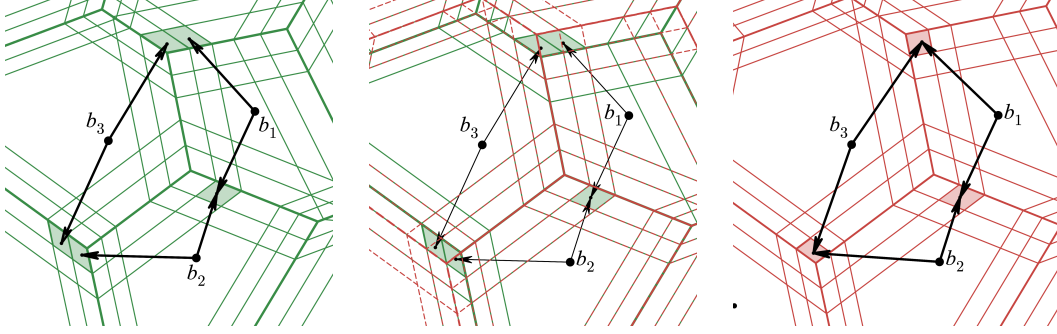
Figure 8: An example of a cycle in a transport plan that is created while increasing weights: (left) a transport plan that is a forest, (middle) the new Voronoi diagram and partitioning (red dashed lines) after increasing the weight of b_1, b_2, and b_3, and (right) a cycle formed in the new residual graph.

As a result, the transport plan τ_δ along with the updated weights $y(\cdot)$ is δ-feasible and property (W1) holds.

Note that each free point $b_f \in B$ has an edge from the source vertex s; therefore, $b_f \in \mathcal{K}$ and the INCREASEWEIGHTS increases the weight of b_f by δ, proving (W2). Furthermore, for each point $b \in B$ with a free region $\varphi \in V_b^\delta$, since we defined $d_y(r_\varphi, t) = \min_{b' \in B} d_y(r_\varphi, b')$, the triple (b, r_φ, t) is admissible. Therefore, the SEARCHANDAUGMENT procedure should have added r_φ and b to the search path, not found an admissible augmenting path, and marked b (resp. (r, t)) as visited. Therefore, there are no admissible paths from the source to b, or equivalently $b \notin \mathcal{K}$, and the weight of b remains unchanged, leading to (W3). $\qquad\square$

B.3 Missing Details of the ACYCLIFY Procedure

The goal of the ACYCLIFY procedure is to ensure that the invariant (I2) holds, i.e., that at the beginning of each iteration of our algorithm, the transport plan is a forest and there are no admissible cycles in the residual graph. The procedure runs in three steps: (1) make $\hat{\tau}_\delta$ a forest, as described in Section B.3.1, (2) cancel any admissible cycles from the residual graph, as described in Section B.3.2, and (3) acyclify the transport plan again, as described in Section B.3.1. Note that our procedure acyclifies the transport plan (to make it a forest) twice, in steps (1) and (3). Making the transport plan a forest in the first step is essential for the efficiency of the second step, and making it a forest in the third step is essential for invariant (I2), as canceling admissible cycles might introduce cycles in the transport plan.

B.3.1 Acyclifying the Transport Plan

Similar to the Acyclify procedure introduced in [54, Section 3.3], we use a dynamic tree structure to make $\hat{\tau}_\delta$ a forest as follows. Let $\mathcal{E} = \langle e_1, e_2, \ldots, e_u \rangle$ denote the set of all edges $e = (r, b) \in A_\delta \times B$ with $\hat{\tau}_\delta(r, b) > 0$. For any $k \leq u$, let $\mathcal{E}_k := \langle e_1, e_2, \ldots, e_k \rangle$. Define $F_0 := \emptyset$ as an empty forest and $\hat{\tau}_0'(r, b) = 0$ for all pairs $(r, b) \in A_\delta \times B$. Starting from $k = 1$, for any $k \leq u$, the algorithm computes a forest F_k and a transport plan $\hat{\tau}_k'$ defined over F_k using F_{k-1} and $\tilde{\tau}_{k-1}'$ as follows. If adding the edge e_k to F_{k-1} does not create a cycle, then the algorithm simply sets $F_k \leftarrow F_{k-1} \cup \{e_k\}$, $\hat{\tau}_k'(e_k) \leftarrow \hat{\tau}_\delta(e_k)$ and $\hat{\tau}_k'(e) \leftarrow \tau_{k-1}'(e)$ for all edges $e \in F_{k-1}$. Otherwise, adding e_k to F_{k-1} results in the creation of an even-length cycle $\mathcal{C}$. Let c denote the minimum capacity of the edges in $\mathcal{C}$, and let e^* denote the edge with the minimum capacity. Consider an ordering of the edges of the cycle $\mathcal{C}$ that starts with e^*, i.e., $\mathcal{C} = \langle e^* = e_1', e_2', \ldots, e_{2j}' \rangle$. The algorithm increases (resp. reduces) the mass transported along the edge e_{2i}' (resp. e_{2i-1}') by c for each $i \in [1, j]$. Finally, the algorithm sets $F_k \leftarrow F_{k-1} \cup \{e_k\} \setminus \{e^*\}$. This completes the description of step 1. Using the dynamic tree structure by Sleator and Tarjan [56], each operation takes $O(\log n)$ amortized time, and since $|\mathcal{E}| = O(n^3)$, this process takes a total of $O(n^3 \log n)$ time.
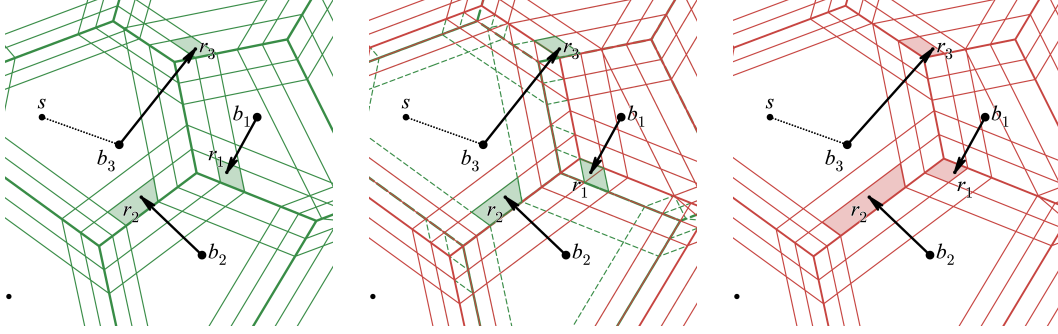
24

Figure 9: An example of the formation of admissible cycles due to updating the weights and the residual graph: (left) a transport plan with no admissible cycles in the corresponding residual graph, (middle) the new Voronoi diagram and partitioning (red lines) after increasing the weights of b_3 and b_2, and (right) an admissible cycle $\langle b_1, r_1, b_2, r_2, b_3, r_3 \rangle$ formed in the new residual graph.

B.3.2 Acyclifying the Admissible Triples

To remove all admissible cycles from the residual graph, we use a partial DFS similar to the one described in the SEARCHANDAUGMENT procedure. Let $\overleftarrow{\mathcal{G}_\delta}$ be the graph formed by reversing the direction of all the edges of $\mathcal{G}_\delta$. The procedure first marks all points of B and all backward edges as unvisited and defines $U := B$ as the set of unvisited points. While there exists an unvisited point $b \in B$, the procedure initializes a partial DFS by setting $Q = \langle b = b_1 \rangle$ and searches as follows until Q becomes empty.

1. If $Q = \langle b_1, r_1, \ldots, b_i \rangle$ for some $i \geq 1$,

    (a) If there are no unvisited backward edges (b_i, r) in $\overleftarrow{\mathcal{G}_\delta}$, then mark b_i as visited and remove b_i from Q and U.
    (b) Otherwise, there exists an unvisited backward edge (b_i, r_φ). Add r_φ to Q as r_{i+1}.

2. If $Q = \langle b_1, r_1, \ldots, b_i, r_i \rangle$ for some $i \geq 1$, let $b := \arg\min_{b' \in U} \mathrm{d}_y(r_i, b')$ be the unvisited point with the minimum weighted distance to r_i.

    (a) If (b, r_i, b_i) is admissible, i.e., $\mathrm{d}_y(r_i, b) < \mathrm{d}_y(r_i, b_i)$,
        – If b already exists in the path Q as b_j, then $C = \langle b_j, r_j, \ldots, b_i, r_i, b_{i+1} = b = b_j \rangle$ is an admissible cycle. Cancel the cycle C (as described below) and set $Q = \langle b_1, r_1, \ldots, b_j \rangle$.
        – Otherwise, add b as b_{i+1} to Q.
    (b) Otherwise, remove r_i from Q and mark the backward edge (r_i, b_i) as visited.

Given an admissible cycle $C = \langle b_j, r_j, \ldots, b_i, r_i, b_{i+1} = b_j \rangle$, the procedure cancels the cycle as follows. Let $bc(C) := \min_{t \in [j,i]} \hat{\tau}_\delta(r_t, b_t)$ denote the bottleneck capacity of the cycle C. For any forward edge (b_{t+1}, r_t) (resp. backward edge (r_t, b_t)) in C, set $\hat{\tau}_\delta(r_t, b_{t+1}) \leftarrow \hat{\tau}_\delta(r_t, b_{t+1}) + bc(C)$ (resp. $\hat{\tau}_\delta(r_t, b_t) \leftarrow \hat{\tau}_\delta(r_t, b_t) - bc(C)$). In this way, at least one of the backward edges of the cycle C is removed from the residual graph and the cycle has vanished.

Lemma 3.3. *Suppose invariant (I1) holds at the start of the* ACYCLIFY *procedure. Then, during the execution of the* ACYCLIFY *procedure,*

   (A1) the transport plan $\hat{\tau}_\delta, y(\cdot)$ remains δ-feasible, and
   (A2) the transport plan $\hat{\tau}_\delta$ is a forest and there are no admissible cycles in the residual graph.

Proof. In the first step of the ACYCLIFY procedure, the algorithm makes the transportation network to be a DAG. Note that the resulting transportation network is a subset of the transportation network before the ACYCLIFY procedure, and therefore, for any transporting edge $(r, b) \in A_\delta \times B$, the point b is a 2δ-weighted nearest neighbor of r, i.e., the transport plan obtained after the first step of the ACYCLIFY procedure is δ-feasible.

Next, we show property (A1) and (A2) for the second step of the procedure. By construction, all triples on the search path maintained by the procedure are admissible, and therefore, any cycle

25

computed by the second step of ACYCLIFY procedure is admissible. Since all forward edges on the computed cycles are from points $b \in B$ to regions in $V_b^{2\delta}$, canceling a cycle C does not violate the δ-feasibility condition (F1); hence, the transport plan $\hat{\tau}_\delta, y(\cdot)$ is a δ-feasible transport plan during the execution of the second step of the ACYCLIFY procedure and (A1) holds.

To prove property (A2), we show that

(A3) any point $b \in B$ (resp. backward edge (r, b)) marked as visited will not form an admissible cycle during the execution of the procedure, and

(A4) during the execution of the second step of the ACYCLIFY procedure, the subgraph of the residual graph induced by visited vertices and their neighboring regions does not have any cycles of admissible triples. Furthermore, there are no admissible paths from an unvisited point to a visited point.

Assuming property (A3) holds, any point $b \in B$ that is marked as visited does not participate in an admissible cycle. Since the ACYCLIFY procedure stops when all points in B are visited, there are no admissible cycles in the residual graph. Furthermore, since the transport plan is maintained by a dynamic tree structure the transport plan is a forest. Therefore, to prove property (A2), we show that (A3) holds.

Furthermore, the property (A3) is a direct corollary of property (A4), as explained next: For each visited point $b \in B$, all points $b' \in B$ having admissible paths to b are also visited (note that the ACYCLIFY procedure searches on the residual graph in the reverse direction of the edges), and if (A4) holds, there are no admissible cycles solely formed by visited vertices; hence, b does not form admissible cycles. Furthermore, the procedure marks a backward edge (r, b) as visited if, for each admissible triple (b', r, b), the point b' is visited. Since b' is not a part of any admissible cycles (assuming (A4) holds), the triple (b', r, b) also cannot be a part of an admissible cycle. Therefore, to prove property (A3), we prove that (A4) holds in the following.

We use an inductive argument to prove (A4). Let $C^1, \ldots, C^k$ denote the sequence of admissible cycles found by the procedure, and let $\hat{\tau}_\delta^0, \ldots, \hat{\tau}_\delta^k$ denote the sequence of transport plans, where $\hat{\tau}_\delta^0$ is the transport plan maintained by the algorithm at the beginning of the second step of the procedure and $\hat{\tau}_\delta^i$ is obtained by canceling $\hat{\tau}_\delta^{i-1}$ along C^i. The property (A4) trivially holds at the beginning of the execution of the second step of the ACYCLIFY procedure.

Note that the ACYCLIFY procedure marks a point $b \in B$ as visited only if the search from b did not lead to finding an admissible cycle, i.e., for all pairs $(b', r) \in B \times A_\delta$ such that (b', r, b) is admissible, the point b' is marked as visited and the backward edge (r, b) is marked as visited. Hence, all points having an admissible path to the visited points (i.e., all points that are reachable from the visited points in our backward DFS) are also visited. Therefore, Assuming that property (A4) holds before marking b as visited, property (A4) holds after marking b as visited as well. Next, we show the same when we cancel a cycle C^i.

First, note that all points of B that are on the search path are unvisited. Therefore, for any admissible cycle found by the procedure, the points of B on the cycle are unvisited. Using an identical proof as in Lemma B.1, one can show that after canceling a cycle, for any newly formed admissible triple (b, r, b'), the point b is visited and the point b' is unvisited. In other words, canceling a cycle C^i only creates additional admissible paths from visited points to unvisited points. Assuming that (A4) holds before canceling C^i, since no new admissible triples are created from a visited point to another visited point, the subgraph induced by visited points and their neighboring regions remain free of admissible cycles after canceling C^i. Furthermore, since no new admissible triples are created from an unvisited point to a visited point upon canceling C^i, there will be no admissible paths from an unvisited point to a visited point after cancellation; hence, (A4) remains satisfied. □

C Missing Details of Section 4

In this section, we analyze the efficiency of the three procedures SEARCHANDAUGMENT, IN-CREASEWEIGHTS, and ACYCLIFY.

C.1 Efficiency of the SEARCHANDAUGMENT Procedure

The SEARCHANDAUGMENT procedure runs a partial DFS on the residual graph to find a set of admissible augmenting paths. The partial DFS procedure, upon backtracking from a point $b \in B$ (resp. $r \in A_\delta$), marks the point b (resp. the backward edge (b', r) used to reach r) as visited and does not visit the point b (resp. the backward edge (b', r)) again in the same execution. Upon finding an augmenting path P, the procedure augments the transport plan along P in $O(|P|)$ time. Let $\langle P_1, \ldots, P_k \rangle$ denote the set of all augmenting paths found by the SEARCHANDAUGMENT procedure. In 2 dimensions (resp. d dimensions), since the residual graph has $O(n^3)$ edges (resp. $O(n^{d+1})$), the running time of the procedure would be $O(n^3 + \sum_{i=1}^k |P_i|)$ (resp. $O(n^{d+1} + \sum_{i=1}^k |P_i|)$). In Lemma C.1, we show that $\sum_{i=1}^k |P_i| = O(n^3)$ (resp. $\sum_{i=1}^k |P_i| = O(n^{d+1})$). Hence, each execution of the SEARCHANDAUGMENT procedure takes $O(n^3)$ (resp. $O(n^{d+1})$) time.

Lemma C.1. *The total length of augmenting paths computed during the execution of the* SEARCHANDAUGMENT *procedure is* $O(n^3)$ *in 2 dimensions and* $O(n^{d+1})$ *in d dimensions.*

Proof. Let $\hat{\tau}_\delta^0$ denote the transport plan maintained by the algorithm at the beginning of execution of the SEARCHANDAUGMENT procedure. To prove this lemma, we categorize the augmenting paths found by the procedure based on the source of their bottleneck capacity, namely (1) set $\mathcal{P}_v$ consisting of augmenting paths whose bottleneck capacity is determined based on the residual capacity of its endpoints, and (2) set $\mathcal{P}_e$ consisting of augmenting paths whose bottleneck capacity is determined based on mass transportation over its backward edges. We first show that $|\mathcal{P}_v| = O(n^2)$ and then show the same bound for $\mathcal{P}_e$. Since each augmenting path has a length of at most $2n$, we then conclude that the total length of all augmenting paths is $O(n^3)$.

Let P be an augmenting path in $\mathcal{P}_v$. If the bottleneck capacity of P is determined by a free point $b \in B$ (resp. free region $r \in A_\delta$), then the mass of b (resp. r) will be fully transported after augmentation; therefore, since $|A_\delta \cup B| = O(n^2)$, we have $|\mathcal{P}_v| = O(n^2)$. Next, let P be an augmenting path in $\mathcal{P}_e$; in this case, the backward edge (r, b) determining the bottleneck capacity of P will be removed from the transport plan after augmentation. Note that by Lemma B.1, for any newly formed admissible triples (b', r, b), the point b' is already marked as visited. By property (S2) in Lemma 3.1, the point b' cannot form an admissible augmenting path during the same execution of the SEARCHANDAUGMENT procedure. Hence, the edge (r, b) determining the bottleneck capacity of P was a backward edge of the initial transport plan $\hat{\tau}_\delta^0$ and augmentation along each path $P \in \mathcal{P}_e$ removes one of the transporting edges of the transport plan $\hat{\tau}_\delta^0$. Using invariant (I2), $|\mathcal{P}_e| = O(n^2)$, as claimed. Hence, the total number of augmenting paths found by the procedure is $O(n^2)$, and since each augmenting path has a length of at most $2n$, their total length is $O(n^3)$.

Next, we extend our analysis to d dimensional space, for any $d \geq 2$. Note that in d dimensions, the residual graph has $O(n^d)$ vertices and $O(n^{d+1})$ edges. Hence, $|\mathcal{P}_v| = O(n^d)$. Since the transport plan $\hat{\tau}_\delta^0$ is a forest over the point set $A_\delta \cup B$, the total number of edges transporting a positive mass in $\hat{\tau}_\delta^0$ would be $O(n^d)$; since augmenting the transport plan along each augmenting path in $\mathcal{P}_e$ eliminates one of the edges transporting positive mass in $\hat{\tau}_\delta^0$, $|\mathcal{P}_e| = O(n^d)$. Finally, since each augmenting path has a length of at most $2n$, the total length of all augmenting paths found by the SEARCHANDAUGMENT procedure would be $O(n^{d+1})$. $\square$

C.2 Efficiency of the INCREASEWEIGHTS Procedure

In this section, we show that for 2-dimensional (resp. d-dimensional) distributions, the INCREASEWEIGHTS procedure runs in $O(n^2(\Phi + n \log n))$ (resp. $O(n^d(\Phi + n \log n))$) time. The INCREASEWEIGHTS procedure runs a DFS that visits each edge of the residual graph at most once and has a total running time of $O(n^3)$ (resp. $O(n^{d+1})$). Furthermore, in the arrangement used to construct partitioning $\mathcal{Y}$, each point $b \in B$ has at most 6 Voronoi cells (three cells that are used in the construction of $\mathcal{X}_\delta$ and three that are used in the construction of $\mathcal{X}_\delta'$). Using a similar discussion as Section A, one can show that the total number of vertices in the arrangement used to construct $\mathcal{Y}$ is $O(n^2)$ (resp. $O(n^d)$), and the number of regions in $\mathcal{Y}$ is at most $O(n^2)$ (resp. $O(n^d)$). The construction of the transport plan $\hat{\tau}$ can be done in $O(n^2(\Phi + n))$ (resp. $O(n^d(\Phi + n))$) time since (1) the mass of all regions in $\mathcal{Y}$ can be determined in $O(n^2\Phi)$ (resp. $O(n^d\Phi)$) time (partitioning the regions in $\mathcal{Y}$ into simplices remains an arrangement with $O(n^2)$ (resp. $O(n^d)$) vertices and therefore,

have $O(n^2)$ (resp. $O(n^d)$) regions), and (2) the mass transported on each pair $(\varrho, b) \in \mathcal{Y} \times B$ can be determined in $O(1)$ time. Converting $\hat{\tau}$ to $\hat{\tau}_\delta$, as is done in the merge step, also takes $O(n^3)$ (resp. $O(n^{d+1})$) time, given that the total complexity of $\hat{\tau}$ is $O(n^3)$ (resp. $O(n^{d+1})$). Finally, storing a sorted list of neighbors for each region $r \in A_\delta$ takes $O(n^3 \log n)$ (resp. $O(n^{d+1}) \log n$) time in total. Hence, the execution of the INCREASEWEIGHTS procedure takes $O(n^2(\Phi + n \log n))$ (resp. $O(n^d(\Phi + n \log n))$) time.

Lemma C.2. *Each execution of the* INCREASEWEIGHTS *procedure takes* $O(n^2(\Phi + n \log n))$ *time in 2 dimensions and* $O(n^d(\Phi + n \log n))$ *time in d dimensions.*

C.3 Efficiency of the ACYCLIFY Procedure

The first step of this procedure uses a dynamic tree structure to acyclify the transport plan $\hat{\tau}_\delta$. Using the dynamic tree structure by Sleator and Tarjan [56], since the total number of edges of the graph is $O(n^3)$ in 2 dimensions (resp. $O(n^{d+1})$ in d dimensions), the running time of this step would be $O(n^3 \log n)$ (resp. $O(n^{d+1} \log n)$). In the second step, the procedure runs a partial DFS procedure on the residual graph and cancels the admissible cycles. The partial DFS procedure, upon backtracking from a point $b \in B$ (resp. $r \in A_\delta$), marks the point b (resp. the backward edge (b', r) used to reach r) as visited and does not visit the point b (resp. the edge (b', r)) again in the same execution. Furthermore, upon finding an admissible cycle C, it cancels the cycle in $O(|C|)$ time. Let $\langle C_1, \ldots, C_k \rangle$ denote the set of all cycles found in the execution of the ACYCLIFY procedure. In Lemma C.4, we show that $\sum_{i=1}^{k} |C_i| = O(n^3)$ in 2 dimensions (resp. $\sum_{i=1}^{k} |C_i| = O(n^{d+1})$ in d dimensions). Given that the size of the residual graph is at most $O(n^3)$ (resp. $O(n^{d+1})$), the second step of the ACYCLIFY procedure takes a total of $O(n^3 + \sum_{i=1}^{k} |C_i|) = O(n^3)$ (resp. $O(n^{d+1})$) time, leading to the following lemma.

Lemma C.3. *Each execution of the* ACYCLIFY *procedure takes* $O(n^3 \log n)$ *time in 2 dimensions and* $O(n^{d+1} \log n)$ *time in d dimensions.*

Lemma C.4. *The total length of admissible cycles computed during the execution of the second step of the* ACYCLIFY *procedure is* $O(n^3)$ *in 2 dimensions and* $O(n^{d+1})$ *time in d dimensions.*

Proof. Let $\hat{\tau}_\delta^0$ denote the transport plan maintained by the algorithm at the beginning of execution of the ACYCLIFY procedure. To prove this lemma, we show that the ACYCLIFY procedure finds $O(n^2)$ admissible cycles, where each cycle has a length of at most $2n$; hence, the total length of all cycles found by the procedure would be $O(n^3)$.

Let C be an admissible cycle found by the procedure; in this case, the backward edge determining the bottleneck capacity of C will be removed from the transport plan after cancellation. For any admissible triple (b, r, b') formed after canceling C, using an identical argument as in Lemma B.1, one can show that the edge (r, b') is a backward edge that was on the cycle C as a forward edge, and the point b is marked as visited; hence, by Lemma 3.3, the point b does not form an admissible cycle in the same execution of the ACYCLIFY procedure and therefore, the newly formed backward edge (r, b') cannot be included in any admissible cycles. Therefore, each cycle cancellation removes one of the backward edges of $\hat{\tau}_\delta^0$, which is the transport plan obtained after the first step of the ACYCLIFY procedure, i.e., the transportation network of $\hat{\tau}_\delta^0$ is a forest and the number of its transporting edges is $O(n^2)$. Therefore, the total number of cycles found by the ACYCLIFY procedure is $O(n^2)$, and their total length is $O(n^3)$, as claimed.

We next show that the total length of admissible cycles in d dimensions is $O(n^{d+1})$ in d dimensions, for any $d \geq 2$. Note that in d dimensions, the residual graph has $O(n^{d+1})$ edges. Since the transport plan $\hat{\tau}_\delta^0$ is a forest over the point set $A_\delta \cup B$, the total number of edges transporting a positive mass in $\hat{\tau}_\delta^0$ would be $O(n^d)$; since canceling each admissible cycle eliminates one of the edges transporting positive mass in $\hat{\tau}_\delta^0$, $|\mathcal{C}| = O(n^d)$. Since each admissible cycle has a length of at most $2n$, the total length of all admissible cycles found by the INCREASEWEIGHTS procedure would be $O(n^{d+1})$. $\square$

C.4 Number of Iterations

Lemma 4.1. *For each scale δ, the total number of iterations of step 2 of our algorithm is $O(n)$.*

Proof. Let $\tau_{2\delta}, y_{2\delta}(\cdot)$ denote the 2δ-feasible transport plan computed by our algorithm for scale 2δ, and let $\tau_{\delta}, y_{\delta}(\cdot)$ denote a partial transport plan maintained during the execution of step 2 of our algorithm. Let $\mathcal{X}_{2\delta}$ (resp. $\mathcal{X}_{\delta}$) denote the partitioning of the set A with respect to weights $y_{2\delta}(\cdot)$ (resp. $y_{\delta}(\cdot)$). Let $\mathcal{Y}$ be the arrangement of all $3n$ cells used to construct $\mathcal{X}_{2\delta}$ with all $3n$ cells used to construct $\mathcal{X}_{\delta}$, i.e., $\mathcal{Y}$ is the arrangement of Voronoi cell, δ-expanded Voronoi cell, and 2δ-expanded Voronoi cell of each point $b \in B$ with respect to weights $y_{\delta}(\cdot)$ along with the Voronoi cell, 2δ-expanded Voronoi cell, and 4δ-expanded Voronoi cell of each point $b \in B$ with respect to weights $y_{2\delta}(\cdot)$. For each region $\varrho \in \mathcal{Y}$, pick an arbitrary representative point r_ϱ inside ϱ. We denote the set of all representative points of the regions in $\mathcal{Y}$ by Y. Let $\hat{\tau}_{2\delta}$ (resp. $\hat{\tau}_\delta$) denote the compressed transport plan for $\tau_{2\delta}$ (resp. τ_δ) using the partitioning $\mathcal{Y}$. Note that the partitioning $\mathcal{Y}$ is a refinement of both partitionings $\mathcal{X}_\delta$ and $\mathcal{X}_{2\delta}$. Define $\tau' := \hat{\tau}_{2\delta} - \hat{\tau}_\delta$.

We construct a bipartite graph $\mathcal{G}'$ over $Y \times B$, where for any pair $(r, b) \in Y \times B$, there exists an edge directed from r to b if $\tau'(r, b) < 0$ and directed from b to r if $\tau'(r, b) > 0$. Consider any directed path $P = \langle r_1, b_1, \ldots, r_k, b_k \rangle$ from a free region $r \in Y$ to a free point $b \in B$ (with respect to $\hat{\tau}_\delta$). The path P is an augmenting path in the residual graph corresponding to $\hat{\tau}_\delta, y_\delta(\cdot)$.

Similar to the standard graph algorithms, we define the net-cost of the path P as $\phi(P) := \sum_{i=1}^{k} d(r_k, b_k) - \sum_{i=1}^{k-1} d(r_k, b_{k+1})$. Let $b_0 := b_{r_1}$ be the weighted nearest neighbor of r_1. Then, we can rewrite the net-cost of P as

$$\phi(P) = d(r_1, b_0) + \sum_{i=1}^{k} \big[d(r_i, b_i) - d(r_i, b_{i-1}) \big]$$

$$= d_{y_\delta}(r_1, b_0) + \sum_{i=1}^{k} \big[d_{y_\delta}(r_i, b_i) - d_{y_\delta}(r_i, b_{i-1}) \big] + y_\delta(b_k).$$

Due to δ-feasibility of the transport plan $\hat{\tau}_\delta, y_\delta(\cdot)$, for all edges (r_i, b_{i-1}), $i \in [1, k]$, the point b_{i-1} is a 2δ-weighted nearest neighbor of r_i; hence, $d_{y_\delta}(r_i, b_i) - d_{y_\delta}(r_i, b_{i-1}) \geq -2\delta$. Since the length of P is at most $2n - 1$,

$$\phi(P) = d_{y_\delta}(r_1, b_0) + \sum_{i=1}^{k} \big[d_{y_\delta}(r_i, b_i) - d_{y_\delta}(r_i, b_{i-1}) \big] + y_\delta(b_k) \geq d_{y_\delta}(r_1, b_0) + y_\delta(b_k) - 2n\delta. \quad (13)$$

Similarly, we can rewrite the net-cost of P using weights $y_{2\delta}$ as follows.

$$\phi(P) = d_{y_{2\delta}}(r_1, b_0) + \sum_{i=1}^{k} \big[d_{y_{2\delta}}(r_i, b_i) - d_{y_{2\delta}}(r_i, b_{i-1}) \big] + y_{2\delta}(b_k).$$

According to the 2δ-feasibility of $\hat{\tau}_{2\delta}, y_{2\delta}(\cdot)$, for all edges (r_i, b_i), $i \in [1, k]$, the point b_i is a 4δ-weighted nearest neighbor of r_i (with respect to weights $y_{2\delta}(\cdot)$); hence, $d_{y_{2\delta}}(r_i, b_i) - d_{y_{2\delta}}(r_i, b_{i-1}) \leq 4\delta$. Since the length of P is at most $2n - 1$,

$$\phi(P) = d_{y_{2\delta}}(r_1, b_0) + \sum_{i=1}^{k} \big[d_{y_{2\delta}}(r_i, b_i) - d_{y_{2\delta}}(r_i, b_{i-1}) \big] + y_{2\delta}(b_k) \leq d_{y_{2\delta}}(r_1, b_0) + y_{2\delta}(b_k) + 4n\delta.$$

$$(14)$$

By property (W3) in Lemma 3.3, during the execution of step 2 of our algorithm, the weights of the points in B with free regions in their δ-expansion remain unchanged; therefore, since b_0 contains free regions inside its δ-expanded Voronoi cell, $y_{2\delta}(b_0) = y_\delta(b_0)$ and $d_{y_{2\delta}}(r_1, b_0) = d_{y_\delta}(r_1, b_0)$. Combining with Equations (13) and (14),

$$y_\delta(b_k) - 2n\delta \leq \phi(P) - d_{y_\delta}(r_1, b_0) = \phi(P) - d_{y_{2\delta}}(r_1, b_0) \leq y_{2\delta}(b_k) + 4n\delta.$$

Equivalently,

$$y_\delta(b_k) - y_{2\delta}(b_k) \leq 6n\delta.$$

By property (W2) in Lemma 3.3, our algorithm increases the weight of the point b_k by δ in each iteration while its mass is not fully transported by $\hat{\tau}_\delta$; therefore, the point b_k cannot remain free after $6n$ iterations, i.e., after $O(n)$ iterations, there cannot be any remaining free points in B and the step 2 of our algorithm terminates after $O(n)$ iterations. $\qquad\square$

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: This paper presents a new combinatorial framework for the semi-discrete OT problem. It is used to design an algorithm for computing a ε-close semi-discrete transport plan, as well as a data structure for the discrete OT problem. Our main results are stated in Theorems 1.1 and 1.2 in the introduction and are summarized in the abstract.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Our semi-discrete OT algorithm relies on the existence of an oracle that computes the mass of μ inside a triangle, which is common in existing algorithms for the semi-discrete optimal transport problem. We clearly state this assumption in the abstract and Theorem 1.1.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: All theorems and lemmas contain the full set of assumptions, and their proofs are either provided in the main text or the appendix.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [NA]

   Justification: This paper does not include experiments.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [NA]

   Justification: This paper does not include experiments.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [NA]

   Justification: This paper does not include experiments.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [NA]

   Justification: This paper does not include experiments.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [NA]

Justification: This paper does not include experiments.

9. **Code Of Ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

Answer: [Yes]

Justification: There are no data or human subjects involved in the research process of this paper and there are no known societal impacts and potentially harmful consequences associated with the research conducted in this paper.

10. **Broader Impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [NA]

Justification: None of the categories described by the NeurIPS Code of Ethics, namely, safety, security, discrimination, surveillance, deception and harassment, environment, human rights, and bias and fairness is impacted by the research conducted in this paper.

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: The paper poses no such risks.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [NA]

Justification: The paper does not use existing assets.

13. **New Assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [NA]

Justification: The paper does not release new assets.

14. **Crowdsourcing and Research with Human Subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: This paper does not involve crowdsourcing nor research with human subjects.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: This paper does not involve crowdsourcing nor research with human subjects.