Overcoming Computational Complexity Barriers for Optimal Transport in Discrete and Semi-Discrete Settings

Pouyan Shirzadian

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Application

Sharath Raghvendra, Chair
Pankaj K. Agarwal
Lenwood S. Heath
Bo Ji
Jamie Sikora

August 29, 2025
Blacksburg, Virginia

Keywords: Optimal Transport, Bipartite Matching, k-Server, Exact Algorithms

Overcoming Computational Complexity Barriers for Optimal Transport in Discrete and Semi-Discrete Settings

Pouyan Shirzadian

(ABSTRACT)

Given two probability distributions μ and ν, Optimal Transport (OT) measures the minimum effort required to transport mass between μ and ν. OT provides a meaningful distance between distributions and acts as a dissimilarity measure between them. Due to its useful statistical properties, OT cost has been used in numerous machine-learning applications.

More formally, given a d-dimensional continuous (resp. discrete) probability distribution μ defined over a support $A \subset \mathbb{R}^d$ and a discrete distribution ν defined over a support $B \subset \mathbb{R}^d$, the semi-discrete (resp. discrete) optimal transport problem asks for computing a minimum-cost plan to transport mass between μ and ν. In the special case of the discrete OT problem, where μ and ν are defined on sets A and B of n points each, and any point in $A \cup B$ is given $1/n$ mass, the problem is equivalent to the minimum-cost bipartite matching problem. In this thesis, we study the semi-discrete OT and the minimum-cost bipartite matching problems.

The sensitivity of OT to noise has motivated the study of robust variants. Two such formulations are: (i) the α-optimal partial transport, which is the minimum cost required to transport a mass of α, and (ii) the λ-robust optimal transport, which regularizes the OT problem using the total variation (TV) distance.

For a continuous distribution μ defined over a bounded support $A \subset \mathbb{R}^d$ and a discrete distribution ν with a support $B \subset \mathbb{R}^d$ of n points, suppose $C_{\max}$ denotes the diameter of

the supports of μ and ν and assume we have access to an oracle that outputs the mass of μ inside a constant-complexity region in $O(1)$ time. In this thesis, given a parameter $\varepsilon > 0$, we present the following results for the semi-discrete OT problem between μ and ν:

(i) *Additive approximation for semi-discrete OT:* We present an ε-additive approximation algorithm for the semi-discrete OT problem that runs in $n^{O(d)} \log \frac{C_{\max}}{\varepsilon}$ time. Our algorithm works for several ground distances, including the ℓ_p-norm and the squared-Euclidean distance.

(ii) *Combinatorial algorithm for semi-discrete OT:* 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 for 2-dimensional settings, computes an ε-additive approximate semi-discrete transport plan in $\tilde{O}(n^4 \log \frac{1}{\varepsilon})$ time.

(iii) *Combinatorial algorithm for semi-discrete robust OT:* We provide a novel characterization of the optimal solutions to the semi-discrete robust OT problem and show that they can be represented as a restricted Voronoi diagram. We also present an algorithm that computes an ε-additive approximate solution in $O(n^4 \log n \log \frac{1}{\varepsilon})$ time in 2 dimensions and $n^{O(d)} \log(1/\varepsilon)$ time in d dimensions.

Furthermore, for point sets A and B of size n, let Δ be the spread, i.e., the ratio of the distance of the farthest to the closest pair of points in $A \cup B$. Furthermore, let $\Phi(n)$ be the query/update time of a d-dimensional dynamic weighted bichromatic closest pair data structure. For point sets in d dimensions, in this thesis, we present the following result for the bipartite matching problem:

(iv) *Robust algorithm for stochastic Euclidean matching:* We present a new algorithm to

compute a minimum-cost bipartite matching under Euclidean distances between A and B with a worst-case execution time of $\tilde{O}(n^2\Phi(n)\log\Delta)$. However, when A and B are drawn independently and identically from a fixed distribution that is not known to the algorithm, the execution time of our algorithm is, in expectation, $\tilde{O}(n^{2-\frac{1}{2d}}\Phi(n)\log\Delta)$.

For any given metric space, obtaining an optimal solution to the offline version of the classical k-server problem reduces to solving a minimum-cost partial bipartite matching between two point sets A and B within that metric space. Our final result in this thesis is the following:

(v) *Efficient exact offline algorithm for k-server:* We present an $\tilde{O}(n^{2-\frac{1}{2d+1}}\Phi(n)\log\Delta)$ time algorithm for solving the instances of minimum-cost partial bipartite matching defined by the offline version of the k-server problem.

Overcoming Computational Complexity Barriers for Optimal Transport in Discrete and Semi-Discrete Settings

Pouyan Shirzadian

(GENERAL AUDIENCE ABSTRACT)

Every day, we encounter matching problems, such as assigning taxis to passengers or pairing deliveries with addresses. Similar challenges arise in computer science, particularly in fields like machine learning and generative modeling, where we often need to compare sets of data. For example, to evaluate the quality of images produced by a model, we compare them to real-world images. A mathematical tool called Optimal Transport (OT) helps solve such problems by finding the most efficient way to move "mass" from one group (called a distribution) to another. OT has become a popular method in machine learning, computer vision, and statistics because it offers a meaningful way to compare complex data distributions.

In the first part of this thesis, we focus on the semi-discrete OT problem, where one distribution is continuous (i.e., a smooth map of mass or values over a region) and the other consists of a finite set of points. We develop new algorithms that make these computations faster and more accurate.

Unfortunately, real-world data is often noisy, which can make OT results less reliable. To address this, researchers have introduced robust variants of OT that allow for transporting only a portion of the mass to tolerate small errors in the data. In this thesis, we provide new characterizations of the solutions to these robust OT problems and design efficient algorithms that remain accurate even when the input data is imperfect.

In the second part of the thesis, we study a special case of OT known as bipartite matching, a classic computer science problem where the goal is to pair points from two sets at minimum cost. We develop a new algorithm that performs particularly well when the input datasets are similar, an advantage in many practical settings.

Finally, we apply our matching techniques to the well-known k-server problem, which models how systems with limited resources can respond efficiently to a stream of requests. Our algorithm is efficient and outperforms existing methods when the number of servers is not too small.

In summary, this thesis introduces new methods to solve optimal transport and matching problems. These advances have practical implications for data science, logistics, and foundational areas of computer science.

Dedication

To my beautiful wife, my loving parents, and my supportive brother — your patience, wisdom, and kindness have been my anchor through every challenge. Thank you for being my greatest source of strength and my constant companions through every season of life.

Acknowledgments

My heartfelt thanks go to my advisor, Dr. Sharath Raghvendra, whose steadfast support, sharp insights, and constant encouragement have guided me throughout my Ph.D. His mentorship has been a cornerstone of my growth, shaping both my research perspective and my ambition as a scholar.

I am sincerely grateful to Dr. Pankaj K. Agarwal for his invaluable insights, generous support, and thoughtful feedback throughout my Ph.D. journey. It has been a true privilege to collaborate with him, and the experience has been profoundly enriching.

I am also thankful to the committee members, Dr. Lenwood S. Heath, Dr. Jamie Sikora, and Dr. Bo Ji for their valuable feedback, time, and support throughout my research and thesis process. Their perspective helped me refine the way I communicate technical ideas and frame the broader impact of my work.

I would like to acknowledge my collaborator, Keegan Yao, for years of engaging discussions and fruitful teamwork that helped bring many ideas in this thesis to life, and to my co-authors at Virginia Tech, Kaiyi Zhang, Rachita Sowle, and Akshaykumar Gattani, for creating a supportive and intellectually stimulating environment.

I gratefully acknowledge the support of the National Science Foundation through grants CCF-1909171 and CCF-2223871, which helped make this research possible. Finally, I am grateful to the broader academic community for providing the tools, resources, and opportunities that made this research possible.

Contents

List of Figures

Chapter 1

Introduction

Optimal Transport (OT), which is also referred to by the p-Wasserstein distance, is a fundamental mathematical concept for comparing probability distributions by quantifying the most efficient way to transform one distribution into another. The problem was originally formulated in the 18th century by Monge [116] and further developed by Kantorovich [89] and Rachev [130]. Put simply, for distributions μ and ν in a metric space equipped with a distance function $\mathrm{c}(\cdot,\cdot)$, OT measures the minimum cost required to move probability mass from one distribution to the other, where the cost of transporting a mass of β between point u and point v is $\beta\mathrm{c}(u,v)$. For any parameter $p \geq 1$, the p-Wasserstein problem aims to minimize the OT cost under distances $\mathrm{c}(\cdot,\cdot)^p$. This deceptively simple problem has evolved into a rich field of study that connects analysis, geometry, probability, and computer science. At its core, OT captures the geometry of the space in which the distributions lie, offering a structured and interpretable means of measuring dissimilarity.

In mathematics, OT has played a central role in areas such as partial differential equations, where it provides variational formulations for nonlinear diffusion processes [18, 64, 74, 86, 142]. Other applications include goodness-of-fit tests [28, 83] and geometric optics [113, 156]. In recent years, the rise of data-driven applications has led to a renewed interest in OT, particularly in machine learning [36, 43, 63, 88, 106, 122, 155], computer vision applications [23, 82, 143], generative modeling [17, 19, 40, 57, 78, 141], computer graphics [53, 73, 126], and parameter estimation [27, 105]. Traditional distance metrics such as Euclidean

or KL divergence often fall short when the support of the distributions differs or when one seeks to incorporate geometric structure in the comparison. The optimal transport cost, on the other hand, takes into account the actual locations and displacements of probability mass, enabling more meaningful comparisons in tasks like clustering [62, 161], barycenter computation [10, 44, 51, 154], and domain adaptation [46, 47, 48, 94, 159, 160].

One important setting where the OT problem has been explored and used is the semi-discrete optimal transport, where one distribution is continuous and the other is discrete. This arises naturally in applications such as image processing [72, 73, 85, 100], variational inference [16], and blue noise generation [53, 126], where a continuous distribution (e.g., a probability density) must be mapped to a discrete set of targets.

In many practical scenarios, the distributions of interest are either unknown (such as the underlying distribution of real-world observations) or are stored and presented by models (such as neural networks) that only allow us to sample from the distribution. In such cases, to compute the OT cost, one can draw n samples from each distribution and create empirical distributions where each sample is given a mass of $1/n$. The OT problem between these empirical distributions is equivalent to computing a minimum-cost bipartite matching between the samples, which finds a one-to-one correspondence between the samples (i.e., a matching) that minimizes the total distance of the matched pairs. As $n \to \infty$, the empirical OT cost converges to the true OT cost between them [12, 56, 60, 150] with several results showing sharp upper and lower bounds on the empirical p-Wasserstein distances [35, 59, 67, 148, 157]. See also [124] for a survey on such results. Due to these properties, the empirical Wasserstein distance has found applications in training generative adversarial networks [21, 34, 81, 105], image retrieval [129, 139], graph predictions [94, 109], and two-sample tests [54, 55, 79, 87, 137].

While classical optimal transport assumes that all mass must be transported between the

two distributions, this assumption can be overly rigid in the presence of noise and outliers. Partial optimal transport relaxes this requirement by allowing only a portion of the total mass to be matched, effectively ignoring a fraction of the data that may be corrupted or uninformative [37]. This formulation enhances the robustness of transport-based methods, particularly in real-world settings where distributions are often noisy or contaminated [119]. Partial transport has been applied to various machine learning tasks [41, 99, 120] and has also been used as a basis for designing more robust metrics for comparing distributions [117, 135].

Despite its conceptual appeal, computing the optimal transport costs remains computationally expensive. This challenge has motivated a wave of algorithmic research focused on improving scalability while preserving theoretical guarantees. In this thesis, we explore the semi-discrete optimal transport problem as well as the minimum-cost bipartite matching problem (i.e., empirical optimal transport problem) and design efficient algorithms for computing them in complete and partial settings.

1.1 Problem Definition

For points in the d-dimensional space, let $\mathrm{c} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_{\geq 0}$ denote the ground distance function, i.e., for any pair of points $a, b \in \mathbb{R}^d$, we refer to the distance between a and b by $\mathrm{c}(a, b)$. For two distributions μ and ν with support sets A and B in $\mathbb{R}^d$, we refer to the diameter of the space (i.e., the maximum pairwise distance between all pairs of points in $A \cup B$) by $C_{\max}$. In geometric settings, given integers $p, q \geq 1$, the ground distance is defined as the ℓ_p^q norm, where the ℓ_p^q norm of two points $a = (a_1, a_2, \ldots, a_d)$ and $b = (b_1, b_2, \ldots, b_d)$, denoted by $\|a - b\|_p^q$, is

$$\|a - b\|_p^q := \left(\sum_{i=1}^{d} |a_i - b_i|^p \right)^{q/p}. \tag{1.1}$$

In this thesis, all the proposed algorithms solve the OT problem in geometric settings.

Semi-discrete OT. Let μ be a continuous probability distribution (a density function) defined on a bounded set $A \subset \mathbb{R}^d$ and ν be a discrete distribution defined on a set B of n points in $\mathbb{R}^d$. A semi-discrete transport plan τ between μ and ν is a distribution $\tau\colon A \times B \to \mathbb{R}_{\geq 0}$ where for each point $a \in A$, $\sum_{b\in B} \tau(a,b) \leq \mu(a)$ and for each point $b \in B$, $\int_A \tau(a,b)\, da \leq \nu(b)$, i.e., the first and second marginals of τ are dominated by μ and ν, respectively. The cost of τ under the ground distances $\mathrm{c}(\cdot,\cdot)$ is defined as

$$w(\tau) := \sum_{b\in B} \int_A \mathrm{c}(a,b)\tau(a,b)\, da.$$

Define the mass of τ, denoted by $\mathcal{M}(\tau)$, as the amount of mass transported by the transport plan τ, i.e.,

$$\mathcal{M}(\tau) := \sum_{b\in B} \int_A \tau(a,b)\, da.$$

A transport plan τ is *complete* if τ transports all mass of μ to ν, i.e., if $\mathcal{M}(\tau) = 1$. Any complete transport plan τ with a minimum cost among all complete transport plans between μ and ν is called an *optimal transport plan*, or simply an *OT plan*. In the *semi-discrete OT problem*, the goal is to find an optimal transport plan between μ and ν.

For any value $\alpha \in [0,1)$, an *α-partial* transport plan is a transport plan τ that transports α mass, i.e., $\mathcal{M}(\tau) = \alpha$. In the *semi-discrete α-optimal partial transport* (or simply *semi-discrete α-OPT*) problem, the goal is to find a minimum-cost α-partial transport plan between μ and ν. We refer to any minimum cost α-partial transport plan as an *α-OPT plan*.

Another variant of the OT problem, called the *λ-robust OT* (or *λ-ROT*) problem, introduces a regularization via total variation distance. Given a parameter $\lambda > 0$, δ units of mass can either be transported from $a \in A$ to $b \in B$ incurring a cost of $\delta \mathrm{c}(a,b)$ or be burnt incurring

a cost of $\lambda\delta$ [117]. The λ-ROT is the minimum cost under this burn-or-transport trade-off. More precisely, we define the *λ-robust cost* of a transport plan τ, denoted by $w_\lambda^{\mathrm{ROT}}(\tau)$, as

$$w_\lambda^{\mathrm{ROT}}(\tau) := w(\tau) + \lambda(1 - \mathcal{M}(\tau)).$$

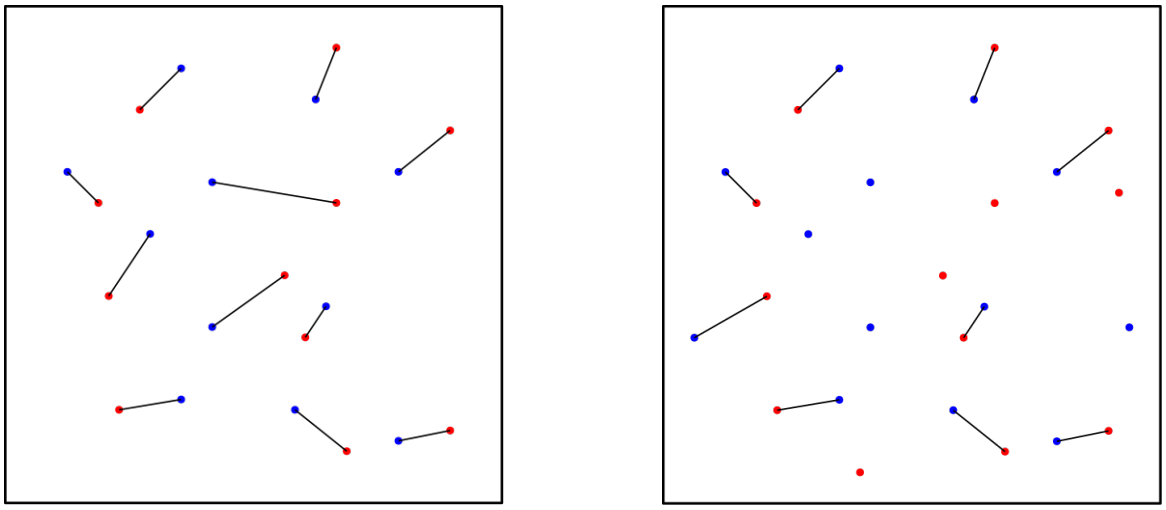
We refer to a minimum-cost transport plan under the λ-robust costs as a *λ-ROT plan*.

For any parameter $\varepsilon > 0$, a complete transport plan τ between μ and ν is called *ε-close* if the cost of τ is at most ε more than the cost of the optimal transport plan τ^*, i.e., $w(\tau) \leq w(\tau^*) + \varepsilon$. For any $\beta \geq 1$, the transport plan τ is *β-approximate* if $w(\tau) \leq \beta w(\tau^*)$. We extend these definitions to the OPT problem and for any $\alpha \in [0, 1]$, we refer to any α-partial transport plan τ as ε-close if its cost is within an ε additive error from the cost of an α-optimal partial transport plan, and refer to τ as β-approximate if its cost is at most β times the cost of an α-optimal partial transport plan.

Discrete OT. When the distribution μ is also a discrete distribution defined on a discrete support $A \subset \mathbb{R}^d$, the problem is called *discrete optimal transport* (*discrete OT*). A *discrete transport plan* τ is a function $\tau\colon A \times B \to \mathbb{R}_{\geq 0}$ such that for all $a \in A$, $\sum_{b \in B} \tau(a, b) \leq \mu(a)$ and for all $b \in B$, $\sum_{a \in A} \tau(a, b) \leq \nu(b)$. We define the cost of τ as

$$w(\tau) = \sum_{(a,b) \in A \times B} \mathrm{c}(a, b)\tau(a, b). \tag{1.2}$$

The transport plan τ is complete if τ transports all of the mass of the distributions, i.e., $\mathcal{M}(\tau) = 1$, where $\mathcal{M}(\tau) = \sum_{(a,b) \in A \times B} \tau(a, b)$ is the total mass transported by the transport plan τ. The *discrete OT problem* asks for a minimum-cost complete transport plan between μ and ν.



(a) (b)

Figure 1.1: (a) A minimum-cost perfect matching between points A (red points) and points B (blue points), and (b) a minimum-cost 9-matching between A and B.

Bipartite Matching. The special case of the discrete OT problem, where the support size of the two distributions are equal, i.e., $|A| = |B| = n$, and each point in $A \cup B$ is given a mass of $1/n$, the problem reduces to computing a minimum-cost perfect bipartite matching from the point set B to A, which is described next. A *bipartite matching* is a subset M of disjoint edges between A and B. A *perfect matching* is a matching M of size $|M| = n$, i.e., M is perfect if each point in $A \cup B$ is incident on exactly one matching edge in M (Figure 1.1(a)). Furthermore, for any value $t < n$, a matching M of size t is called a *t-matching* (Figure 1.1(b)). The cost of M, denoted by $w(M)$, is simply the sum of the costs of all edges in the matching; more formally,

$$w(M) := \sum_{(a,b)\in M} \mathrm{c}(a, b). \tag{1.3}$$

1.2 Background

1.2.1 Semi-discrete Optimal Transport Problem

For a weighted point set $P \subset \mathbb{R}^d$ with weights $y : P \to \mathbb{R}$ and a distance function $\mathrm{c} : \mathbb{R}^d \times P \to \mathbb{R}_{\geq 0}$, the (additively) *weighted distance* from a point $p \in P$ to any point $x \in \mathbb{R}^d$ is defined as

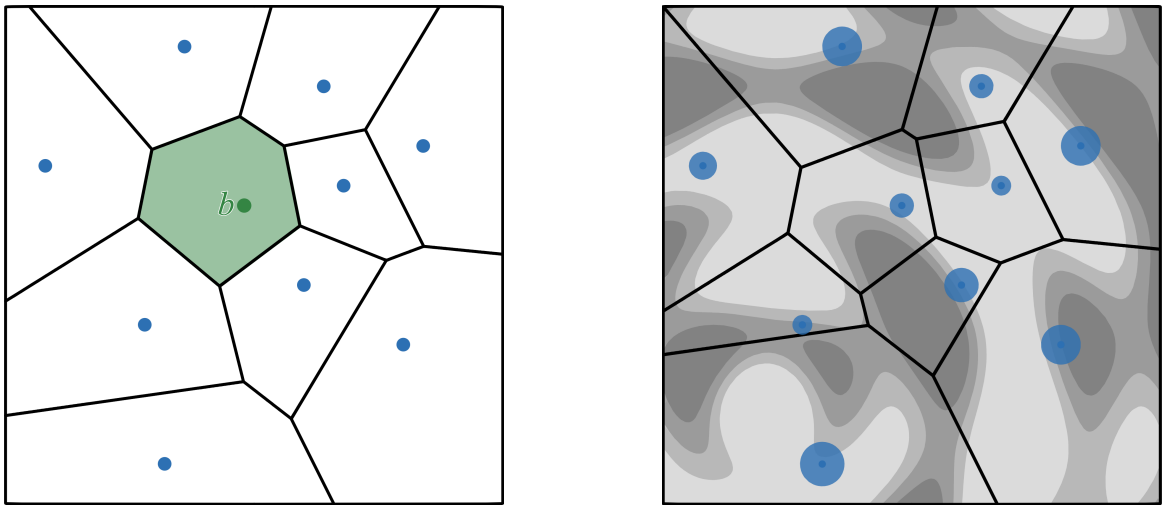
$$\mathrm{c}_y(x, p) = \mathrm{c}(x, p) - y(p). \tag{1.4}$$

For any point $x \in \mathbb{R}^d$, the weighted nearest neighbor of x in P is the point $p \in P$ with the minimum weighted distance to x. For any point $p \in P$, the *Voronoi cell* of p, denoted by $\mathrm{Vor}_y(p)$, is the locus of points in $\mathbb{R}^d$ with p as their weighted nearest neighbor; more precisely,

$$\mathrm{Vor}_y(p) := \{x \in \mathbb{R}^d \mid \mathrm{c}_y(x, p) \leq \mathrm{c}_y(x, p'), \forall p' \in P\}. \tag{1.5}$$

The *Voronoi diagram* $\mathrm{VD}_y(P)$ is the decomposition of $\mathbb{R}^d$ induced by Voronoi cells [66]. See Figure 1.2(a).

The problem of computing semi-discrete OT between μ and ν reduces to the problem of finding a set of weights $y : B \to \mathbb{R}_{\geq 0}$ so that, for any point $b \in B$, the amount of continuous mass inside the Voronoi cell of b is equal to $\nu(b)$, i.e., $\mu(\mathrm{Vor}_y(b)) = \nu(b)$. In this case, an optimal transport plan transports the mass of μ in $\mathrm{Vor}_y(b)$ to b; see [22]. One can thus define an optimal semi-discrete transport plan by describing the weights of points in B. For instance, in Figure 1.2(b), the probability mass of the continuous distribution μ is shown by the gray shades, and the discrete mass at the points of B is visualized by the blue discs centered at each point. In this case, an OT plan is computed by an optimal weight assignment to the points of B such that for each point $b \in B$, the Voronoi cell of b contains continuous mass equal to $\nu(b)$. A well-known algorithm for the semi-discrete OT problem is



(a) (b)

Figure 1.2: (a) The Voronoi cell of a point b (the green shaded region) and the Voronoi diagram of a weight point set (black lines), and (b) a semi-discrete OT plan characterized by the Voronoi diagram corresponding to an optimal weight assignment to points in B, where the discrete mass at each point is visualized by the radius of the blue disc centered at the point and the continuous mass is visualized by the gray shades.

the algorithm by Oliker and Prussner [121], which we summarize next.

Oliker-Prussner's Algorithm

The algorithm by Oliker and Prussner [121] starts with a set of weights $y(\cdot)$ for the set B and iteratively updates the weights until the corresponding Voronoi diagram satisfies the following property:

- For each point $b \in B$, the difference in the mass of μ inside the Voronoi cell of b and $\nu(b)$ is at most ε, for some tolerance parameter $\varepsilon > 0$, i.e., for each point $b \in B$, $|\mu(\mathrm{Vor}_y(b)) - \nu(b)| \leq \varepsilon$.

The weight initialization and update process is as follows. Initially, pick a point $b_0 \in B$ and assign a large weight $y(b_0) \leftarrow C_{\max}$ to b_0 and assign a zero weight $y(b) \leftarrow 0$ to all other

points $b \in B, b \neq b_0$. This choice of initial weights ensures that the Voronoi cell of b_0 initially consumes all the continuous mass. At each iteration, if there exists a point $b \in B$ with $\mu(\mathrm{Vor}_y(b)) < \nu(b) - \frac{\varepsilon}{n}$, then the algorithm increases the weight of b by the smallest amount that makes the mass of μ inside the new Voronoi cell of b to be at least $\nu(b)$.

Note that for all points $b \in B, b \neq b_0$, the amount of continuous mass inside $\mathrm{Vor}_y(b)$ is at most $\nu(b)$, and b_0 is the only point that the continuous mass inside its Voronoi cell is more than its discrete mass. Therefore, if there are no points $b \in B$ with $\mu(\mathrm{Vor}_y(b)) < \nu(b) - \frac{\varepsilon}{n}$ (which means that the algorithm has terminated), then for the point b_0, we have $\mu(\mathrm{Vor}_y(b_0)) \leq \nu(b_0) + \varepsilon$, as desired. Merigot and Thibert [111] showed that this algorithm terminates after Cn^3/ε iterations, for some constant $C \geq 0$, where each iteration requires the computation of a weighted Voronoi diagram as well as the computation of the minimum increase in the weight of a point that equalizes its discrete mass and the continuous mass inside its Voronoi cell.

Optimal Partial Transport Problem

Although the α-OPT problem has not been extensively studied in the semi-discrete setting, a recent work by Lévy [102] investigates a variant of the semi-discrete optimal transport problem where the total continuous mass in the distribution μ exceeds the total discrete mass in the distribution ν. They show that the optimal transport plan for this unbalanced OT problem can be characterized using a set of restricted Voronoi cells.

For a set of weights $y(\cdot)$ for B, the *restricted Voronoi cell* of each point $b \in B$, denoted by $\mathrm{ResVor}_y(b)$, is defined as the intersection of the Voronoi cell $\mathrm{Vor}_y(b)$ and a disc of radius $y(b)$ centered at b; more precisely,

$$\mathrm{ResVor}_y(b) := \{a \in \mathrm{Vor}_y(b) \mid \mathrm{c}(a, b) \leq y(b)\}. \tag{1.6}$$

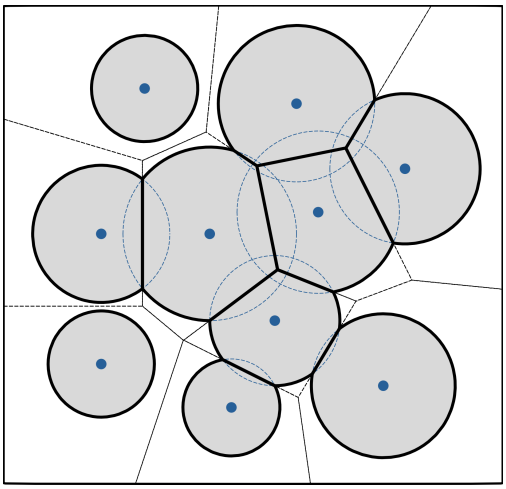


Figure 1.3: The restricted Voronoi cells (gray shaded areas) of a set of points B with weights $y(\cdot)$

See Figure 1.3. Under the assumption that the total mass of ν is less than that of μ, Lévy [102] showed that the optimal transport plan is determined by a set of weights $y(\cdot)$ such that the continuous mass within the restricted Voronoi cell of each $b \in B$ is equal to its discrete mass, i.e., $\mu(\text{ResVor}_y(b)) = \nu(b)$. Once such weights are computed, the OT plan transports the mass associated with each point $b \in B$ to the continuous mass within its restricted Voronoi cell.

1.2.2 Geometric Bipartite Matching Problem

Let G be a complete bipartite graph over point sets A and B. A *matching* M is a subset of disjoint edges of G. Any edge $(a, b) \in M$ is called a *matching edge* and any edge (a, b) that is not a matching edge is called a *non-matching edge*. The *size* of a matching is the number of edges it contains. For any matching M, any point $u \in A \cup B$ is *free* if no matching edges are incident on u. Let A_F^M and B_F^M be the subset of free vertices in A and B, respectively; we drop the superscript M when the matching is clear from the context.

For any matching M between A and B, an *alternating path* is a simple path whose edges alternate between matching and non-matching edges. Any alternating path whose endpoints are free is called an *augmenting path*. For any augmenting path P, we *augment* M along P by setting $M \leftarrow M \oplus P$, i.e., each matching (resp. non-matching) edge of P will be a non-matching (resp. matching) edge after augmentation. Note that augmenting a matching along an augmenting path increases the number of matching edges by one. The *net-cost* $\phi(P)$ of an augmenting path P is defined as

$$\phi(P) := \sum_{(a,b)\in P\setminus M} \mathrm{c}(a,b) - \sum_{(a,b)\in P\cap M} \mathrm{c}(a,b). \tag{1.7}$$

In words, for any augmenting path P, the net-cost of P measures the increase in the matching cost when we augment M along P. The well-known Hungarian algorithm [97] computes a minimum-cost perfect matching as follows: Initialize an empty matching $M \leftarrow \emptyset$. Iteratively, compute an augmenting path P with the minimum net-cost and augment the matching M along P. It is shown that after n iterations (resulting in a matching of size n), the matching M is a minimum-cost perfect matching. To assist in computing the minimum net-cost augmenting paths, the Hungarian algorithm uses a primal-dual framework, which is described next.

Primal-Dual Framework

The geometric bipartite matching problem admits a simple integer program (IP) formulation that assigns a value $m(a,b) \in \{0,1\}$ to each pair of points $(a,b) \in A \times B$, indicating whether (a,b) is a matching edge ($m(a,b) = 1$) or a non-matching edge ($m(a,b) = 0$), and aims to minimize the matching cost. More formally, the IP formulation for the geometric bipartite

matching problem is as follows:

$$\begin{aligned}
&\min_{m} \sum_{(a,b)\in A\times B} m(a,b)\mathrm{c}(a,b) \\
&\text{s.t.} \sum_{b'\in B} m(a,b') = 1, \qquad \forall a \in A, \qquad (1.8)\\
&\qquad \sum_{a'\in A} m(a',b) = 1, \qquad \forall b \in B, \qquad (1.9)\\
&\qquad m(a,b) \in \{0,1\}, \qquad \forall (a,b) \in A \times B.
\end{aligned}$$

In this IP formulation, Equations (1.8) and (1.9) ensure that exactly one matching edge (i.e., one edge (a,b) with $m(a,b) = 1$) is incident on any point $a \in A$ and $b \in B$, respectively. The dual of this primal IP formulation aims to find a *dual weight* $y(u)$ for each point $u \in A \cup B$ with the goal to maximize the objective function as described below:

$$\begin{aligned}
&\max_{y} \sum_{b\in B} y(b) - \sum_{a\in A} y(a) \\
&\text{s.t. } y(b) - y(a) \le \mathrm{c}(a,b), \qquad \forall (a,b) \in A \times B, \qquad (1.10)\\
&\qquad y(u) \ge 0, \qquad \forall u \in A \cup B.
\end{aligned}$$

From the complementary slackness property of primal-dual formulations, for any matching edge (a,b) (i.e., any edge (a,b) such that the optimal solution to the primal formulation assigns $m(a,b) = 1$), we expect to have an equality in Equation (1.10), i.e., $y(b) - y(a) = \mathrm{c}(a,b)$. We formalize this argument next.

Dual feasibility. Suppose M is a t-matching between A and B, and let $y(\cdot)$ denote a set of weights for $A \cup B$. The pair $M, y(\cdot)$ is *feasible* if:

$$y(b) - y(a) \leq \mathrm{c}(a,b) \qquad \forall (a,b) \in A \times B, \tag{1.11}$$

$$y(b) - y(a) = \mathrm{c}(a,b) \qquad \forall (a,b) \in M. \tag{1.12}$$

These are the classical feasibility conditions for the bipartite matching problem. Kuhn [97] showed that for any perfect matching M and dual weights $y(\cdot)$ for $A \cup B$ where $M, y(\cdot)$ is feasible, the matching M is a minimum-cost perfect matching. For completeness, we formalize this in the following lemma and provide its proof in the appendix.

Lemma 1.1. *Suppose M is a perfect matching and $y(\cdot)$ is a set of dual weights for $A \cup B$ where $M, y(\cdot)$ is feasible. Then, M is a minimum-cost perfect matching.*

Building on the concepts of the complementary slackness property of primal-dual approaches, we next define a slack for any edge $(a,b) \in A \times B$. For any feasible matching $M, y(\cdot)$, define the *slack* $s(a,b)$ of (a,b) as

$$s(a,b) := \mathrm{c}(a,b) - y(b) + y(a). \tag{1.13}$$

The slack values measure how far the feasibility constraint (Equation (1.11)) is from becoming an equality. Note that from Equation (1.11), all slack values are non-negative, and from Equation (1.12), for any matching edge $(a,b) \in M$, $s(a,b) = 0$.

Lemma 1.2. *For any feasible matching $M, y(\cdot)$ and any augmenting path P between a free point $a \in A$ and a free point $b \in B$, $\phi(P) = y(b) - y(a) + \sum_{(a',b') \in P} s(a',b')$.*

For a feasible matching $M, y(\cdot)$, any edge (a,b) with a zero slack is called *admissible*. We extend the definition of admissibility to augmenting paths and refer to any augmenting path

P where all edges of P are admissible by an *admissible* path. Augmenting M along an admissible augmenting path P does not violate the feasibility conditions.

Hungarian Algorithm

The Hungarian algorithm is a primal-dual combinatorial algorithm that computes a minimum-cost perfect matching as follows. The algorithm starts with an empty matching M and initializes all dual weights $y(v) \leftarrow 0$ for all $v \in A \cup B$. It performs a sequence of Hungarian search procedures (described below), each of which updates the dual weights, computes an admissible augmenting path, and augments the matching while maintaining the dual feasibility conditions. Each Hungarian search increases the size of M by 1, and after n such steps, we obtain a minimum-cost perfect matching.

During the execution of the Hungarian algorithm, in addition to the feasibility conditions (1.11) and (1.12), the algorithm also ensures that the following condition holds.

(H) For any free point $b \in B_F$, $y(b) = \max_{b' \in B} y(b')$ and for any free point $a \in A_F$, $y(a) = 0$.

Assuming that the set of dual weights $y(\cdot)$ and the matching M satisfy condition (H), from Lemma 1.2, any minimum net-cost augmenting path would also be a minimum slack path. This is because all free points of A have a zero dual weight and all free points of B have equal dual weights. Based on this observation, we next describe the Hungarian search procedure.

Hungarian search procedure. The Hungarian search procedure receives, as input, a feasible matching M and $y(\cdot)$ satisfying condition (H). It first updates the dual weights while maintaining the feasibility conditions and condition (H). The dual updates guarantee that there exist admissible augmenting paths with respect to the updated dual weights. The

procedure then computes an admissible augmenting path P and augments the matching along the path P. The procedure finally returns the updated matching and dual weights.

To compute admissible augmenting paths, the procedure constructs a *residual graph* $\mathcal{G}$. The vertex set of the residual graph consists of a source vertex s in addition to the points in $A \cup B$. For any matching edge $(a, b) \in M$, there is a directed edge from a to b with $s(a, b)$ as its weight, and for any non-matching edge $(a, b) \notin M$, there is a directed edge from b to a with $s(a, b)$ as its weight in $\mathcal{G}$. Finally, there is an edge from s to all free points $b \in B_F$ with a zero weight. The construction of the residual graph ensures that any directed path P in $\mathcal{G}$ is an alternating path and its cost is the sum of the slacks of the edges along the path.

The Hungarian search procedure computes the shortest path distance from s to all points in $A \cup B$ in the residual graph. For any $v \in A \cup B$, let P_v denote the shortest path from s to v, and let κ_v be the total weight of P_v. Define $a^* := \arg\min_{a \in A_F} \kappa_a$ as the free point of A with the minimum distance from the source, and let $\kappa^* := \kappa_{a^*}$. For any point $v \in A \cup B$, if $\kappa_v < \kappa^*$, set $y(v) \leftarrow y(v) - \kappa_v + \kappa^*$. Let P be the path obtained by removing s from P_{a^*}. Augment M along P. This completes the description of the Hungarian search procedure.

Note that all steps in the Hungarian search except computing the shortest path distances take $O(n)$ time. Suppose we have access to a Dynamic Weighted Bichromatic Closest Pair Data Structure (DWBCP-DS) that stores two sets of points A and B with a weight $y_{\text{bcp}}(v)$ assigned to each point $v \in A \cup B$ and supports (i) inserting and removing points, (ii) updating weights, and (iii) returning the weighted bichromatic closest pair defined as

$$(a, b) := \arg \min_{a' \in A, b' \in B} \mathrm{c}(a, b) - y_{\text{bcp}}(b) + y_{\text{bcp}}(a).$$

Vaidya [152] showed that using a DWBCP-DS with a query/update time of $\Phi(n)$, one can efficiently implement the shortest path computation of the Hungarian search procedure in

$O(n\Phi(n))$ time. We describe this efficient implementation next.

Efficient Dijkstra's shortest path algorithm. Recollect that Dijkstra's algorithm incrementally builds a shortest path tree rooted at the source s. Let $U \subseteq B$ be the points already added to the shortest path tree, and let $V \subseteq A$ be those not yet added. Additionally, for each point $u \in A \cup B$, let κ_u denote the estimated distance of u from the source and let $y_{\text{bcp}}(u)$ denote an adjusted weight for u that is used as the weight of u in the **DWBCP-DS**. Initially, $U = B_F$ and $V = A$. For each $b \in U$, set $\kappa_b = 0$ and $y_{\text{bcp}}(b) = y(b) - \kappa_b$, and for each $a \in V$, set $\kappa_a = 0$ and $y_{\text{bcp}}(a) = y(a)$. Build a **DWBCP-DS** on (U, V) using these weights. In each iteration of the algorithm, query the data structure to find the weighted bichromatic closest pair (a, b). This is equivalent to

$$(a, b) = \arg \min_{a' \in V, b' \in U} \{\mathrm{c}(a', b') - y_{\text{bcp}}(b') + y_{\text{bcp}}(a')\} = \arg \min_{a' \in V, b' \in U} \{\kappa_{b'} + s(a', b')\}.$$

Thus, the edge (b, a) that the standard Dijkstra's shortest path algorithm would add next to the shortest path tree would be exactly the bichromatic closest pair between U and V and can be found in $\Phi(n)$ time. Once the pair (a, b) is selected, the algorithm will remove a from V, add it to the shortest path tree, and set $\kappa_a = \kappa_b + s(a, b)$. If a is free, we have found an augmenting path. Otherwise, let b' be the vertex matched with a in M. Set $\kappa_{b'} = \kappa_a$ and $y_{\text{bcp}}(b') = y(b') - \kappa_{b'}$, and add b' to U.

Each iteration of this implementation of Dijkstra's algorithm deletes one vertex from V and adds at most one vertex to U. Hence, the algorithm terminates with an augmenting path in at most n iterations, leading to an $O(n\Phi(n))$ overall running time.

1.3 Related Work

Semi-discrete OT. Given the inherent complexities of computing the exact semi-discrete OT between distributions, researchers have focused on developing approximation algorithms for this problem, and there has been significant work on designing approximation algorithms for computing an ε-close semi-discrete OT plan. For low-dimensional settings, there are algorithms to compute an ε-close semi-discrete transport plan using numerical solvers [22, 25, 38, 53, 92, 93, 103, 121], entropic regularization [15, 26, 77], and multiscale approaches [100, 110]. The execution time of all these algorithms is exponential in both d and $\log 1/\varepsilon$. The previous best-known algorithm for computing an ε-close semi-discrete OT is presented in the work by Kitagawa [92], which has an execution time $C_{\max} n^{\Omega(d)}/\varepsilon$, where $C_{\max}$ is the diameter of the space. However, the convergence of the majority of these algorithms relies on the smoothness of the continuous distribution μ (i.e., the amount of continuous mass inside any small region is bounded). For instance, under this smoothness assumption, a notable numerical algorithm by Oliker and Prussner [121] executes $\text{poly}(n, 1/\varepsilon)$ iterations, where each iteration computes the mass of the continuous distribution inside the Voronoi cells, 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.

Another approach for solving the semi-discrete OT problem involves drawing samples from the continuous distribution and transforming the semi-discrete OT problem into a discrete one; however, due to sampling errors, this approach provides an additive approximation. Other researchers presented additive approximation algorithms that apply the averaged stochastic gradient descent framework to the discrete instance obtained by drawing samples from the continuous distribution [77, 100]. Their method achieves a convergence rate

of $O(1/\sqrt{k})$, where k is the number of iterations. A notable contribution by Van Kreveld, Staals, Vaxman, and Vermeulen [153] offers a $(1+\varepsilon)$-approximation algorithm for semi-discrete OT in cases where the continuous distribution is a uniform distribution across a collection of simple geometric objects (such as segments). This approach also involves sampling points from the continuous distribution and executing a $(1+\varepsilon)$-approximation discrete OT algorithm. The runtime for this method is approximately $n^2\varepsilon^{-O(d)}\text{poly}\log(n)$. More recently, Agarwal, Raghvendra, Shirzadian, and Yao [9] presented a near-linear time algorithm for computing a $(1+\varepsilon)$-approximate semi-discrete OT plan using the multiplicative weights update method.

Chapel, Alaya, and Gasso [37] were the first to show that the α-OPT can be used to identify and remove outliers, especially when the proportion of outliers is known in advance. See also [33, 65, 119] for results on the structure and properties of optimal partial transport. Although the α-OPT distance is not a metric, Raghvendra, Shirzadian, and Zhang [135] introduced an α-OPT based metric, called the RPW distance, that enjoys improved statistical robustness compared to the p-Wasserstein distance.

Mukherjee, Guha, Solomon, Sun, and Yurochkin [117] introduced the λ-ROT (originally referred to as ROBOT distance) in the discrete settings. Additionally, Ma, Liu, La Vecchia, and Lerasle [107] established mathematical properties of λ-ROT distance as well as designed algorithms for it. More recently, Lévy [102] characterized the optimal solutions to an unbalanced version of the semi-discrete optimal transport problem, where the total mass of the continuous distribution is greater than the total discrete mass.

Discrete OT and bipartite matching. The discrete OT problem under any metric can be modeled and solved as an uncapacitated minimum-cost flow problem [123, 144]. Extensive research has focused on developing near-linear time algorithms that compute a $(1+\varepsilon)$-

approximation for the optimal transport problem [23, 68, 69, 91, 147]; the running times of all these algorithms are exponentially dependent on the dimension, which makes them less usable for the high-dimensional instances that arise in machine learning applications. Due to the lack of fast (relative) approximation algorithms in high dimensions, researchers have designed algorithms with additive guarantees on approximation [7, 13, 14, 29, 50, 61, 80, 98, 127].

For a graph with n nodes and m edges, the classical Hungarian algorithm finds a minimum-cost t-matching for all values of t, where $0 \leq t \leq n$ [97, 125]. The total execution time of the Hungarian algorithm is $O(nm + n^2 \log n)$, or $O(n^3)$ when $m = \Theta(n^2)$. Despite substantial efforts, this remains the most efficient algorithm for the problem. Notable exceptions include specialized cases, such as graphs with small vertex separators [104] or integer edge weights [39, 70, 71]. However, the execution time of these algorithms has a dependence on $\log C_{\max}$, making them weakly polynomial.

In geometric settings, Vaidya [152] showed that using a **DWBCP-DS** with a query/update time of $\Phi(n)$, one can improve the running time of each iteration of the Hungarian algorithm to $\tilde{O}(n\Phi(n))$. Thus, the minimum-cost t-matching can be computed in $O(n^2\Phi(n))$ time, which is sub-cubic in n provided $\Phi(n)$ is sub-linear. For instance, for the ℓ_1 norm and d dimensions, $\Phi(n) = O(\log^d n)$ [152] and for the ℓ_2 norm and 2 dimensions, $\Phi(n) = \log^{O(1)} n$ [90]. In these cases, the Hungarian algorithm can be implemented in near-quadratic time.

Designing algorithms that compute a minimum-cost matching using sub-quadratic **DWBCP-DS** queries remains a central open problem in computational geometry. There are two notable exceptions. First, for points with integer coordinates and the ℓ_1 and ℓ_∞ norms, Sharathkumar and Agarwal [146] adapted a cost-scaling algorithm [70] and presented an algorithm that executes in $\tilde{O}(n^{3/2}\Phi(n))$ time. Second, Sharathkumar [145] extended this result to 2-dimensional point sets with integer coordinates and the ℓ_2 costs and presented a $\tilde{O}(n^{3/2} \log \Delta)$ time exact algorithm for the bipartite matching problem under Euclidean

distances. Their result, however, relies on planarity as well as the edge costs being the square root of integers and does not extend to d-dimensional points with real-valued coordinates.

We also highlight the substantial body of work on designing near-linear time randomized approximation algorithms for optimal transport and bipartite matching problems [1, 20, 23, 69, 91, 133, 146, 147, 162]. The running time of these algorithms is $n(\varepsilon^{-1}\log n)^{O(d)}$. Agarwal, Chang, Raghvendra, and Xiao [5] presented an $n(\varepsilon^{-1}\log n)^{O(d)}$-time deterministic algorithm for computing a $(1+\varepsilon)$-approximate bipartite matching in $\mathbb{R}^d$ and Agarwal, Raghvendra, Shirzadian, and Sowle [6] presented a randomized $(1+\varepsilon)$-approximation algorithm for bipartite matching with run time $n\log^4 n(\varepsilon^{-1}\log\log n)^{O(d)}$. Additionally, Fox and Lu [68] proposed a deterministic algorithm for $(1+\varepsilon)$-approximate OT with run time of $O(n\varepsilon^{-(d+2)}\log^5 n\log\log n)$.

1.4 Algorithmic Challenges of Optimal Transport

The computational cost of solving OT problems can pose challenges, particularly when dealing with continuous distributions. In this section, we delve into the algorithmic difficulties that researchers encounter when computing OT costs.

1.4.1 Semi-Discrete Optimal Transport

For any $\varepsilon > 0$, numerous iterative methods have been proposed to compute an ε-close semi-discrete OT plan, where the running times of all such algorithms have a polynomial dependence on $1/\varepsilon$; furthermore, these algorithms only approximate the cost and do not necessarily provide any guarantees for the optimal transport plan or the optimal mapping of the discrete mass to the continuous mass.

Merigot and Thibert [111] showed that the algorithm by Oliker and Prussner [121] converges in $O(n^3/\varepsilon)$ iterations and conjectured that one might be able to reduce the dependence of the running time of the algorithm on ε from $(1/\varepsilon)^{\Omega(1)}$ to $\log^{O(1)} \varepsilon^{-1}$ using a bit-scaling framework [111, Remark 24]. The following problem formalizes this conjecture.

Problem 1.3. Given a continuous distribution μ defined over a bounded set $A \subset \mathbb{R}^d$, a discrete distribution ν defined over a support $B \subset \mathbb{R}^d$ of n points, and a parameter $\varepsilon > 0$, is there an algorithm for computing an ε-close semi-discrete transport plan between μ and ν in $\text{poly}(n^{O(d)}, \log \frac{C_{\max}}{\varepsilon})$ time under any ℓ_p norm?

Roughly speaking, computing a highly accurate semi-discrete OT plan requires approximating the weighted Voronoi diagram accurately, which has a complexity of $n^{\Theta(d)}$ for d-dimensional spaces. Additionally, all existing algorithms, including the ones that rely on first and second order numerical solvers, require the computation of the mass of the continuous distribution inside a polytope in d-dimensional space. Therefore, all such methods, including an algorithm for Problem 1.3 are impractical in high dimensions.

On the other hand, the computation of the semi-discrete OT in low-dimensional settings has been used in numerous machine learning methods, such as blue noise generation [53], shape reconstruction [52, 112], and caustic design [114, 143], where, in these applications, models are trained on lower-dimensional spaces such as two-dimensional images. Therefore, an interesting problem would be the development of a fast and implementable ε-close semi-discrete OT algorithm designed for two-dimensional distributions.

Problem 1.4. Given a continuous distribution μ defined over a bounded set $A \subset \mathbb{R}^2$ in 2 dimensions, a discrete distribution ν defined over a support $B \subset \mathbb{R}^2$ of n points, and a parameter $\varepsilon > 0$, is there an algorithm for computing an ε-close semi-discrete transport plan between μ and ν in $\tilde{O}(n^4 \log \frac{C_{\max}}{\varepsilon})$ time under any ℓ_p norm?

The next challenge is in characterizing the optimal solutions to the partial and robust variants of the OT problem in the semi-discrete settings and extending the existing frameworks and algorithms for computing complete transport plans in semi-discrete settings to these variants. Previously, Lévy [102] explored an unbalanced version of the semi-discrete optimal transport problem, where the discrete distribution contains more mass than the continuous mass, and the goal is to transport all the discrete mass from the points of B to the continuous mass in A and characterized the optimal solution with a set of restricted Voronoi cells. An important open question is to extend this characterization to the robust optimal transport problem and extend our algorithmic framework for the semi-discrete (complete) OT problem to the semi-discrete robust optimal transport problem.

Problem 1.5. Given a continuous distribution μ defined over a bounded set $A \subset \mathbb{R}^2$, a discrete distribution ν defined over a support $B \subset \mathbb{R}^2$ of n points, and parameters $\lambda \in [0, C_{\max}]$ and $\varepsilon > 0$, is there an algorithm for computing an ε-close semi-discrete λ-robust optimal transport plan between μ and ν in $\tilde{O}(n^4 \log \frac{C_{\max}}{\varepsilon})$ time under any ℓ_p norm?

1.4.2 Euclidean Bipartite Matching

The best known algorithm for computing an exact minimum-cost bipartite matching in geometric settings is the Hungarian algorithm and its fast implementations, which run in $O(n^2\Phi(n))$ time; recall that $\Phi(n)$ is the query/update time of a **DWBCP-DS**. The only known improvement is the algorithm by Sharathkumar [145], which, for planar points with integer coordinates bounded by Γ, computes the exact 1-Wasserstein distance in $\tilde{O}(n^{3/2} \log \Gamma)$ time. The design of a faster algorithm to compute the pth power Euclidean bipartite matching for d-dimensional point sets remains a major open problem.

Many machine learning applications, such as those in detecting distributional shifts [128] and

benchmarking algorithms [31], compute the empirical p-Wasserstein distance between two distributions to identify whether the two distributions are the same or not. In such cases, it is important to have a fast algorithm for computing the exact minimum-cost bipartite matching, even if the algorithm runs fast only when the input points are i.i.d samples from the same distribution.

Problem 1.6. Given two sets of i.i.d samples $A, B \subset \mathbb{R}^d$ from the same distribution μ, each of size n, and any $p \in [1, \infty)$, is there an algorithm that computes the pth power Euclidean bipartite matching between A and B in a time that is sub-quadratic in n, assuming that the algorithm does not know the distribution μ?

In addition to the perfect matching problem, computing (partial) t-matchings efficiently for any value of t is an important problem. Recall that for a graph with n vertices and m edges, the Hungarian algorithm computes minimum-cost t-matchings for all $1 \leq t \leq n$ in $O(t(m + n \log n))$ time. Therefore, for dense graphs (i.e., $m = \Theta(n^2)$) and larger values of t (i.e., $t = \Theta(n)$), the Hungarian algorithm computes a t-matching in $O(n^3)$ time. In geometric settings, using a DWBCP-DS with a query/update time of $\Phi(n)$, one can efficiently implement the Hungarian algorithm in $\tilde{O}(n^2\Phi(n))$ time. Although improving the running time of the Hungarian algorithm for general dense bipartite graphs appears elusive, there may be instances where structural properties of the graph allow for computing a minimum-cost t-matching using a sub-quadratic number of queries to a DWBCP-DS.

Recently, Sowle [151] showed that the offline version of the well-known k-server problem reduces to computing a minimum-cost perfect bipartite matching on a graph. In this problem, given the initial locations of k servers and a sequence of n requests arriving one at a time, the goal is to assign servers to requests in arrival order while minimizing the total travel cost. This reduction can be naturally extended to settings where the initial server locations are not explicitly specified. We next briefly introduce this variant, referred to as the k-sequence

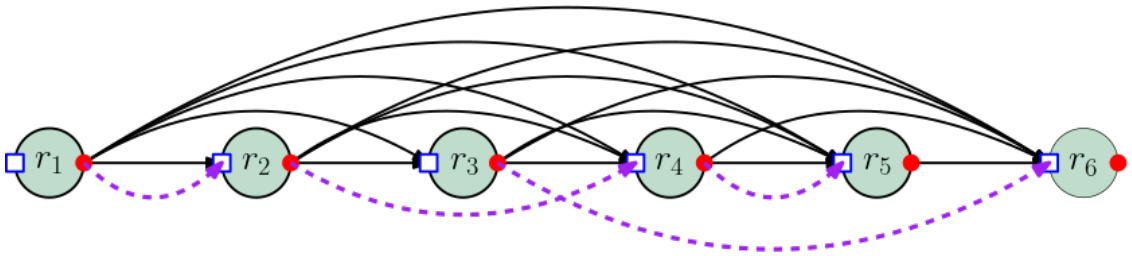


Figure 1.4: The graph $\mathcal{G}_\sigma$ constructed for $\sigma = \langle r_1, r_2, \ldots, r_6 \rangle$. The vertex set A (red disks) and B (blue squares) represent the exit and entry gates of each request, and the purple dashed lines show an $(n-2)$-partial matching on $\mathcal{G}_\sigma$ representing a 2-partitioning $\langle r_1, r_2, r_4, r_5 \rangle$ and $\langle r_3, r_6 \rangle$.

partitioning problem, which omits the assumption of known initial positions.

k-Sequence Partitioning Problem. Consider a sequence of requests $\varsigma = \langle r_1, \ldots, r_m \rangle$ in a metric space equipped with the cost function $\mathrm{c}(\cdot, \cdot)$. The cost of a single server *servicing* the requests in ς is the sum of the distances between every consecutive pair of points in the sequence, i.e., $w(\varsigma) = \sum_{i=1}^{m-1} \mathrm{c}(r_i, r_{i+1})$. Given a sequence $\sigma = \langle r_1, \ldots r_n \rangle$ of n requests and an integer $1 \leq k \leq n$, the k-sequence partitioning problem (or simply the k-SP problem) requires partitioning the requests in σ into k subsequences $\varsigma_1, \ldots, \varsigma_k$ so that $\sum_{i=1}^{k} w(\varsigma_i)$ is minimized. The optimal solution for the k-sequence partitioning problem is the cheapest way for k servers to service all of the requests in σ.

Given an input sequence σ of requests to the k-SP problem, we construct a bipartite graph G_σ with a vertex set $A \cup B$ and a set of edges as follows. *Vertex Set:* For each request r_i, we create a vertex b_i (resp. a_i) in B (resp. A) and designate it as the *entry* (resp. *exit*) gate for request r_i. *Edge Set:* The exit gate a_i of request r_i is connected to the entry gate b_j of every subsequent request r_j with $j > i$ with an edge. The cost of this edge is $\mathrm{c}(a_i, b_j) = \|r_i - r_j\|_p$. It is easy to see that a minimum-cost $(n-k)$-matching M in G_σ can be used to find an optimal solution to the k-SP problem. See Figure 1.4.

Problem 1.7. Given a sequence of n requests σ in two dimensions, an integer $0 \leq k \leq n$,

and any $p \in [1, \infty)$, is there an algorithm with a sub-quadratic running time that computes an optimal solution to the k-SP problem?

In this thesis, we present algorithms for Problems 1.3–1.7.

Chapter 2

Our Results

In this section, we summarize our results and provide a high-level overview of our techniques.

2.1 Semi-Discrete Optimal Transport Problem

2.1.1 A Highly-Accurate Algorithm for the Semi-Discrete OT Problem

The first result of this thesis is a cost-scaling algorithm that computes an ε-close transport plan for a semi-discrete OT instance in $\mathrm{Q}n^{O(d)}\log(C_{\max}/\varepsilon)$ time, assuming that we have access to an ORACLE that, given a constant complexity region φ, returns $\mu(\varphi)$ in Q time; in computational geometry, a constant complexity region refers to a geometric region whose boundary can be described using a constant number of simple components, regardless of the size of the overall input. This result is published as part of our work in [9].

Theorem 2.1. *Let μ be a continuous distribution defined on a bounded set $A \subset \mathbb{R}^d$ for some fixed $d \geq 1$, ν a discrete distribution with a support $B \subset \mathbb{R}^d$ of size n, and $\varepsilon > 0$ a parameter. Suppose there exists an* ORACLE *that, given a constant complexity region φ, returns $\mu(\varphi)$ in* Q *time. Then, an ε-close transport plan can be computed in $\mathrm{Q}n^{O(d)}\log(\frac{C_{\max}}{\varepsilon})$ time, where $C_{\max}$ is the diameter of $A \cup B$.*

To the best of our knowledge, our algorithm is the first one to compute an ε-close transport plan in time that is polynomial in both n and $\log(\varepsilon^{-1})$. Earlier algorithms had an $\varepsilon^{-O(1)}$ factor in the run time. Our result answers Problem 1.3 in the affirmative. Our algorithm not only computes an ε-close transport plan, it also finds the optimal dual weights within an additive error of ε, i.e., it computes optimal dual weights up to $O(\log \varepsilon^{-1})$ bits of accuracy. Our algorithm works for several important distances, including the ℓ_p-norm and the squared Euclidean distance. The previous best-known algorithm by Kitagawa [92] for the semi-discrete OT has an execution time $n^{\Omega(d)} C_{\max}/\varepsilon$; furthermore, their algorithm only approximates the cost and does not necessarily provide any guarantees for the transport plan or the dual weights of B.

For each scale δ, our algorithm starts with a set of dual weights assigned to B and constructs an instance of discrete OT by using the arrangement of $4n+1$ shifts of the Voronoi cell of each point in B. This discrete instance, which is of size $n^{O(d)}$, is then solved using a primal-dual solver. The optimal dual weights for this discrete instance are then used to refine the dual weights of B. These refined dual weights act as the starting dual weights for the next scale $\delta/2$. Starting with $\delta = C_{\max}$, our algorithm executes $O(\log(C_{\max}/\varepsilon))$ scales and stops when $\delta \leq \varepsilon$. In order to show that the semi-discrete transport plan computed in scale δ is δ-close, we introduce a set of exponentially many relaxed feasibility constraints and show that any transport plan that satisfies these is a δ-close transport plan. We then show that, in scale δ, the semi-discrete OT plan along with the dual weights computed by our polynomial time algorithm satisfy all of these exponentially many constraints and, therefore, is δ-close.

2.1.2 A Combinatorial Algorithm for the Semi-Discrete OT Problem

In our second result, which is presented in [8], 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 $O(n^4 \log n \log \frac{C_{\max}}{\varepsilon})$ time in the worst case, assuming we have access to an oracle that computes the continuous mass inside a constant complexity region in constant time.

Theorem 2.2. *Let μ be a continuous probability distribution defined on a bounded 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 there exists an* ORACLE *that, given a triangle φ, returns the mass of μ inside φ in Q time. Then, an ε-close OT plan between μ and ν can be computed in $O(n^3(\mathrm{Q} + n \log n) \log \frac{C_{\max}}{\varepsilon})$ time, where $C_{\max}$ is the diameter of $A \cup B$.*

Similar to the algorithm from Theorem 2.1, this algorithm is also based on a cost-scaling approach and executes $O(\log \frac{C_{\max}}{\varepsilon})$ scales. However, the algorithm within each scale is different. 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. To efficiently find 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 4.1), as well as our algorithm (in particular Sections 4.2.1 and 4.2.3) contain a number of novel ideas, carefully designed to address vari-

ous 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 [115], but we are not aware of any previous work that extends a combinatorial discrete OT framework to continuous space.

Our algorithm extends to any $p \geq 1$ and any dimension $d \geq 2$ in a straightforward way. For $d > 2$, the algorithm in Theorem 2.2 has an execution time of $O(n^{d+1}(\mathrm{Q} + n \log n) \log \frac{C_{\max}}{\varepsilon})$. Note that the runtime of the algorithm by Oliker and Prussner [121] 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 our algorithm for Theorem 2.1, our transport plan approximates an optimal weighted Voronoi diagram within a small additive factor.

2.1.3 A Efficient Algorithm for the Semi-Discrete Robust Optimal Transport Problem

In our third result, we investigate the semi-discrete λ-ROT problem. We first show that the optimal solution for the semi-discrete λ-ROT problem can be described using restricted Voronoi cells. Recall that the $\mathrm{ResVor}_y(b)$ of each point $b \in B$ with respect to weights $y(\cdot)$ is defined as the intersection of the Voronoi cell of b with a ball centered at b of radius $y(b)$. We show that in the optimal solution to the semi-discrete λ-ROT problem, all weights $y(b)$ are bounded by λ, and either (i) the mass of b in ν equals the continuous mass inside $\mathrm{ResVor}_y(b)$, or (ii) the mass at b is larger and $y(b) = \lambda$. We also provide a similar characterization for the semi-discrete α-OPT problem. To our knowledge, this is the first result to give such a characterization of partial or robust OT in the semi-discrete settings. This result has been submitted for publication [11].

We also extend our combinatorial algorithm from Theorem 2.2 to the λ-ROT problem and design a cost-scaling algorithm that computes an ε-close λ-robust transport plan in $n^4(\log n)(\log(\frac{C_{\max}}{\varepsilon}))$ time, assuming we have access to an oracle that computes the continuous mass inside a constant complexity region in constant time.

Theorem 2.3. *Let μ be a continuous distribution with bounded support $A \subset \mathbb{R}^d$, ν be a discrete distribution with support of n points $B \subseteq \mathbb{R}^d$, and $\alpha > 0$ be a parameter. Suppose there exists an* ORACLE *that returns the mass of μ inside a constant complexity region in Q time. Then, there is an algorithm that computes an ε-close λ-robust optimal transport plan in $O(n^3(\mathrm{Q} + n\log n)\log(\frac{C_{\max}}{\varepsilon}))$ time, where $C_{\max}$ is the diameter of $A \cup B$.*

Extending cost-scaling approaches to solve optimal partial transport is challenging, even in the fully discrete case [138], and these challenges only increase in the semi-discrete setting. We detail this challenge as well as our approach for resolving them in Section 5.3.

Our cost-scaling algorithm for Theorem 2.3 maintains an approximate restricted Voronoi diagram, which is successively refined at each scale. The algorithm maintains the invariants that the untransported mass of the continuous distribution lies outside the approximate Voronoi cells. Ensuring this invariant as we move from one scale to the next is challenging and requires new ideas. Our algorithm runs in $\log(C_{\max}/\varepsilon)$ scales and computes a solution to the λ-ROT problem with $\log(1/\varepsilon)$ bits of precision.

2.1.4 Our Framework

Consider a set of weights $y(\cdot)$ for the points in B. Recall that the weighted distance between each point $b \in B$ and any point $x \in \mathbb{R}^d$ is defined as $\mathrm{c}_y(x,b) = \mathrm{c}(x,b) - y(b)$, and the Voronoi cell of each point $b \in B$ is defined as $\mathrm{Vor}_y(b) := \{x \in \mathbb{R}^d \mid \mathrm{c}_y(x,b) \leq \mathrm{c}_y(x,b'), \forall b' \in B\}$.

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.

$$y_b^\delta(b') = \begin{cases} y(b') + \delta, & b' = b, \\ y(b'), & b' \neq b. \end{cases} \tag{2.1}$$

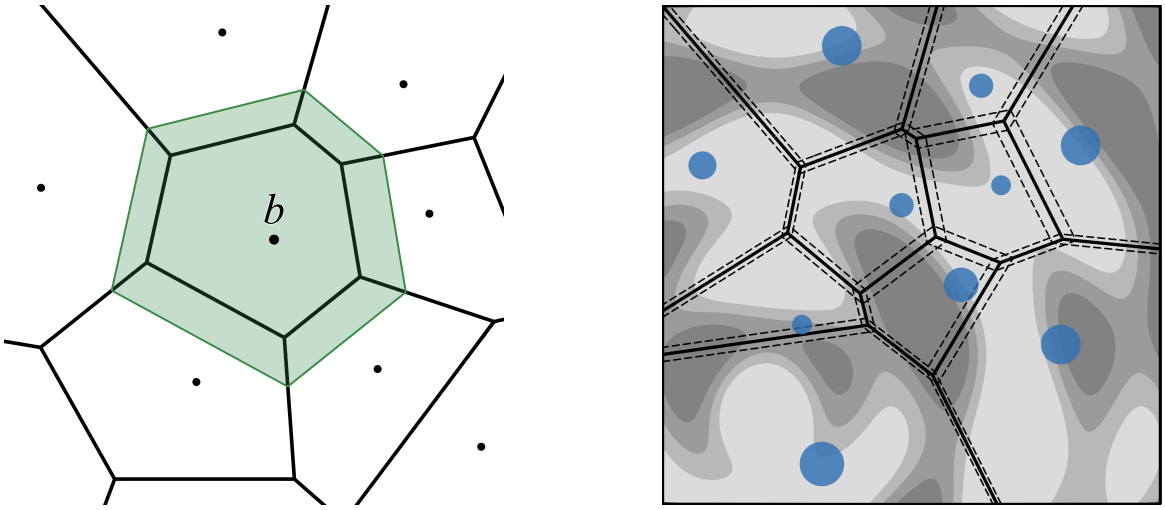
The *δ-expanded Voronoi cell* of b, denoted by $\mathrm{Vor}_y^\delta(b)$, is simply the Voronoi cell of b in the weighted Voronoi diagram $\mathrm{VD}_{y_b^\delta}(B)$ of the point set B with weights $y_b^\delta(\cdot)$. When the weights $y(\cdot)$ are clear from the context, we denote the δ-expanded Voronoi cell of b by Vor_b^δ. See Figure 2.1(a).

δ-feasibility. Any (possibly partial) transport plan τ between μ and ν along with a set of weights $y(\cdot)$ for 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 δ-expanded Voronoi cell Vor_b^δ.

For any point $a \in A$, we say that a point $b \in B$ is a *weighted nearest neighbor* (WNN) of a if $\mathrm{c}_y(a, b) \leq \mathrm{c}_y(a, b')$ for any point $b' \in B$; furthermore, b is called a *δ-weighted nearest neighbors* (δ-WNN) of a if $\mathrm{c}_y(a, b) \leq \mathrm{c}_y(a, b') + \delta$ for any point $b' \in B$. Using this definition, we can restate the definition of δ-feasibility as follows: a transport plan $\tau, y(\cdot)$ is δ-feasible if, for any point $a \in A$, all points $b \in B$ that transport a positive mass to a are δ-weighted nearest neighbors of a.

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 δ-expansion of the Voronoi cell of b,



(a) (b)

Figure 2.1: (a) The δ-expanded Voronoi cell of a point b (the green shaded region), and (b) The δ-expanded Voronoi cell of all points.

creating "soft" boundaries for the Voronoi cells. See Figure 2.1(b). This introduces an additive increase of at most δ in the cost of the transport plan.

Lemma 2.4. *Any δ-optimal transport plan $\tau, y(\cdot)$ between μ and ν is δ-close.*

Proof. Define the weighted cost of a transport plan τ' as

$$w_y(\tau') := \sum_{b\in B} \int_A \mathrm{c}_y(a,b)\tau'(a,b)\,da.$$

For any complete transport plan τ',

$$w_y(\tau') = \sum_{b\in B} \int_A \mathrm{c}_y(a,b)\tau'(a,b)\,da = \sum_{b\in B} \int_A \big(\mathrm{c}(a,b) - y(b)\big)\tau'(a,b)\,da = w(\tau') - \sum_{b\in B} y(b)\nu(b). \tag{2.2}$$

For any point $a \in A$, let b_a denote the weighted nearest neighbor of a in the set B, i.e.,

$b_a := \arg\min_{b \in B} \mathrm{c}_y(a, b)$. By property (F1) in the definition of δ-feasibility,

$$\begin{aligned} w_y(\tau) &= \sum_{b \in B} \int_A \mathrm{c}_y(a, b)\tau(a, b)\, da \leq \sum_{b \in B} \int_A \big(\mathrm{c}_y(a, b_a) + \delta\big)\tau(a, b)\, da \\ &= \sum_{b \in B} \int_A \mathrm{c}_y(a, b_a)\tau(a, b)\, da + \delta. \end{aligned} \tag{2.3}$$

Let τ^* denote an optimal transport plan between μ and ν. Then,

$$w_y(\tau^*) = \sum_{b \in B} \int_A \mathrm{c}_y(a, b)\tau^*(a, b)\, da \geq \sum_{b \in B} \int_A \mathrm{c}_y(a, b_a)\tau^*(a, b)\, da; \tag{2.4}$$

here, the second inequality holds since for each point $a \in A$, b_a is the weighted nearest of a and $\mathrm{c}_y(a, b) \geq \mathrm{c}_y(a, b_a)$ for any point $b \in B$. Combining Equations (2.3) and (2.4) and plugging τ and τ^* in Equation (2.2),

$$\begin{aligned} w(\tau) = w_y(\tau) + \sum_{b \in B} y(b)\nu(b) &\leq \left[\sum_{b \in B} \int_A \mathrm{c}_y(a, b_a)\tau(a, b)\, da + \delta\right] + \sum_{b \in B} y(b)\nu(b) \\ &\leq w_y(\tau^*) + \delta + \sum_{b \in B} y(b)\nu(b) = w(\tau^*) + \delta. \end{aligned}$$

Therefore, $w(\tau) \leq w(\tau^*) + \delta$, and τ is δ-close. □

From Lemma 2.4, to compute an ε-close transport plan, it suffices to compute a δ-feasible transport plan $\tau, y(\cdot)$ for some parameter $\delta \leq \varepsilon$. Our scaling algorithms for the semi-discrete OT problem maintain a parameter $\delta > 0$ initialized to $\delta = C_{\max}$ and run in iterations, where in each iteration, they compute a δ-feasible transport plan using the weights computed in the previous iteration and update δ as $\delta \leftarrow \delta/2$. Our algorithms terminate when the error of the computed transport plan reaches below ε and return the computed transport plan.

For the λ-robust optimal transport problem, we use a similar framework, where we define δ-expansions of the restricted Voronoi cells and refer to a transport plan $\tau, y(\cdot)$ as δ-feasible

if τ transports the mass of each point $b \in B$ to its δ-expanded restricted Voronoi cell.

2.2 Euclidean Bipartite Matching Problem

2.2.1 A Sub-Quadratic Algorithm for the Stochastic Bipartite Matching Problem

Our next result is an algorithm for the minimum-cost perfect bipartite matching problem, which is published in [75]. For a point set X, the *spread* of X, denoted by Δ, is the ratio of the largest to the smallest positive pairwise distances of the points in X.

Theorem 2.5. *Suppose U is a set of $2n$ points inside the unit d-dimensional hypercube, for any constant $d \geq 2$, and $A \subset U$ is a subset chosen uniformly at random from all subsets of size n. Let $B = U \setminus A$. Then, for any parameters $q \geq 1$, a minimum-cost perfect matching between A and B under Euclidean distances can be computed in $\tilde{O}(n^{2-\frac{1}{2d}}\Phi(n)\log\Delta)$ expected time; here, $\Phi(n)$ is the query/update time of a* DWBCP-DS *and Δ is the spread of U.*

In the general setting, for any two sets A and B each consisting of n points inside a d-dimensional unit hypercube, our algorithm computes a minimum-cost perfect matching under any ℓ_2^q norm in $\tilde{O}(n^2\Phi(n)\log\Delta)$ time, which matches the running time of the classical Hungarian algorithm up to $\text{poly}\{\log n, \log\Delta\}$ factors. Our algorithm, however, runs more efficiently when the input points A and B are random subsets of a larger point set U. This includes the case where both A and B consist of n i.i.d samples from a distribution μ, which we refer to by *stochastic point sets*.

To the best of our knowledge, this is the first sub-quadratic weakly polynomial exact algorithm for computing the pth power Euclidean bipartite matching for 2-dimensional stochastic

point sets. Furthermore, for many distributions, such as the uniform distribution on the unit square, one can show that, with high probability, the spread of the point set is bounded by a polynomial in n. For all such distributions, we achieve a strongly polynomial sub-quadratic exact algorithm.

Our algorithm uses a hierarchical partitioning tree, whose nodes (referred to as cells) are axis-parallel rectangles with an aspect ratio of at most 3. Each cell is divided into two smaller rectangles, forming its children. At any point during its execution, our algorithm maintains a set of cells $\mathcal{C}$ that partition the input points. We refer to this set of cells as *current* cells. Our algorithm computes a minimum-cost "extended" matching, where the points of B are allowed to also match to the boundaries of these cells. To compute a minimum-cost extended matching, our algorithm repeatedly identifies a minimum net-cost augmenting path and augments the matching along this path. Once a minimum-cost extended matching is computed, the algorithm removes a pair of sibling rectangles from the set of current cells and instead makes their parent cell current. By doing so, the common boundary between the siblings is erased, creating additional free points, which are then matched again by finding the minimum net-cost extended augmenting paths. When all the boundaries are erased, the algorithm terminates with the desired minimum-cost perfect matching.

The construction of our hierarchical partitioning ensures that there are only a few points close to the boundaries of the cells; furthermore, using the assumption that the point sets A and B are randomly selected from a set U of $2n$ points, we show that there exists a partial matching with a low cost and a high cardinality inside each cell of the hierarchical partitioning. Combining these two observations, we are able to prove a sub-linear bound on the expected number of points that are matched to the boundaries in the extended matchings, leading to a sub-quadratic overall running time.

2.2.2 A Geometric Partial Matching Based Sub-Quadratic Algorithm for the k-SP Problem

Our final result, presented in [136], is an algorithm that solves the bipartite matching instances generated by the k-SP problem using sub-quadratic queries to a DWBCP-DS. To do so, we design an algorithm for computing minimum-cost t-matchings, for any parameter $t \in [1, n]$, extending our approach from Section 2.2.1 to computing partial matchings. Although the running time of our algorithm for general instances of the Euclidean partial matching problem might be quadratic, we use the structure of the instances generated by the k-SP problem to show that the running time of our algorithm for computing a minimum-cost $(n-k)$-matching is $O(n^{1.8}\Phi(n)\log\Delta)$ in 2 dimensions. The following theorem states our result more formally.

Theorem 2.6. *Given any sequence σ of n requests in d dimensions, for $d \geq 2$, with a spread of Δ, and a value $1 \leq k \leq n$, there exists a deterministic algorithm that computes the optimal solution for the instance of k-SP problem under the Euclidean distances in $\tilde{O}(n^{2-\frac{1}{2d+1}}\Phi(n)\log\Delta)$ time.*

An optimal solution to the k-SP problem can be computed in $\tilde{O}(nk\Phi(n))$ time by non-trivially adapting the Hungarian algorithm to find a minimum-cost $(n-k)$-matching in $\mathcal{G}_\sigma$. However, it is worth noting that this algorithm still makes quadratic queries to the DWBCP-DS when $k = \Theta(n)$. The main contribution of this paper is the design of a novel algorithm that, for any k, computes the optimal solution to the k-SP problem while making only a sub-quadratic number of queries to a DWBCP-DS.

There are three major hurdles in proving the efficiency of our algorithm. The first challenge is in finding a minimum net-cost augmenting path, which typically requires a search on the entire graph. However, for extended matchings, we show that the minimum net-cost

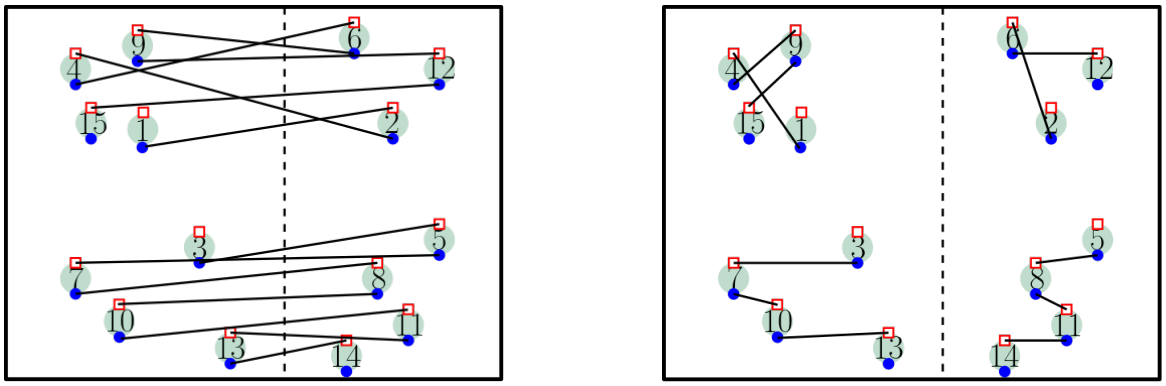


Figure 2.2: (a) A sub-problem of the k-SP problem, where the optimal solution has a high cost, and (b) there exists a low-cost high-cardinality matching inside the sub-problems.

augmenting path is fully contained inside one of the current cells. This significantly improves efficiency, as the search can be limited to individual current cells, and the overall minimum can then be determined by selecting the path with the smallest net-cost among all current cells. The second challenge is in bounding the time to merge two cells. The k-SP instance may have current cells with $\Theta(n)$ points, where the optimal solution matched each point of B to a point of A that is very far (see Figure 2.2 (a)). Despite this, we show that any minimum-cost extended matching has only $O(n^{0.8})$ points that are matched to their common boundary, helping us in bounding the time required to merge cells. To establish this, we critically use the fact that every sub-problem has a hidden low-cost high-cardinality matching M (with only sub-linearly many free points, see Figure 2.2 (b)).

The final challenge in the analysis is the following. The k free points associated with the extended $(n-k)$-matching M maintained by our algorithm may differ from the k free points in the minimum-cost $(n-k)$ matching M^*. For instance, a point $b \in B$ may be free in M^* but matched to the boundary in M, thereby leaving some other point b' unmatched. Our algorithm corrects them via alternating and augmenting paths, each of which can take $\Theta(n)$ time to find. Since there can be k such corrections, a naïve analysis leads to an upper

bound of $O(nk)$. Interestingly, by exploiting geometry, we show that the number of such corrections inside any current cell cannot exceed $O(n^{0.8})$. Using this observation, we show that our algorithm runs in $\tilde{O}(n^{1.8}\Phi(n)\log\Delta)$ time.

2.2.3 Our Framework

Recall that the Hungarian algorithm maintains a matching M and a set of non-negative dual weights $y(\cdot)$ for $A \cup B$. The matching $M, y(\cdot)$ is feasible if,

$$y(b) - y(a) \leq \mathrm{c}(a,b), \qquad \forall (a,b) \in A \times B, \tag{2.5}$$

$$y(b) - y(a) = \mathrm{c}(a,b), \qquad \forall (a,b) \in M. \tag{2.6}$$

In the Euclidean setting, for any point $v \in A \cup B$, we view the dual weight $y(v)$ as the radius of a disc centered at v. We refer to these discs as the *dual discs*. For any edge (a,b), if the dual disc of a is in the interior of the dual disc of b (Figure 2.3(a)), then $y(b) - y(a) > \mathrm{c}(a,b)$, making the dual assignment infeasible. Therefore, for any feasible dual weight assignment, the dual disc of a cannot completely lie inside the dual disc of b. The condition (2.6) corresponds to Figure 2.3(b) where the boundary of the disc of a touches the boundary of the disc of b from inside. Figure 2.3(c) is an example of an edge that only satisfies (2.5).

In order to speed up the Hungarian algorithm, we localize the search for augmenting paths, and instead of visiting the entire point set to find a minimum net-cost augmenting path, we restrict the search to points inside a small region, leading to more efficient search procedures. More precisely, we construct a hierarchical partitioning, where each cell of this hierarchical partitioning is an axis-parallel rectangle and is divided into two children by a vertical or a horizontal line.

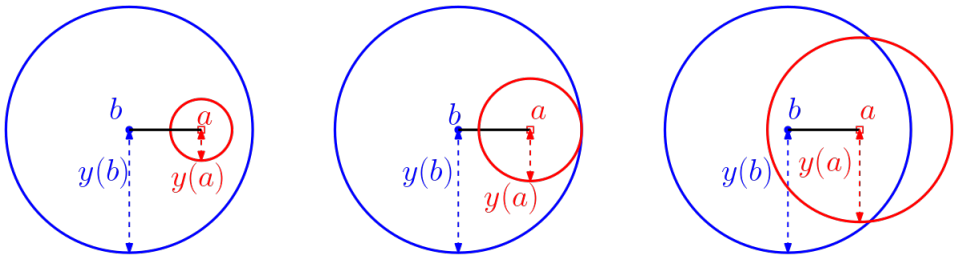
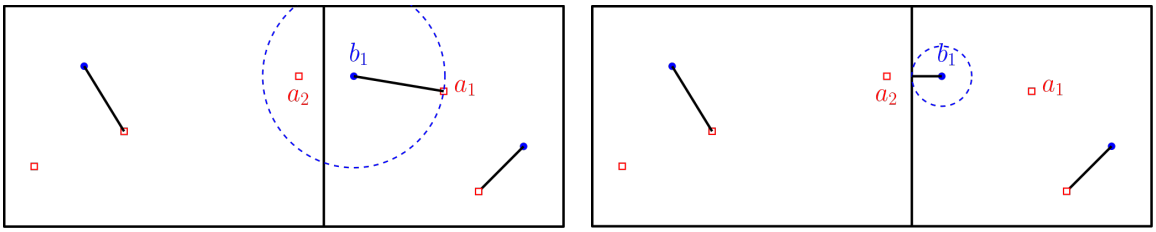


Figure 2.3: (a) An infeasible edge, (b) a feasible matching edge satisfying constraint (2.6), (c) a feasible non-matching edge satisfying constraint (2.5).

Consider a naïve implementation of the Hungarian algorithm using hierarchical partitioning. For any rectangle $\square$ in the partition, we recursively compute feasible (maximum-cardinality) matchings within each of its two child cells. We then attempt to combine these matchings at the parent cell and continue the algorithm by finding augmenting paths from unmatched points inside the parent cell. This approach, however, does not guarantee that the matchings in the children can be feasibly combined. Specifically, two points $a \in A$ and $b \in B$ located in different children $\square_a$ and $\square_b$, respectively, may violate the feasibility condition (2.6). More precisely, the optimal solution in $\square_b$ might assign a large dual weight to b, such that when $\square_a$ and $\square_b$ are merged and their shared boundary is removed, the dual disc of b fully contains that of a, resulting in an infeasibility (see Figure 2.4(a)). Resolving such violations while merging two child cells can be time-consuming, often requiring a reduction in dual weights, thereby undoing part of the progress made.

We resolve this challenge by allowing the points of B to match to the boundaries of the cells. In this way, the dual discs of all points of B inside a cell $\square_b$ are bounded by the boundaries of $\square_b$, and no point $b \in \square_b$ can have a large dual disc that might violate the feasibility conditions when $\square_b$ is being merged with a sibling cell $\square_a$ (see Figure 2.4(b)).



(a) (b)

Figure 2.4: (a) Computing feasible matchings independently inside the children of a cell, which will violate the feasibility conditions when the two matchings are combined at the parent (the pair a_2 and b_1 violate the feasibility conditions), and (b) computing restricted feasible matchings inside the children, which restricts the dual discs of the points by the boundaries of the cells and results in a feasible matching when combined at the parent.

Chapter 3

A Highly-Accurate Algorithm for the Semi-Discrete OT Problem

Given a continuous distribution μ over a bounded set $A \subset \mathbb{R}^d$, a discrete distribution ν over a set $B \subset \mathbb{R}^d$ of n points, and a parameter $\varepsilon > 0$, we present a cost-scaling algorithm for computing an ε-close transport plan between μ and ν. Assuming the existence of an oracle that computes the continuous mass inside a constant complexity region (e.g., a triangle) in constant time, our algorithm runs in $n^{O(d)} \log \frac{C_{\max}}{\varepsilon^{-1}}$ time, proving Theorem 2.1. For simplicity in presentation, we present our results for the squared Euclidean distances (i.e., ℓ_2^2 norm). Our algorithm, as well as its analysis, extends to any ℓ_p^q norm in a straightforward way. We present the details of our algorithm in Section 3.1 and analyze its efficiency and correctness in Section 3.2.

3.1 The Scaling Framework

In our algorithm, we use a black-box primal-dual discrete OT solver $\textsf{PD-OT}(\mu', \nu')$ that, given two discrete distributions μ' and ν' with supports A' and B', returns a transport plan

σ from μ' to ν' and dual weights $y(\cdot)$ for $A' \cup B'$ such that for any pair $(a, b) \in A' \times B'$,

$$y(b) - y(a) \leq \mathrm{c}(a, b), \tag{3.1}$$

$$y(b) - y(a) = \mathrm{c}(a, b), \qquad \sigma(a, b) \geq 0. \tag{3.2}$$

We note that conditions (3.1) and (3.2) are analogous to the feasibility conditions (1.11) and (1.12) that are defined for bipartite matchings, and standard primal-dual methods construct a transport plan while maintaining such constraints. For concreteness, we use Orlin's algorithm [123] that runs in $O(|A' \cup B'|^3)$ time.

The algorithm works in $O(\log(\frac{C_{\max}}{\varepsilon}))$ rounds, where $C_{\max}$ is the diameter of $A \cup B$. In each round, we have a parameter $\delta > 0$ that we refer to as the *current scale*, and we also maintain a weight $y(b)$ for every point $b \in B$. Initially, set $\delta = C_{\max}$ and $y(b) = 0$ for all $b \in B$. Execute the following steps $s = \lceil \log_2(\frac{C_{\max}}{\varepsilon}) \rceil$ times[1].

(i) *Construct a discrete OT instance:* Using the current values of dual weights of B, as described below, construct a discrete distribution $\hat{\mu}_\delta$ with a support set A_δ, where $|A_\delta| = n^{O(d)}$, and define a (discrete) distance function $\mathrm{c}_\delta : A_\delta \times B \to \{0, \ldots, 4n + 1\}$.

(ii) *Solve OT instance:* Compute an optimal transport plan between discrete distributions $\hat{\mu}_\delta$ and ν using the procedure **PD-OT**$(\hat{\mu}_\delta, \nu)$. Let σ_δ be the coupling and $\hat{y} : B \to \mathbb{R}$ be the dual weights returned by the procedure.

(iii) *Update dual weights:* $y(b) \leftarrow y(b) + \delta\hat{y}(b)$ for each point $b \in B$.

(iv) *Update scale:* $\delta \leftarrow \delta/2$.

[1] Computing an ε-close transport plan requires $O(\log(C_{\max}/\varepsilon))$ iterations. When the goal, on the other hand, is to obtain accurate dual weights up to $O(\log \varepsilon^{-1})$ bits, we need to execute our algorithm for $O(\log(nC_{\max}/\varepsilon))$ iterations. See Section 3.3.

Our algorithm terminates when $\delta \leq \varepsilon$. We now describe the details of step (i) of our algorithm, which is the only non-trivial step. Let $y(\cdot)$ be the dual weights of B at the start of scale δ.

Constructing a discrete OT instance. For each point $b \in B$, define

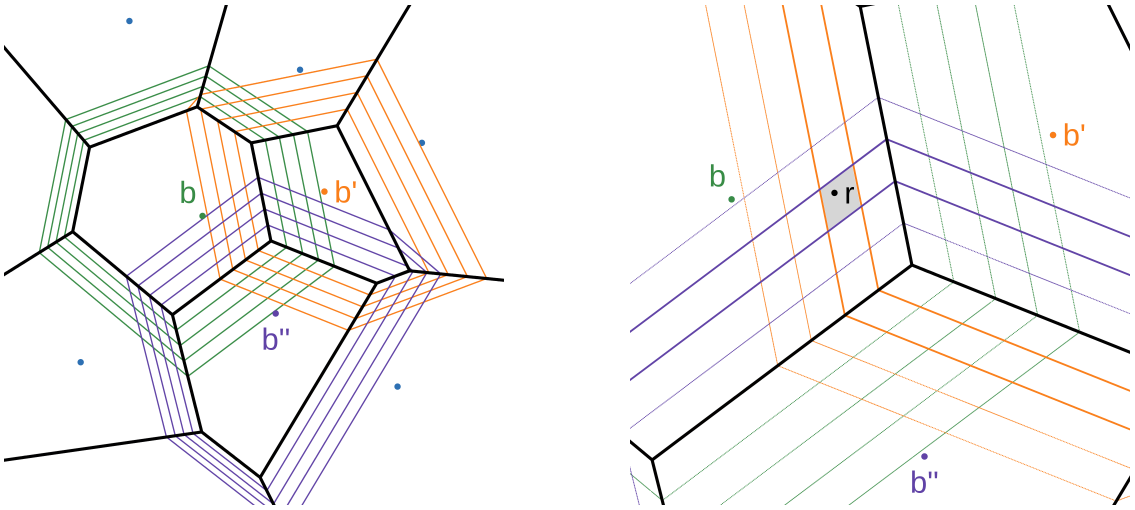
$$\mathcal{V}_b = \{\mathrm{Vor}_b^{i\delta} \mid i \in [1, 4n+1]\}$$

as the set of all $(i\delta)$-expanded Voronoi cells of b for all values $i \in [1, 4n+1]$. Define $\mathcal{V} = \bigcup_{b \in B} \mathcal{V}_b$ (See Figure 3.1(a)). Let $\mathcal{A}(\mathcal{V})$ be the *arrangement* of $\mathcal{V}$, which is the decomposition of $\mathbb{R}^d$ into (connected) regions induced by $\mathcal{V}$; each region of $\mathcal{A}(\mathcal{V})$ is the maximum connected region lying in the intersection of a subset of cells $V \subseteq \mathcal{V}$ and outside all other cells $\mathcal{V} \setminus V$ [2, 3].

For each region φ in $\mathcal{A}(\mathcal{V})$, we choose a *representative point* r_φ arbitrarily and set its mass to $\hat{\mu}_\delta(r_\varphi) = \mu(\varphi)$, where for any region ρ in $\mathbb{R}^d$, $\mu(\rho) = \int_\rho \mu(a)\, da$ is the mass of μ inside ρ; here we assume the mass to be 0 outside the support A of μ. Define a set $A_\delta = \{r_\varphi \mid \varphi \in \mathcal{A}(\mathcal{V})$ as the set of all representative points of all regions in $\mathcal{A}(\mathcal{V})$. The distribution $\hat{\mu}_\delta$ is a discrete distribution defined over A_δ. Next, define a (discrete) distance $\mathrm{c}_\delta(r, b)$ between any point $b \in B$ and a point $r \in A_\delta$ as

$$\mathrm{c}_\delta(r, b) = \begin{cases} 0, & \text{if } r \in \mathrm{Vor}_b^\delta, \\ i, & \text{if } r \in \mathrm{Vor}_b^{(i+1)\delta} \setminus \mathrm{Vor}_b^{i\delta}, \quad i \in [1, 4n], \\ 4n+1, & \text{if } r \notin \mathrm{Vor}_b^{(4n+1)\delta}. \end{cases}$$

For instance, in Figure 3.1(b), a region $\varphi \in \mathcal{A}(\mathcal{V})$ (highlighted in gray) with a representative point $r \in A_\delta$ is shown, where $\mathrm{c}_\delta(b, r) = 0$ since $r \in \mathrm{Vor}_b^\delta$, $\mathrm{c}_\delta(r, b') = 1$ since $r \in \mathrm{Vor}_{b'}^{2\delta} \setminus \mathrm{Vor}_{b'}^\delta$,



(a) (b)

Figure 3.1: (a) The expanded Voronoi cells $V_\cdot^{i\delta}$ of three points $b, b', b'' \in B$ for $i \in [1, 4]$, and (b) a region $\varphi \in \mathcal{A}(\mathcal{V})$ (highlighted in gray) with a representative point $r \in A_\delta$, where $\mathrm{c}_\delta(b, r) = 0$, $\mathrm{c}_\delta(r, b') = 1$, and $\mathrm{c}_\delta(r, b'') = 2$. The ground distance $\mathrm{c}(\cdot, \cdot)$ in this figure is squared Euclidean.

and $\mathrm{c}_\delta(r, b'') = 2$ since $r \in \mathrm{Vor}_{b''}^{3\delta} \setminus \mathrm{Vor}_{b''}^{2\delta}$. This completes the construction of $A_\delta, \hat{\mu}_\delta$, and c_δ.

Note that the partitioning $\mathcal{X}_\delta$ is defined by the arrangement of $O(n^2)$ Voronoi cells, each defined by $O(n)$ *algebraic surfaces of constant degree*. Since the combinatorial complexity of the arrangement of N algebraic surfaces of constant degree is $O(N^d)$ [30, 45], we conclude that the total number of regions in the set $\mathcal{X}_\delta$, and consequently the number of points in the set A_δ, is bounded by $n^{O(d)}$. Furthermore, each representative point can be described as a solution to a system of constant-degree inequalities, and the set A_δ can be constructed in $n^{O(d)}$ time [24].

Computing a semi-discrete transport plan. At the end of any scale δ, we compute a δ-close transport plan τ_δ from the discrete transport plan σ_δ as follows: For any edge $(r_\varphi, b) \in A_\delta \times B$, we arbitrarily transport $\sigma_\delta(r_\varphi, b)$ mass from the points inside the region φ to the point b. A simple construction of such transport plan is to set, for any region φ,

any point $a \in \varphi$, and any point $b \in B$, $\tau_\delta(a,b) = \frac{\mu(a)}{\hat{\mu}_\delta(r_\varphi)}\sigma_\delta(r_\varphi, b)$. Our algorithm will only compute the transport plan at the end of the last scale, i.e., $\delta \leq \varepsilon$.

3.2 Analysis

3.2.1 Efficiency Analysis

Our algorithm runs $O(\log(\frac{C_{\max}}{\varepsilon}))$ scales, where in each scale, it computes an arrangement of the space in $n^{O(d)}$ time, computes the amount of mass in each region of this arrangement and constructs a discrete OT instance in $Qn^{O(d)}$ time, and solves the discrete OT instance using a polynomial-time primal-dual OT solver. Since the size of the discrete OT instance is $n^{O(d)}$, solving it also takes $n^{O(d)}$ time, resulting in a total execution time of $Qn^{O(d)} \log(\frac{C_{\max}}{\varepsilon})$ for our algorithm.

3.2.2 Correctness Analysis

In this section, we show that our algorithm correctly computes an ε-close semi-discrete transport plan between μ and ν. In our analysis, we show that at any scale δ, the transport plan τ along with dual weights $y(\cdot)$ maintained by our algorithm is δ-feasible (defined in Section 2.1.4). To do so, we first describe a fine decomposition of the support A into small regions and describe a dual view of the δ-feasibility conditions.

In the discrete setting, cost scaling algorithms obtain an ε-close transport plan that satisfies (1.12) and an additive ε relaxation of (1.11). For our proof, we extend these relaxed feasibility conditions to a semi-discrete setting and show that the transport plan computed by our algorithm for a scale δ satisfies these conditions. We use the relaxed feasibility con-

ditions to show that our transport plan is δ-close. Thus, our algorithm returns an ε-close transport plan from μ to ν at the end of the last scale ($\delta \leq \varepsilon$).

Fine decomposition of A. For each scale δ, we define a decomposition of A into fine regions $\mathcal{X}_\delta$ as follows. For any pair of points b_1 and b_2 with weights w_1 and w_2, define the *weighted bisector* of b_1 and b_2, denoted by $\pi_{w_1,w2}(b_1, b_2)$, as the locus of points with the same weighted distance to b_1 and b_2, i.e.,

$$\pi_{w_1,w_2}(b_1, b_2) := \{x \in \mathbb{R}^d \mid \mathrm{c}(x, b_1) - w_1 = \mathrm{c}(x, b_2) - w_2\}.$$

Note that the Voronoi cell of any point $b \in B$ is formed by a subset of its weighted bisectors with all other points $b' \in B$. See Figure 3.2(a). For any pair of distinct points $(b_1, b_2) \in B \times B, b_1 \neq b_2$, let $\Gamma_\delta(b_1, b_2)$ denote the set of all weighted bisectors of b_1 and b_2 when the weight of b_2 is 0 and the weight of b_1 is a non-negative integer multiple of δ that is at most $C_{\max}$. More precisely, define

$$\Gamma_\delta(b_1, b_2) := \{\pi_{w,0}(b_1, b_2) \mid w \in \delta\mathbb{Z}_{\geq 0}, w \leq C_{\max}\}.$$

For instance, in Figure 3.2(b), the blue lines show the weighted bisectors in $\Gamma_\delta(b, b_1)$, the red lines show the weighted bisectors in $\Gamma_\delta(b, b_2)$, the green lines show the weighted bisectors in $\Gamma_\delta(b, b_3)$, the purple lines show the weighted bisectors in $\Gamma_\delta(b, b_4)$, and the orange lines show the weighted bisectors in $\Gamma_\delta(b, b_5)$.

Define the decomposition $\mathcal{X}_\delta$ of the support A of μ as the arrangement of all weighted bisectors in $\Gamma_\delta(b_1, b_2)$ for all pairs $(b_1, b_2) \in B \times B, b_1 \neq b_2$ over the support A. For each region $\varrho \in \mathcal{X}_\delta$, let r_ϱ denote an arbitrary representative point inside ϱ. The following lemma provides important properties of the decomposition $\mathcal{X}_\delta$.

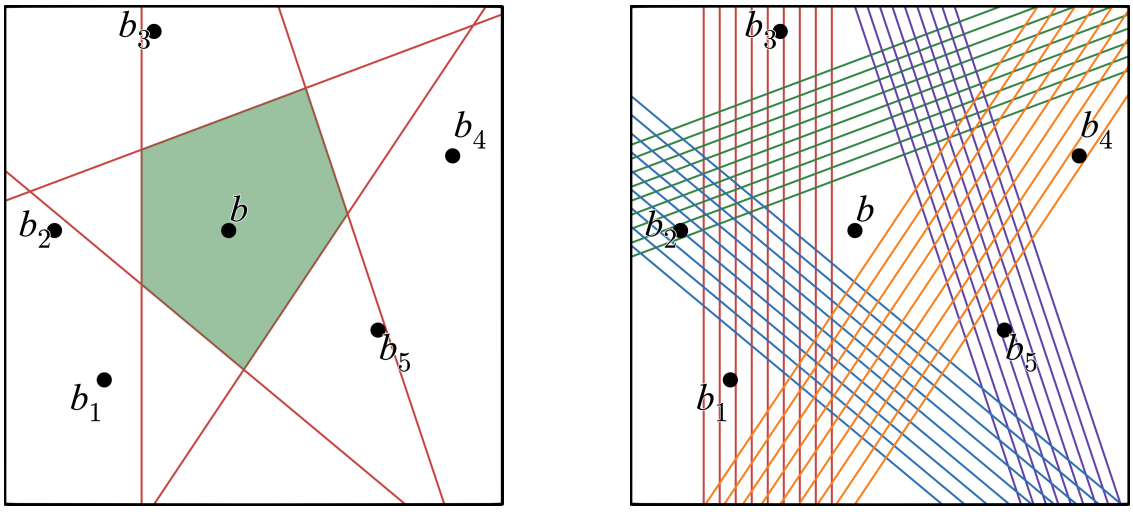


Figure 3.2: (a) The Voronoi cell of $b \in B$ is formed by a subset of the weighted bisectors between b and all other points $b' \in B$, and (b) the solid lines show the weighted bisectors in $\Gamma_\delta(b, b_i)$ for all $i \in [1, 5]$. The ground distance in this figure is squared Euclidean.

Lemma 3.1. *The decomposition $\mathcal{X}_\delta$ satisfies the following conditions: Suppose $w(\cdot)$ is a set of weights assigning a non-negative integer multiple of δ to each point $b \in B$. For any region $\varrho \in \mathcal{X}_\delta$,*

(P1) Any two points x and y in ϱ have the same weighted nearest neighbor in B with respect to $w(\cdot)$, and

(P2) For any point x inside ϱ, any δ-weighted nearest neighbor of the representative point r_ϱ with respect to $w(\cdot)$ is also a δ-weighted nearest neighbor of x.

Proof. Proof of (P1). Suppose the WNN of x with respect to $w(\cdot)$ is a point $b_x \in B$, i.e., for any other point $b' \in B$, the point x has a lower weighted distance to b_x than b'. To prove property (P1), we show that the point y also has a lower weighted distance to b_x than b' and conclude that b_x is also a WNN for y.

- If $w(b_x) - w(b') > C_{\max}$, then any point in the support A would have a lower weighted

distance to b_x than b', and hence, $\mathrm{c}_w(y, b_x) < \mathrm{c}_w(y, b')$.

- Otherwise, $w(b_x) - w(b') \leq C_{\max}$. In this case, we have $|w(b_x) - w(b')| \leq C_{\max}$, because if $w(b') - w(b_x) > C_{\max}$, the point x would have had a lower weighted distance to b' than b_x. Since $\mathrm{c}_w(x, b_x) < \mathrm{c}_w(x, b')$, the point x lies on the side of the weighted bisector $\pi_{w(b_x),w(b')}(b_x, b')$ of b_x and b' that corresponds to b_x. Since $\mathcal{X}_\delta$ is an arrangement of a set of weighted bisectors including $\pi_{w(b_x),w(b')}(b_x, b')$ (since $|w(b_x) - w(b')| \leq C_{\max}$ is an integer multiple of δ), the weighted bisector of b_x and b' cannot cross the region ϱ, and all points in ϱ has to lie in one side of $\pi_{w(b_x),w(b')}(b_x, b')$. Therefore, the point y also lies in the side of $\pi_{w(b_x),w(b')}(b_x, b')$ corresponding to b_x and $\mathrm{c}_w(y, b_x) < \mathrm{c}_w(y, b')$.

Proof of (P2). For a region $\varrho \in \mathcal{X}_\delta$, suppose a point $b \in B$ is a δ-WNN of the representative point r_φ, i.e., for any point $b' \in B$,

$$\mathrm{c}_w(r_\varphi, b) - \delta \leq \mathrm{c}_w(r_\varphi, b'). \tag{3.3}$$

Define the weights $w_+(\cdot)$ as a set of weights that assigns $w_+(b) = w(b) + \delta$ and $w_+(b') = w(b')$ to each point $b' \neq b$ in B. For any point $b' \neq b$ in B, by Equation (3.3),

$$\begin{aligned} \mathrm{c}_{w_+}(r_\varphi, b) &= \mathrm{c}(r_\varphi, b) - w_+(b) = \mathrm{c}(r_\varphi, b) - w(b) - \delta = \mathrm{c}_w(r_\varphi, b) - \delta \\ &\leq \mathrm{c}_w(r_\varphi, b') = \mathrm{c}_{w_+}(r_\varphi, b'). \end{aligned} \tag{3.4}$$

In other words, b is a WNN for the point r_φ with respect to weights w_+. Since all weights in $w_+(\cdot)$ are non-negative integer multiples of δ, by condition (P1), b is also a WNN for any point $x \in \varrho$ with respect to the weights $w_+(\cdot)$. Therefore, for any point $b' \neq b$ in B,

$$\mathrm{c}_w(x, b) - \delta = \mathrm{c}(x, b) - w(b) - \delta = \mathrm{c}_{w_+}(x, b) \leq \mathrm{c}_{w_+}(x, b') = \mathrm{c}_w(x, b'),$$

i.e., the point b is also a δ-WNN for x. □

Dual view of δ-feasibility. For any point $b \in B$, let $y(b)$ be the dual weight of b maintained by our algorithm, where $y(b)$ is a non-negative integer multiple of δ. For each region $\varrho \in \mathcal{X}_\delta$, we derive a weight $y_\delta(r_\varrho)$ for its representative point as follows. Let $b_\varrho \in B$ be the weighted nearest neighbor of r_ϱ with respect to weights $y(\cdot)$. We set the dual weight of r_ϱ as

$$y_\delta(r_\varrho) \leftarrow y(b_\varrho) - \mathrm{c}(r_\varrho, b_\varrho) - \delta. \tag{3.5}$$

Consider a transport plan τ between μ and ν along with dual weights $y(\cdot)$ for B such that, for each point $b \in B$ and each region $\varrho \in \mathcal{X}_\delta$,

$$y(b) - y_\delta(r_\varrho) \leq \mathrm{c}(r_\varrho, b) + \delta, \tag{3.6}$$

$$y(b) - y_\delta(r_\varrho) \geq \mathrm{c}(r_\varrho, b) \qquad \text{if } \tau(\varrho, b) > 0. \tag{3.7}$$

Note that conditions (3.6) and (3.7) are δ additive relaxations of conditions (3.1) and (3.2). Recall that we defined a transport plan $\tau, y(\cdot)$ as δ-feasible if for any pair of points $(a, b) \in A \times B$ with $\tau(a, b) > 0$, the point b is a δ-WNN of a (see Section 2.1.4). The following lemma shows that any transport plan $\tau, y(\cdot)$ satisfying conditions (3.6) and (3.7) is a δ-feasible transport plan.

Lemma 3.2. *Suppose τ is a transport plan and $y(\cdot)$ denotes a set of dual weights for B. Let $y_\delta(\cdot)$ denote the weights derived by Equation (3.5), and suppose the transport plan and weights satisfy conditions (3.6) and (3.7). Then, $\tau, y(\cdot)$ is δ-feasible.*

Proof. Consider any pair $(a, b) \in A \times B$ with $\tau(a, b) > 0$. Let ϱ denote the region in $\mathcal{X}_\delta$

containing the point a. From conditions (3.6) and (3.7),

$$\mathrm{c}_y(r_\varrho, b) \leq -y_\delta(r_\varrho) \leq \mathrm{c}_y(r_\varrho, b_\varrho) + \delta.$$

In other words, for any region $\varrho \in \mathcal{X}_\delta$, any point $b \in B$ that transports a positive mass to ϱ would be a δ-WNN of r_ϱ. Consequently, from Lemma 3.1 (P2), the point b is also a δ-WNN for all points inside ϱ, including the point a. Therefore, for any pair $(a, b) \in A \times B$ with $\tau(a, b) > 0$, the point b is a δ-WNN of a, and $\tau, y(\cdot)$ is δ-feasible. □

Combining Lemmas 2.4 and 3.2, any transport plan $\tau, y(\cdot)$ satisfying conditions (3.6) and (3.7) would be a δ-close transport plan between μ and ν.

Let $y(\cdot)$ denote the set of dual weights maintained by our algorithm at the beginning of scale δ. For any point $b \in B$ and any region $\varrho \in \mathcal{X}_\delta$, we define a *$\delta$-adjusted slack* on condition (3.6) for the pair (ϱ, b), denoted by $s_\delta(\varrho, b)$, as

$$s_\delta(\varrho, b) := \left\lfloor \frac{\mathrm{c}(r_\varrho, b) + \delta - y(b) + y_\delta(r_\varrho)}{\delta} \right\rfloor \delta.$$

δ-feasibility of the computed transport plan. Next, we show that for each scale δ, the semi-discrete transport plan τ_δ and dual weights $(y + \delta\hat{y})(\cdot)$ for the points in B computed by our algorithm at the end of the scale is a δ-feasible transport plan. We begin by relating the decomposition $\mathcal{X}_\delta$ to the partitioning $\mathcal{A}(\mathcal{V})$ that is constructed in step (i) of our algorithm. We also relate the distance $\mathrm{c}_\delta(\cdot, \cdot)$ computed by our algorithm to the slacks $s_\delta(\cdot, \cdot)$.

In any scale δ, since the dual weight of each point is an integer multiple of δ, for each point $b \in B$ and each $i \in [1, 4n + 1]$, all boundaries of the $(i\delta)$-expansion $\mathrm{Vor}_b^{i\delta}$ are either parts of the weighted bisectors that are used in the construction of the fine decomposition $\mathcal{X}_\delta$ or are completely outside of the support A of μ. Hence, by the construction of $\mathcal{X}_\delta$, each region

$\varrho \in \mathcal{X}_\delta$ completely lies inside some region $\varphi \in \mathcal{A}(\mathcal{V})$, i.e., each region in $\mathcal{A}(\mathcal{V})$ consists of a collection of regions in $\mathcal{X}_\delta$.

In the next lemma, we establish a connection between the slacks and the distances $\mathrm{c}_\delta(\cdot,\cdot)$.

Lemma 3.3. *For any region $\varphi \in \mathcal{A}(\mathcal{V})$, any region $\varrho \in \mathcal{X}_\delta$ inside φ, and any point $b \in B$, if $\mathrm{c}_\delta(r_\varphi, b) \leq 4n$, then $s_\delta(\varrho, b) = \mathrm{c}_\delta(r_\varphi, b)\delta$. Furthermore, if $\mathrm{c}_\delta(r_\varphi, b) = 4n+1$, then $s_\delta(\varrho, b) \geq (4n+1)\delta$.*

Proof. For any region $\varrho \in \mathcal{X}_\delta$, suppose $b_\varrho \in B$ denotes the WNN of r_ϱ with respect to $y(\cdot)$. For any point $b \in B$, rewrite the slack $s_\delta(\varrho, b)$ as follows.

$$\begin{aligned} s_\delta(\varrho, b) &= \left\lfloor \frac{\mathrm{c}(r_\varrho, b) + \delta - y(b) + y_\delta(r_\varrho)}{\delta} \right\rfloor \delta \\ &= \left\lfloor \frac{\mathrm{c}(r_\varrho, b) + \delta - y(b) + (y(b_\varrho) - \mathrm{c}(r_\varrho, b_\varrho) - \delta)}{\delta} \right\rfloor \delta \\ &= \left\lfloor \frac{\mathrm{c}_y(r_\varrho, b) - \mathrm{c}_y(r_\varrho, b_\varrho)}{\delta} \right\rfloor \delta. \end{aligned} \tag{3.8}$$

For each point $b \in B$, let $\mathrm{Vor}_b = \mathrm{Vor}_y(b)$ denote the Voronoi cell of b in $\mathrm{VD}_y(B)$. Recall that for any $i \in [1, 4n+1]$, $\mathrm{Vor}_b^{i\delta}$ denotes the δ-expanded Voronoi cell of b. Consider any pair $(\varrho, b) \in \mathcal{X}_\delta \times B$, and suppose r_ϱ lies inside $\mathrm{Vor}_b^{i\delta}$ for some $i \in [1, 4n+1]$. Let $y_b^i(\cdot)$ denote a set of dual weights for B that assigns $y_b^i(b) = y(b) + i\delta$ to b and $y_b^i(b') = y(b')$ to each point $b' \neq b$ in B. Since r_ϱ lies inside $\mathrm{Vor}_b^{i\delta}$, then b is the WNN of r_ϱ with respect to $y_b^i(\cdot)$, i.e., $\mathrm{c}_{y_b^i}(r_\varrho, b) < \mathrm{c}_{y_b^i}(r_\varrho, b')$ for each point $b' \neq b \in B$. Therefore,

$$\begin{aligned} \mathrm{c}_y(r_\varrho, b) &= \mathrm{c}(r_\varrho, b) - y(b) = \mathrm{c}(r_\varrho, b) - (y_b^i(b) - i\delta) = \mathrm{c}_{y_b^i}(r_\varrho, b) + i\delta \\ &< \mathrm{c}_{y_b^i}(r_\varrho, b_\varrho) + i\delta = \mathrm{c}_y(r_\varrho, b_\varrho) + i\delta. \end{aligned}$$

Plugging into Equation (3.8), $s_\delta(\varrho, b) = \left\lfloor \frac{\mathrm{c}_y(r_\varrho, b) - \mathrm{c}_y(r_\varrho, b_\varrho)}{\delta} \right\rfloor \delta < i\delta$ for any region $\varrho \in \mathcal{X}_\delta$ inside

$\mathrm{Vor}_b^{i\delta}$. Furthermore, for any region $\varrho \in \mathcal{X}_\delta$ outside of $\mathrm{Vor}_b^{i\delta}$, the WNN of ϱ with respect to $y_b^i(\cdot)$ remains b_ϱ and $\mathrm{c}_{y_b^i}(r_\varrho, b) > \mathrm{c}_{y_b^i}(r_\varrho, b_\varrho)$. Therefore,

$$\mathrm{c}_y(r_\varrho, b) = \mathrm{c}_{y_b^i}(r_\varrho, b) + i\delta > \mathrm{c}_{y_b^i}(r_\varrho, b_\varrho) + i\delta = \mathrm{c}_y(r_\varrho, b_\varrho) + i\delta.$$

Plugging into Equation (3.8), $s_\delta(\varrho, b) = \left\lfloor \frac{\mathrm{c}_y(r_\varrho, b) - \mathrm{c}_y(r_\varrho, b_\varrho)}{\delta} \right\rfloor \delta \geq i\delta$ for any region $\varrho \in \mathcal{X}_\delta$ outside $\mathrm{Vor}_b^{i\delta}$. Thus, for any point $b \in B$, any region $\varphi \in \mathcal{A}(\mathcal{V})$, and any $\varrho \in \mathcal{X}_\delta$ inside φ,

- if r_ϱ lies inside Vor_b^{δ}, then $s_\delta(\varrho, b) = 0$. In this case, r_φ also lies inside Vor_b^{δ} and $\mathrm{c}_\delta(r_\varphi, b) = 0$,
- if r_ϱ lies inside $\mathrm{Vor}_b^{(i+1)\delta} \setminus \mathrm{Vor}_b^{i\delta}$ for some $i \in [1, 4n+1]$, then $s_\delta(\varrho, b) = i\delta$. In this case, r_φ also lies in $\mathrm{Vor}_b^{(i+1)\delta} \setminus \mathrm{Vor}_b^{i\delta}$ and $\mathrm{c}_\delta(r_\varphi, b) = i$, and
- if r_ϱ lies outside $\mathrm{Vor}_b^{(4n+1)\delta}$, then $s_\delta(\varrho, b) \geq (4n+1)\delta$. In this case, r_φ also lies outside of $\mathrm{Vor}_b^{(4n+1)\delta}$ and $\mathrm{c}_\delta(r_\varphi, b) = 4n+1$.

This completes the proof of this lemma. □

The following lemma, whose proof is provided in Appendix A.1, is helpful in proving Lemma 3.5.

Lemma 3.4. *Let $\tau_{2\delta}, y(\cdot)$ be any 2δ-feasible transport plan between μ and ν, where the dual weights of points in B are integer multiples of 2δ. Then, for any region $\varrho \in \mathcal{X}_\delta$ and any point $b \in B$, if $\tau_{2\delta}(\varrho, b) > 0$, then $s_\delta(\varrho, b) \leq 4\delta$.*

Recall that A_δ denotes the set of representative points of the regions in $\mathcal{A}(\mathcal{V})$ and $\hat{\mu}_\delta$ is the discrete distribution over A_δ computed by our algorithm at step (i). We next show that any optimal transport plan σ^* between $\hat{\mu}_\delta$ and ν under the distance function $\mathrm{c}_\delta(\cdot, \cdot)$ does not transport mass on edges $(r_\varphi, b) \in A_\delta \times B$ with cost $\mathrm{c}_\delta(r_\varphi, b) > 4n$.

Lemma 3.5. *For any scale δ, let σ^* be any optimal transport plan between $\hat{\mu}_\delta$ and ν. For any point $b \in B$ and any region $\varphi \in \mathcal{A}(\mathcal{V})$, if σ^* transports mass from r_φ to b, then $\mathrm{c}_\delta(r_\varphi, b) \leq 4n$.*

Proof. Let $\tau_{2\delta}, y(\cdot)$ be the 2δ-feasible transport plan computed by our algorithm at scale 2δ. Let $\sigma_{2\delta}$ denote a transformation of $\tau_{2\delta}$ into a discrete transport plan between $\hat{\mu}_\delta$ and ν by simply setting, for each region $\varphi \in \mathcal{A}(\mathcal{V})$, $\sigma_{2\delta}(r_\varphi, b) := \tau_{2\delta}(\varphi, b)$. Let σ^* be any optimal transport plan between $\hat{\mu}_\delta$ and ν, where the cost of each edge (r_φ, b) is set to $\mathrm{c}_\delta(r_\varphi, b)$. Define a directed graph $\mathcal{G}_\delta$ on the vertex set $A_\delta \cup B$ as follows. For any pair $(r, b) \in A_\delta \times B$, if $\sigma^*(r, b) > \sigma_{2\delta}(r, b)$, then we add an edge, called a forward edge, directed from r to b with a capacity $\sigma^*(r, b) - \sigma_{2\delta}(r, b)$; otherwise, if $\sigma^*(r, b) < \sigma_{2\delta}(r, b)$, then we add an edge, called a *backward edge*, directed from b to r with a capacity $\sigma_{2\delta}(r, b) - \sigma^*(r, b)$. This completes the construction of $\mathcal{G}_\delta$. Note that, since both transport plans σ^* and $\sigma_{2\delta}$ are complete transport plans, any edge in $\mathcal{G}_\delta$ would be contained in a simple directed cycle (see Lemma A.3 in the appendix for a proof).

For the sake of contradiction, suppose there is a pair $(r^*, b^*) \in A_\delta \times B$ with $\mathrm{c}_\delta(r^*, b^*) > 4n$ such that $\sigma^*(r^*, b^*) > 0$. From Lemma 3.4, $\sigma_{2\delta}(r^*, b^*) = 0$ and the edge (r^*, b^*) is a forward edge; hence, (r^*, b^*) is contained in a simple directed cycle $C = \langle b_1, r_1, \ldots, b_k, r_k, b_{k+1} = b_1 \rangle$ in $\mathcal{G}_\delta$. Note that by the construction of $\mathcal{G}_\delta$, the edges of C alternate between forward and backward edges. Define the cost of the cycle C as

$$\phi(C) := \sum_{i=1}^{k} \mathrm{c}_\delta(r_i, b_{i+1}) - \sum_{i=1}^{k} \mathrm{c}_\delta(r_i, b_i);$$

i.e., the cost of C is simply the total cost of its forward edges minus the total cost of its backward edges. Since σ^* is an optimal transport plan between $\hat{\mu}_\delta$ and ν, any directed cycle C on $\mathcal{G}_\delta$ has a non-positive cost (otherwise, by canceling the cycle, one can obtain a transport plan with a cost lower than $w(\sigma^*)$, which is a contradiction with the assumption

that σ^* is an optimal transport plan). Since C is a simple cycle, the length of C is at most $2n$. Furthermore, from Lemma 3.4, any backward edge has a distance at most 4, i.e., for each $i \in [1, k], \mathrm{c}_\delta(r_i, b_i) \leq 4$. Finally, by construction, all edges have a non-negative distance. Therefore,

$$\phi(C) = \sum_{i=1}^{k} \mathrm{c}_\delta(r_i, b_{i+1}) - \sum_{i=1}^{k} \mathrm{c}_\delta(r_i, b_i) \geq \mathrm{c}_\delta(r^*, b^*) - \sum_{i=1}^{k} 4 \geq \mathrm{c}_\delta(r^*, b^*) - 4n > 0,$$

which is a contradiction of the fact that all simple cycles have a non-positive cost. Hence, σ^* cannot transport mass on edges (r^*, b^*) with distance $\mathrm{c}_\delta(r^*, b^*) > 4n$. □

Let $\sigma_\delta, \hat{y}(\cdot)$ be the optimal transport plan between $\hat{\mu}_\delta$ and ν computed at step (ii) of our algorithm, and recall that τ_δ is the transport plan between μ and ν computed at the end of scale δ. In the following lemma, we show that $\tau_\delta, (y + \delta\hat{y})(\cdot)$ is a δ-feasible transport plan.

Lemma 3.6. *For each scale δ, let $(y + \delta\hat{y})(\cdot)$ denote the set of dual weights for points in B computed at step (iii) of our algorithm. Then, the transport plan $\tau_\delta, (y + \delta\hat{y})(\cdot)$ is a δ-feasible transport plan.*

Proof. Let $y_\delta(\cdot)$ denote the set of dual weights derived for the representative points of regions in $\mathcal{X}_\delta$ using Equation (3.5) at the beginning of scale δ. Consider a set of dual weights $y'_\delta(\cdot)$ for $\mathcal{X}_\delta$ that assigns, for each region $\varrho \in \mathcal{X}_\delta$ inside a region $\varphi \in \mathcal{A}(\mathcal{V})$, a dual weight $y'_\delta(r_\varrho) := y_\delta(r_\varrho) + \delta\hat{y}(r_\varphi)$. In what follows, we show that the transport plan τ_δ along with dual weights $(y + \delta\hat{y})(\cdot)$ for B and $y'_\delta(\cdot)$ for $\mathcal{X}_\delta$ satisfy conditions (3.6) and (3.7). In Lemma A.1 in the appendix, we show that by reassigning the dual weights of the representative points of regions in $\mathcal{X}_\delta$ as in Equation (3.5), the conditions (3.6) and (3.7) remain satisfied; hence, we conclude that $\tau, (y + \delta\hat{y})(\cdot)$ is δ-feasible, as claimed.

For any region $\varphi \in \mathcal{A}(\mathcal{V})$, any region $\varrho \in \mathcal{X}_\delta$ inside φ, and any point $b \in B$,

- by Lemma 3.3, $\mathrm{c}_\delta(r_\varphi, b)\delta \leq s_\delta(\varrho, b)$. Combining with feasibility condition (3.1),

$$\begin{aligned}(y+\delta\hat{y})(b) - y'_\delta(r_\varrho) &= (y(b)+\delta\hat{y}(b)) - (y_\delta(r_\varrho)+\delta\hat{y}(r_\varphi)) \\ &= (y(b)-y_\delta(r_\varrho)) + \delta(\hat{y}(b)-\hat{y}(r_\varphi)) \\ &\leq (y(b)-y_\delta(r_\varrho)) + \mathrm{c}_\delta(r_\varphi, b)\delta \\ &\leq (y(b)-y_\delta(r_\varrho)) + s_\delta(\varrho, b) \\ &\leq (y(b)-y_\delta(r_\varrho)) + (\mathrm{c}(r_\varrho, b)+\delta-y(b)+y_\delta(r_\varrho)) \\ &= \mathrm{c}(r_\varrho, b)+\delta,\end{aligned}$$

  leading to condition (3.6).

- if $\tau_\delta(\varrho, b) > 0$, then σ_δ transports mass from r_φ to b, i.e., $\sigma_\delta(r_\varphi, b) > 0$. In this case, by Lemma 3.5, $\mathrm{c}_\delta(r_\varphi, b) \leq 4n$ and by Lemma 3.3, $s_\delta(\varrho, b) = \mathrm{c}_\delta(r_\varphi, b)\delta$. Combining with feasibility condition (3.2),

$$\begin{aligned}(y+\delta\hat{y})(b) - y'_\delta(r_\varrho) &= (y(b)+\delta\hat{y}(b)) - (y_\delta(r_\varrho)+\delta\hat{y}(r_\varphi)) \\ &= (y(b)-y_\delta(r_\varrho)) + \delta(\hat{y}(b)-\hat{y}(r_\varphi)) \\ &= (y(b)-y_\delta(r_\varrho)) + \mathrm{c}_\delta(r_\varphi, b)\delta \\ &= (y(b)-y_\delta(r_\varrho)) + s_\delta(\varrho, b) \\ &\geq (y(b)-y_\delta(r_\varrho)) + (\mathrm{c}(r_\varrho, b)-y(b)+y_\delta(r_\varrho)) \\ &= \mathrm{c}(r_\varrho, b),\end{aligned}$$

  leading to condition (3.7).

Recall that, from Lemma 3.2, any transport plan that satisfies conditions (3.6) and (3.7) is δ-feasible; hence, $\tau_\delta, (y+\delta\hat{y})(\cdot)$ is a δ-feasible transport plan, as desired. □

From Lemma 3.6, the transport plan $\tau_\delta, (y + \delta\hat{y})(\cdot)$ maintained by our algorithm after step (iii) is δ-feasible, and from Lemma 2.4, τ_δ is a δ-close transport plan. Therefore, when $\delta \leq \varepsilon$, the computed transport plan is an ε-close transport plan. This concludes our correctness analysis.

3.3 Computing Optimal Dual Weights

In this section, we show that in addition to computing an ε-close transport cost in the semi-discrete setting, our algorithm can also compute the set of dual weights for the points in B accurately, up to $O(\log \varepsilon^{-1})$ bits. To obtain such an accurate set of dual weights, we execute our algorithm for $O(\log(nC_{\max}/\varepsilon))$ iterations so that the final value of δ when the algorithm terminates is at most $\varepsilon/5n$. In the following, we show that the dual weight computed for each point in B at the last scale is ε-close to the optimal dual weight value.

Note that any edge in the graph constructed in step (i) of our algorithm has a cost at most $4n+1$. Consequently, in step (ii), the largest dual weight returned by the primal-dual solver is at most $4n + 1$[2] and in step (iii), the dual weight of any point $b \in B$ changes by at most $(4n+1)\delta$. Since the dual weight of b becomes the optimal dual weight in the limit, to bound the difference between the current dual weight and the optimal dual weight, it suffices if we bound the total change in the dual weights for all scales after scale $\delta \leq \varepsilon/5n$. The difference between the optimal dual weight and the current dual weight is at most

$$(4n + 1) \sum_{i=1}^{\infty} \delta/2^i = (4n + 1)\delta \leq (4n + 1)(\varepsilon/5n) \leq \varepsilon.$$

Therefore, after $O(\log(nC_{\max}/\varepsilon))$ iterations of the algorithm, the difference in the optimal

[2]Any set of dual weights returned by the algorithm can be translated by a fixed value so that the smallest dual weight becomes 0. Assuming this, it is easy to see that the largest dual weight is $4n + 1$.

dual weight $y(b)$ and the current dual weight of b is at most ε.

Chapter 4

A Combinatorial Algorithm for the Semi-Discrete OT Problem

In this chapter, we extend the combinatorial framework used by discrete OT algorithms to the semi-discrete settings and use this framework to design an efficient scaling algorithm for computing an ε-close semi-discrete OT algorithm. Our algorithm as well as its analysis extends to any ℓ_p^q norm and any dimension d in a straightforward way. To better convey the main ideas, we describe our algorithm for the 2-dimensional setting where the ground distance is the squared Euclidean distance.

4.1 Combinatorial Framework

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 $\mathbb{R}^2$. Let $y(\cdot)$ denote a set of weights for the points in B. Recall that for any parameter $\delta > 0$, the δ-expanded Voronoi cell of any point $b \in B$ with weights $y(\cdot)$, denoted by Vor_b^δ, is defined as the Voronoi cell of b with respect to a weight function y_b^δ defined as follows:

$$y_b^\delta(b') = \begin{cases} y(b') + \delta, & b' = b, \\ y(b'), & b' \neq b. \end{cases} \tag{4.1}$$

See Figure 4.1(a). Additionally, recall that any (possibly partial) transport plan τ along with weights $y(\cdot)$ for 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 δ-expanded Voronoi cell Vor_b^δ.

Finally, recall that any δ-feasible transport plan $\tau, y(\cdot)$ is called a δ-optimal transport plan if τ is a complete transport plan between μ and ν. As shown in Lemma 2.4, for any δ-optimal transport plan $\tau, y(\cdot)$, the transport plan τ is δ-close.

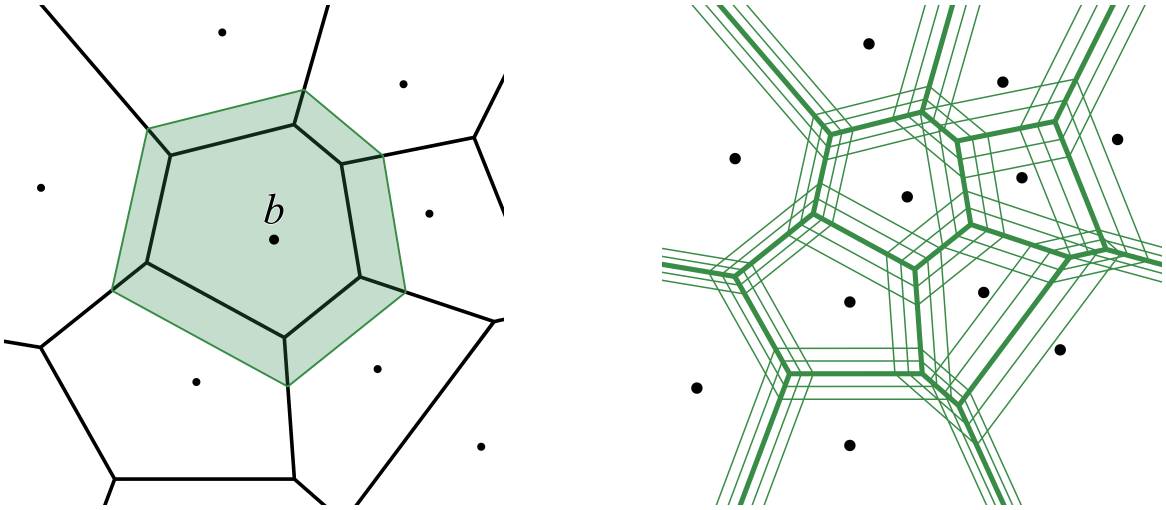
Given a 2δ-feasible transport plan $\tau, y(\cdot)$, in the following, we define a residual graph and an augmenting path. We also introduce the process of augmenting τ along an augmenting path, which allows us to increase the mass transported by τ.

Residual graph. Given a 2δ-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.

Partitioning of A. For each point $b \in B$, consider the Voronoi cell of b and its δ- and 2δ-expansions Vor_b^δ and $\mathrm{Vor}_b^{2\delta}$. Let $\mathcal{X}_\delta$ denote the arrangement of these $3n$ cells across all n points of B. See Figure 4.1(b). For each region φ in this arrangement, pick an arbitrary representative point r_φ inside φ and assign it a mass of $\hat{\mu}_\delta(r_\varphi) := \mu(\varphi)$, 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 $\mathcal{X}_\delta$. Note that $\hat{\mu}_\delta$ is a distribution over A_δ.

The following observation is straightforward from the construction of the partitioning $\mathcal{X}_\delta$.

Proposition 4.1. *For any region $\varphi \in \mathcal{X}_\delta$ and any point $b \in B$, the region φ either completely lies inside $\mathrm{Vor}_b^{2\delta}$ or it is completely outside $\mathrm{Vor}_b^{2\delta}$.*



(a) (b)

Figure 4.1: (a) The δ-expanded Voronoi cell Vor_b^δ of b (green shaded area), and (b) the partitioning $\mathcal{X}_\delta$.

Vertex set. 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*, denoted by $\mathrm{ex}(b)$, 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 $\mathrm{ex}(r_\varphi) = \hat{\mu}_\delta(r_\varphi) - \sum_{b' \in B} \tau(\varphi, b')$.

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 $r_\varphi \in \mathrm{Vor}_b^{2\delta}$ and $\tau(\varphi, b) < \min\{\mu(\varphi), \nu(b)\}$, 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 4.2. *For any $\delta > 0$ and a 2δ-feasible transport plan $\hat{\tau}, y(\cdot)$, the residual graph $\mathcal{G}_\delta$ has $O(n^2)$ nodes and $O(n^3)$ edges.*

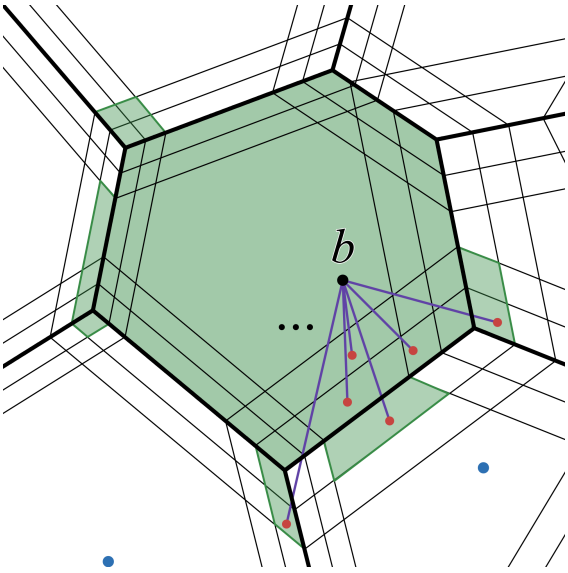


Figure 4.2: The green region shows the mass of μ that is transported to b, the red points show the representative points of regions, and the purple segments show the compressed transport plan $\hat{\tau}$.

The proof of this lemma is provided in Appendix A.2.1. 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., $b_\varphi := \min_{b \in B} \mathrm{c}_y(r_\varphi, b)$. Define $\mathrm{c}_y(r_\varphi, t) := \mathrm{c}_y(r_\varphi, b_\varphi) + \delta$.

Compressing a semi-discrete transport plan. Given a semi-discrete transport plan τ between μ and ν, we construct a discrete transport plan $\hat{\tau}$ between $\hat{\mu}_\delta$ and ν 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). See Figure 4.2. We refer to the transport plan $\hat{\tau}$ as the *compressed transport plan.*

Lemma 4.3. *For any 2δ-feasible semi-discrete transport plan $\tau, y(\cdot)$ between μ and ν, the compressed transport plan $\hat{\tau}$ between $\hat{\mu}_\delta$ and ν along with weights $y(\cdot)$ is also 2δ-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 $r_\varphi \in \text{Vor}_b^{2\delta}$. 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 $a \in \text{Vor}_b^{2\delta}$. Therefore, using Proposition 4.1, the region φ has to completely lie inside $\text{Vor}_b^{2\delta}$, and the representative point r_φ, which is a point inside φ, also lies inside $\text{Vor}_b^{2\delta}$. Hence, $\hat{\tau}, y(\cdot)$ is 2δ-feasible. □

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

Lemma 4.4. *Any 2δ-feasible (discrete) transport plan $\hat{\tau}, y(\cdot)$ between $\hat{\mu}_\delta$ and ν can be converted into a 2δ-feasible semi-discrete transport plan τ between μ and ν.*

Proof. Consider any transport plan τ over $A \times B$ such that for any point $b \in B$ and any region $\varphi \in \mathcal{X}_\delta$, τ transports a mass of $\hat{\tau}(r_\varphi, b)$ from b to the continuous 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 2δ-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 $a \in \text{Vor}_b^{2\delta}$. 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 $r_\varphi \in \text{Vor}_b^{2\delta}$. By Proposition 4.1, the whole region φ lies inside $\text{Vor}_b^{2\delta}$ and therefore, the point a also lies inside $\text{Vor}_b^{2\delta}$. Hence, $\tau, y(\cdot)$ is 2δ-feasible. □

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 2δ-feasible transport plan $\hat{\tau}, y(\cdot)$, an

alternating path (resp. *alternating cycle*) is a directed path (resp. directed cycle) in $\mathcal{G}_\delta$. Notice that for any directed path (resp. cycle) on 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, denoted by $\mathrm{bc}(P)$, 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 4.5. *Suppose $\hat{\tau}, y(\cdot)$ is a 2δ-feasible transport plan and P denotes an augmenting path in the residual graph. Then, after augmenting $\hat{\tau}$ along P,*

(i) the transport plan $\hat{\tau}$ remains a valid transport plan and $\hat{\tau}, y(\cdot)$ remains 2δ-feasible, and

(ii) either a backward edge gets removed from the transport plan or a free point becomes fully transported.

Proof. Let $P = \langle s, b_1, r_1, \ldots, b_k, r_k, t \rangle$ denote an augmenting path in the residual graph. 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, $\mathrm{bc}(P) \leq \hat{\tau}(r_i, b_{i+1})$; hence, $\hat{\tau}(r_i, b_{i+1}) \geq 0$ after augmentation. Furthermore, for any forward edge (b_i, r_i), $i \in \{1, \ldots, k\}$, we increase $\hat{\tau}(r_i, b_i)$ by $\mathrm{bc}(P)$, and $\hat{\tau}(r_i, b_i)$ remains non-negative.

For any $i \in \{2, \ldots, k\}$, we increase $\hat{\tau}(b_i, r_i)$ and 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 $\hat{\tau}(b_i, r_i)$ and 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), on the other hand, we only increase the amount of mass transported from b_1 (resp. into r_k) by $\mathrm{bc}(P)$, where by definition, $\mathrm{bc}(P) \leq \mathrm{ex}(b_1)$ (resp. $\mathrm{bc}(P) \leq \mathrm{ex}(r_k)$). Therefore, the total mass transport from b_1 (resp. into r_k) after augmentation is at most $\nu(b)$ (resp. μ_{r_k}), and τ remains a valid transport plan.

When augmenting $\hat{\tau}$ along P, we increase the amount of mass transported on forward edges (b_i, r_i) for each $i \in [1, k]$ and decrease the amount of mass transported 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 a point b to the subset of A_δ that lies inside $\mathrm{Vor}_b^{2\delta}$, the edge (r, b) satisfies the 2δ-feasibility condition (F1), and $\hat{\tau}, y(\cdot)$ remains 2δ-feasible after augmentation, proving property (i).

To prove (ii), 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 (r, b) is removed from the residual graph. Otherwise, if the endpoint b_1 (resp. r_k) determines the bottleneck capacity of P, then the mass transport from b_1 (resp. into r_k) will be increased by $\mathrm{ex}(b_1)$ (resp. $\mathrm{ex}(r_k)$), and the mass of b_1 (resp. r_k) would be fully transported after augmentation. □

Consider the following straightforward way to augment a semi-discrete transport plan. Given a 2δ-feasible semi-discrete transport plan τ, $y(\cdot)$, we can compute an augmenting path P in the residual graph and augment the compressed transport plan $\hat{\tau}$ along P. From Lemmas 4.3 and 4.5, $\hat{\tau}$ remains 2δ-feasible and from Lemma 4.4, the updated 2δ-feasible transport plan $\hat{\tau}$ can be converted to a 2δ-feasible semi-discrete transport plan as desired. To obtain a com-

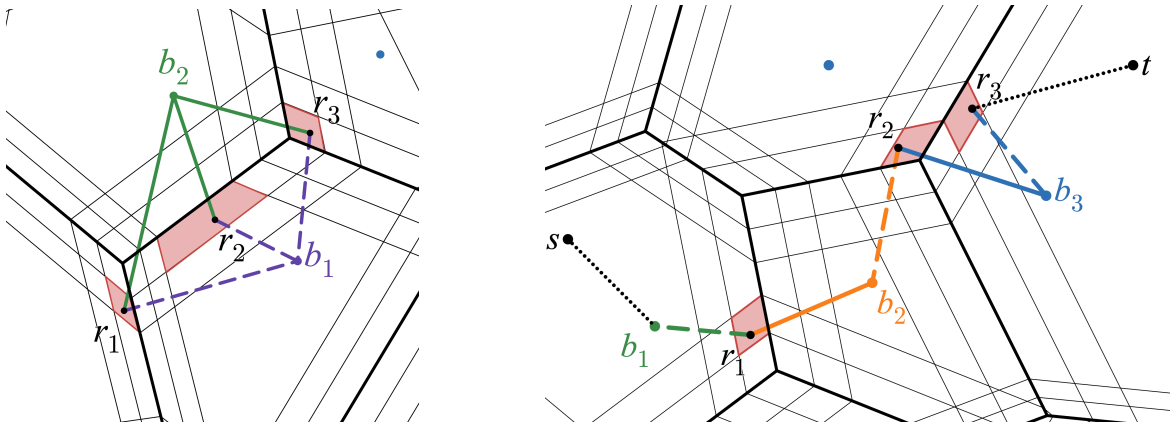


Figure 4.3: (a) Three admissible triples $(b_1, r_1, b_2), (b_1, r_2, b_2)$, and (b_1, r_3, b_2), where solid (resp. dashed) lines show backward (resp. forward) edges, and (b) an admissible augmenting path.

plete 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 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 2δ-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 $\mathrm{c}_y(r_\varphi, b_1) < \mathrm{c}_y(r_\varphi, b_2)$[1]. See Figure 4.3(a). 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 $\mathrm{c}_y(r_\varphi, b_1) < \mathrm{c}_y(r_\varphi, b_2)$. Intuitively, for any admissible triple (b_1, r_φ, b_2), the mass of μ inside r_φ is transported from b_2 but b_1 is closer 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

[1]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 a definition per edge.

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. For instance, in Figure 4.3(b), the point b_2 transports mass to r_1 while 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.

4.2 Algorithm

This section presents our cost-scaling algorithm, which uses the combinatorial framework from Section 4.1 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 4.2.1 for details). Furthermore, the updated residual graph may have cycles. We introduce additional steps in our algorithm to eliminate these cycles (see Section 4.2.3 for details).

Overview

The algorithm runs for $O(\log \frac{C_{\max}}{\varepsilon})$ scales, where $C_{\max}$ 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 $\delta = C_{\max}^2$ 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:

   (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 4.2.1. At the end of this step, there are no admissible augmenting paths in the residual graph.

   (ii) Increase by δ the weights of all points of B that are reachable from the source by admissible paths 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 4.2.2.

   (iii) Update the compressed transport plan $\hat{\tau}_\delta$ and the residual graph using the ACYCLIFY procedure described in Section 4.2.3, 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 $\hat{\tau}_\delta$.

Invariants. As shown in Section 4.3 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 2δ-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. Our algorithm in Chapter 3 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.

4.2.1 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 SEARCHANDAUGMENT procedure is similar in style to the blocking flow procedure in Dinic's max-flow algorithm [58] or the partial-DFS procedure in Gabow-Tarjan's algorithm [70], 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 4.4.

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

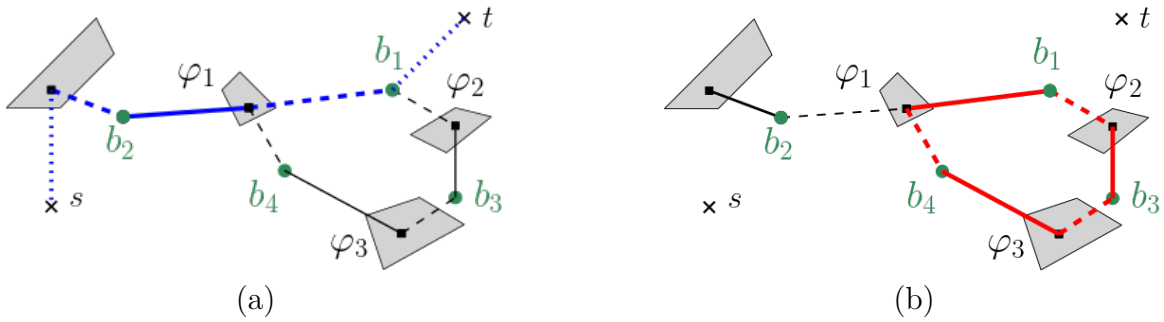


Figure 4.4: (a) An arbitrary admissible augmenting path $\langle s, b_1, r_{\varphi_1}, b_2 \rangle$ (blue path), and (b) an admissible cycle $\langle b_1, r_{\varphi_2}, b_3, r_{\varphi_3}, b_4, r_{\varphi_1} \rangle$ (red cycle) formed after augmentation.

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 $\overleftarrow{\mathcal{G}_\delta}$, add r as r_{i+1} to Q. Otherwise, mark b_i as visited and remove b_i from U and Q.
2. If $Q = \langle t = b_0, r_1, b_1, \ldots, b_i, r_{i+1} \rangle$ for some $i \geq 0$, let $b := \arg\min_{b' \in U} \mathrm{c}_y(r_{i+1}, b')$ denote the unvisited point with the minimum weighted distance to r_{i+1}. 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.
   (a) If (b, r_{i+1}, b_i) is admissible, i.e., $\mathrm{c}_y(r_{i+1}, b) < \mathrm{c}_y(r_{i+1}, b_i)$, 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

SEARCHANDAUGMENT procedure.

Lemma 4.6. *Suppose invariants (I1) and (I2) hold at the start of the* SEARCHANDAUGMENT *procedure. Then, after the execution of the* SEARCHANDAUGMENT *procedure,*

(S1) the transport plan $\hat{\tau}_\delta, y(\cdot)$ remains 2δ-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.

4.2.2 INCREASEWEIGHTS Procedure

After the execution of the SEARCHANDAUGMENT procedure, there are no admissible augmenting paths remaining 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 4.5(a), 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 4.5(b), 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 $\mathrm{c}_y(r, b_1) \geq \mathrm{c}_y(r, b_2) \geq \ldots \mathrm{c}_y(r, b_k)$. Mark all points $b \in B$ and all forward edges (b, r) as

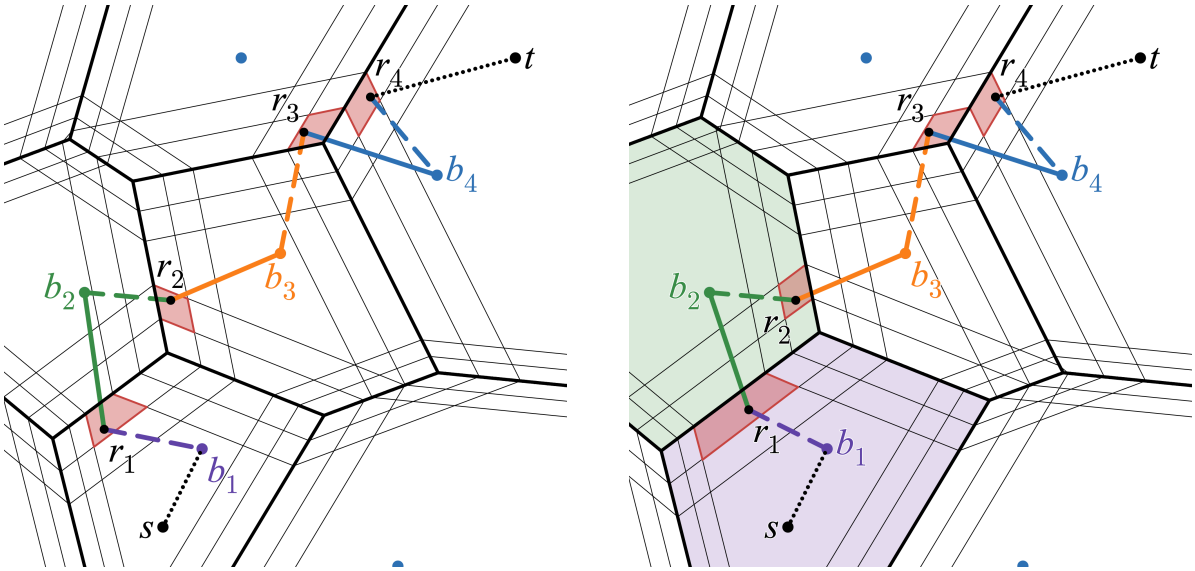


Figure 4.5: (a) After the execution of the SEARCHANDAUGMENT procedure, there are no admissible augmenting paths in $\mathcal{G}_\delta$, and (b) by increasing the weights of the points that are reachable from s by augmenting paths (points b_1 and b_2), new admissible triples are created (e.g. (b_2, r_2, b_3)), which might lead to the formation of admissible augmenting paths.

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.

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 and Q, and add b_i to $\mathcal{K}$.

3. If $Q = \langle s, b_1, r_1, \ldots, b_i, r_i \rangle$ for some $i \geq 1$, let $b := \arg\min_{b' \in U \cap \mathcal{N}(r)} \mathrm{c}_y(r_i, b')$ 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., $c_y(r_i, b) > c_y(r_i, b_i)$, then add b as b_{i+1} to Q.

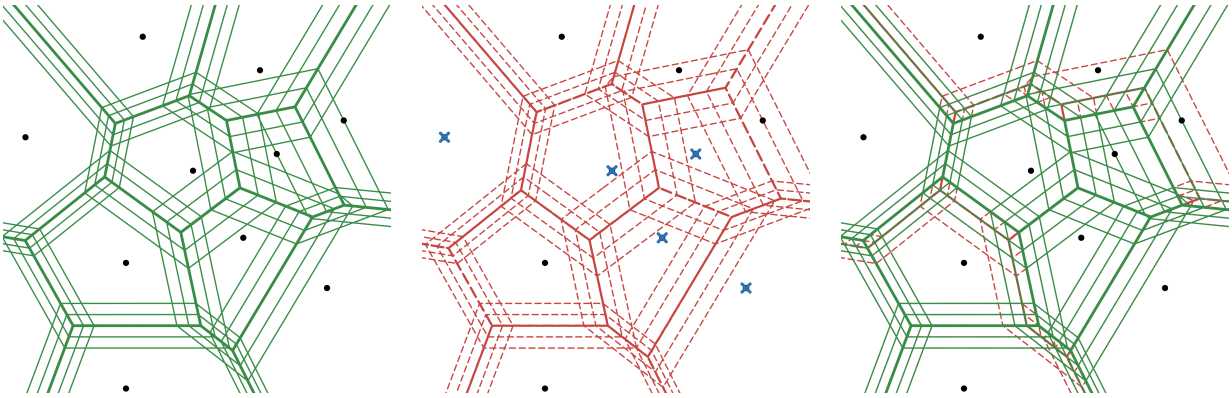
(b) Otherwise, remove r_i from Q and mark (b_i, r_i) as visited.

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. $\mathcal{X}'_\delta$). See Figures 4.6(a) and (b). 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 4.6(c). 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 set $Y \times B$ 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



(a) (b) (c)

Figure 4.6: (a) The partitioning $\mathcal{X}'_\delta$ before updating weights, (b) 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 (c) the combined partitioning $\mathcal{Y}$.

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 $A_\delta \times B$.

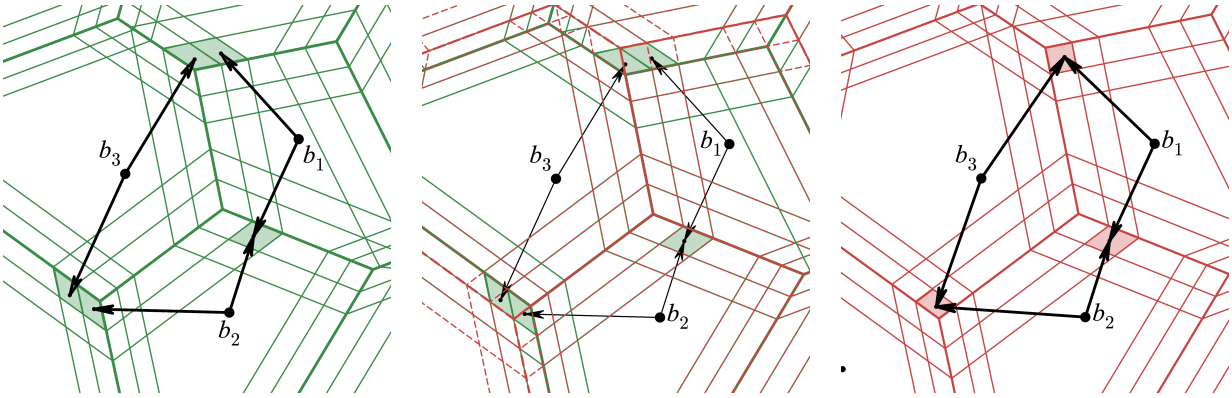
Finally, for each region $\varphi \in \mathcal{X}_\delta$, our algorithm stores a list $\mathcal{N}(r_\varphi)$ of all points $b \in B$ with $r_\varphi \in \mathrm{Vor}_b^{2\delta}$, sorted in increasing order of their weights distance to r_φ, i.e., $\mathcal{N}(r_\varphi) = \langle b_1, \ldots, b_k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

Lemma 4.7. *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 2δ-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.



(a) (b) (c)

Figure 4.7: An example of a cycle in a transport plan that is created while increasing weights: (a) a transport plan that is a forest, (b) the new Voronoi diagram and partitioning (red dashed lines) after increasing the weight of b_1, b_2, and b_3, and (c) a cycle formed in the new residual graph.

4.2.3 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 $\mathcal{G}_\delta$. See Figures 4.7 and 4.8 for examples of a cycle in the transport and an admissible cycle formed after the execution of the INCREASEWEIGHTS procedure. 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 [149] to make the transport plan $\hat{\tau}_\delta$ a forest. Then, execute a partial DFS search from 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

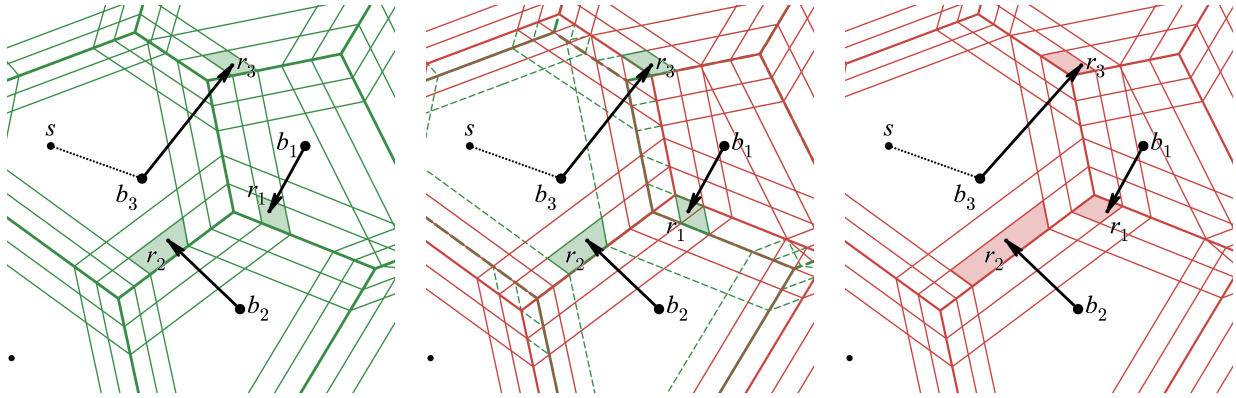


Figure 4.8: An example of the formation of admissible cycles due to updating the weights and the residual graph: (a) a transport plan with no admissible cycles in the corresponding residual graph, (b) the new Voronoi diagram and partitioning (red lines) after increasing the weights of b_3 and b_2, and (c) an admissible cycle $\langle b_1, r_1, b_2, r_2, b_3, r_3 \rangle$ formed in the new residual graph.

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. 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.

Acyclifying the Transport Plan

Similar to the Acyclify procedure introduced in [146, 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 $\hat{\tau}'_{k-1}$ 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 [149], 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.

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 $b := \arg\min_{b' \in U} \mathrm{c}_y(r_i, b')$ be the unvisited

point with the minimum weighted distance to r_i.

(a) If (b, r_i, b_i) is admissible, i.e., $\mathrm{c}_y(r_i, b) < \mathrm{c}_y(r_i, b_i)$,

- 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 $\mathrm{bc}(C) := \min_{t \in [j,i]} \hat{\tau}_\delta(r_t, b_t)$ 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 $\hat{\tau}_\delta(r_t, b_{t+1}) \leftarrow \hat{\tau}_\delta(r_t, b_{t+1}) + \mathrm{bc}(C)$ (resp. $\hat{\tau}_\delta(r_t, b_t) \leftarrow \hat{\tau}_\delta(r_t, b_t) - \mathrm{bc}(C)$). 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 4.8. *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 2δ-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.

4.3 Analysis

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

4.3.1 Correctness Analysis

For any scale δ, the initial transport plan $\hat{\tau}_\delta$ is empty. Therefore, $\hat{\tau}_\delta$ along with the weights $y(\cdot)$ is 2δ-feasible. By properties (S1), (W1), and (A1), the transport plan $\hat{\tau}_\delta, y(\cdot)$ remains 2δ-feasible in each iteration of our algorithm, and therefore, invariant (I1) holds. The invariant (I2) is a direct consequence of property (A2) in Lemma 4.8.

From Invariant (I1), in each scale δ, our algorithm maintains a 2δ-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 2δ-optimal. From Lemma 4.4, one can convert $\hat{\tau}_\delta$ into a 2δ-optimal semi-discrete transport plan τ_δ. Given that our algorithm terminates when $\delta \leq \varepsilon/2$, from Lemma 2.4, the transport plan returned by our algorithm is ε-close, as desired.

4.3.2 Efficiency Analysis

Next, we show that the running time of our algorithm is $O(n^3(\mathrm{Q} + n \log n) \log \frac{C_{\max}}{\varepsilon})$, proving Theorem 2.2. Recall that in 2 dimensions, as shown in Lemma 4.2, the residual graph has $O(n^2)$ vertices and $O(n^3)$ edges. For d-dimensional distributions, we show in Lemma A.4 in the appendix that in d dimensions, the residual graph has $O(n^d)$ vertices and $O(n^{d+1})$ edges. Using these bounds, we analyze the running time of each of our three procedures in the following.

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 add it to the search path again in the same execution. Upon finding an augmenting path P, the procedure augments the transport plan along P in $O(|P|)$ time. Let $\{P_1, \ldots, P_k\}$ 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. $O(n^{d+1}$ edges), the running time of the procedure would be $O(n^3 + \sum_{i=1}^{k} |P_i|)$ (resp. $O(n^{d+1} + \sum_{i=1}^{k} |P_i|)$). Combining with Lemma 4.9 below, each execution of the SearchAndAugment procedure takes $O(n^3)$ time in 2 dimensions and $O(n^{d+1})$ time in d dimensions.

Lemma 4.9. *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 Sketch. Recall that the bottleneck capacity of an augmenting path is determined as the minimum of the excess of the two free endpoints of the path and the amount of mass transported along the backward edges of the path. Therefore, augmenting the transport plan along an augmenting path either removes a backward edge from the residual graph or results in a transport plan that fully transports the mass of a point that was free prior to augmentation. To prove this lemma, we use property (I2) to argue that there are at most $O(n^2)$ backward edges in the residual graph and $O(n^2)$ free points to show that the number of augmenting paths is at most $O(n^2)$ in each execution of the SearchAndAugment procedure. Since each augmenting path has a length of at most $2n$, the total length of all augmenting paths found in each execution of the SearchAndAugment procedure would be $O(n^3)$. For d-dimensional distributions, we follow the same argument to show that the SearchAndAugment procedure finds $O(n^d)$ augmenting paths, where each one has a length of at most $2n$. See Appendix A.2.3 for details. □

Efficiency of the IncreaseWeights Procedure

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. $O(n^{d+1})$) in 2 dimensions (resp. d dimensions). 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 $\mathcal{X}'_\delta$). Similar to our proof for the size of the graph, 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 $\hat{\tau}$, therefore, can be done in $O(n^2(\mathrm{Q} + n))$ (resp. $O(n^d(\mathrm{Q} + n))$) time since

(1) the mass of all regions in $\mathcal{Y}$ can be determined in $O(n^2\mathrm{Q})$ (resp. $O(n^d\mathrm{Q})$) time. To do so, we partition the regions in $\mathcal{Y}$ into constant complexity regions (e.g., triangles), which will still be an arrangement with $O(n^2)$ (resp. $O(n^d)$) vertices and $O(n^2)$ (resp. $O(n^d)$) regions, and use the Oracle to compute the amount of continuous mass inside each region, 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, since 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 $O(n^2(\mathrm{Q} + n \log n))$ (resp. $O(n^d(\mathrm{Q} + n \log n))$) time.

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 [149], 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 add it again to the search path in the same execution. Furthermore, upon finding an admissible cycle C, it cancels the cycle in $O(|C|)$ time. Let $\{C_1, \ldots, C_k\}$ denote the set of all cycles found in the execution of the Acyclify procedure. 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 $O(n^3 + \sum_{i=1}^{k} |C_i|)$ (resp. $O(n^{d+1} + \sum_{i=1}^{k} |C_i|)$) time. The following lemma bounds the total length of all cycles found by the Acyclify procedure.

Lemma 4.10. *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.*

The proof of Lemma 4.10 is similar to that of Lemma 4.9 and is included in Appendix A.2.3. Using Lemma 4.10, the total execution time of the Acyclify procedure is $O(n^3)$ in 2 dimensions and $O(n^{d+1})$ in d dimensions.

Combining the running times of all three procedures, each iteration of step 2 of our algorithm takes $O(n^2(\mathrm{Q} + n \log n))$ time. In the following lemma, we show that in each scale, the total number of iterations of step 2 is at most $O(n)$.

Lemma 4.11. *For each scale δ, the second step of our algorithm runs $O(n)$ iterations.*

Proof Sketch. Let $\tau_{2\delta}, y_{2\delta}(\cdot)$ denote the 4δ-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 4δ-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 2δ-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 4.8), we show that for the free point b_k in P, $y_\delta(b_k) - y_{2\delta}(b_k) \leq 6n\delta$. Our algorithm increases the weight of b_k by δ in each iteration (property (W2) in Lemma 4.8), 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 A.2.3. □

Using Lemma 4.11, the total time spent in step 2 in each scale of our algorithm is $O(n^3(\mathrm{Q} + n \log n))$. Since there are $O(\log \frac{C_{\max}}{\varepsilon})$ scales, the overall runtime is $O(n^3(\mathrm{Q}+n \log n) \log \frac{C_{\max}}{\varepsilon})$, thereby proving Theorem 2.2.

4.4 Applications to the Discrete OT 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 $O(k^3(\sqrt{N} + k \log k) \log \frac{C_{\max}}{\varepsilon})$ 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 $\mathrm{Q} = O(\sqrt{N})$ time [158]. Given a query discrete distribution ν, one can use our algorithm from Section 4.2 in conjunction with the **RS-DS** to compute an ε-close transport plan between μ and ν in $O(k^3(\sqrt{N} + k \log k) \log \frac{C_{\max}}{\varepsilon})$ time leading to the following theorem.

Theorem 4.12. *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 OT plan between μ and ν under the squared Euclidean distances*

can be computed in $O(k^{d+1}(N^{1-1/d} + k \log k) \log \frac{C_{\max}}{\varepsilon})$ *time, where* $C_{\max}$ *is the diameter of* $A \cup B$.

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 the 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 4.13. *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.*

Chapter 5

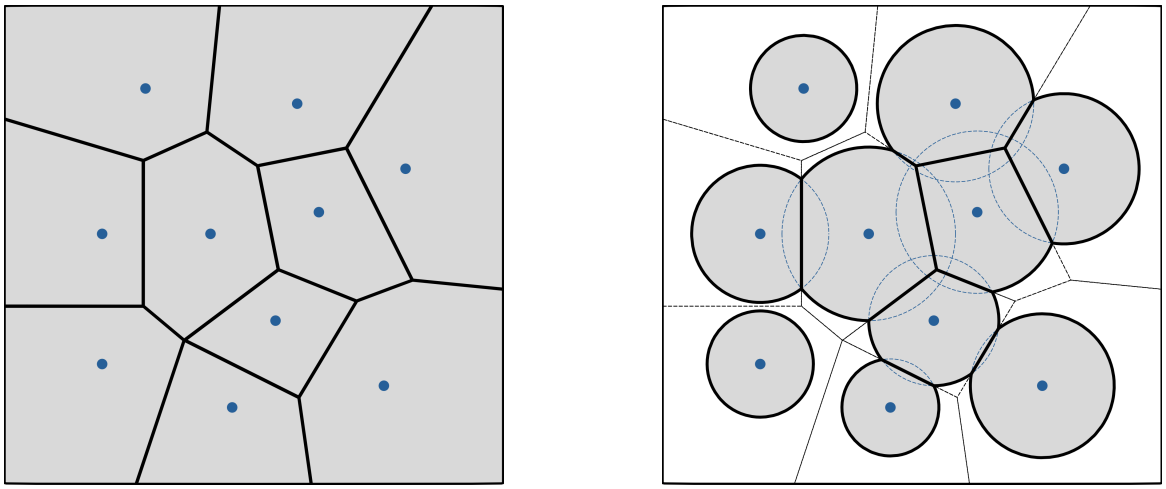
An Efficient Algorithm for the Robust Semi-Discrete OT Problem

In this chapter, we introduce a characterization for the λ-robust optimal transport problems in the semi-discrete settings. We then use this characterization and design a combinatorial algorithm for computing an ε-close semi-discrete λ-robust optimal transport plan.

5.1 Characterizing Robust Semi-Discrete Optimal Transport

In this section, we provide a characterization of partial and robust semi-discrete OT using weighted Voronoi diagrams. We begin by giving a formal definition of the variants of OT.

Partial and Robust OT. Suppose μ is a continuous probability distribution defined over bounded support $A \subset \mathbb{R}^d$ and ν is a discrete distribution with support set B of n points in $\mathbb{R}^d$. Recall that for a parameter $\alpha \in [0, 1]$, a transport plan τ is α-partial if τ transports α mass, i.e., $\mathcal{M}(\tau) = \alpha$, and τ is an α-OPT plan if it has a minimum cost among all α-partial transport plans. Furthermore, recall that for a parameter $\lambda > 0$, the λ-robust cost of a transport plan τ is defined as $w_\lambda^{\mathrm{ROT}}(\tau) := w(\tau) + (1 - \mathcal{M}(\tau))\lambda$, and a λ-ROT plan is defined



(a) (b)

Figure 5.1: (a) Voronoi cells, and (b) restricted Voronoi cells

as a transport plan τ with the minimum λ-robust cost. Next, we restate the definitions of restricted Voronoi cells.

Suppose $y(\cdot)$ denotes a set of non-negative weights for the points in B. Recall that $\mathrm{c}_y(a,b) := \mathrm{c}(a,b) - y(b)$ denotes the weighted distance between a and b and $\mathrm{Vor}_y(b)$ denotes the Voronoi cell of b with weights $y(\cdot)$. See Figure 5.1(a). It is well known that for any set of weights $y(\cdot)$ for B such that $\mu(\mathrm{Vor}_y(b)) = \nu(b)$ for all points $b \in B$, the transport plan that transports the mass of each point $b \in B$ to the mass of μ inside $\mathrm{Vor}_y(b)$ is an optimal transport plan. Furthermore, using the LP formulation of semi-discrete OT and the strong LP duality, one can prove the existence of such weights; see [22, 49, 76].

For any point $b \in B$, recall that the restricted Voronoi cell of b captures the points of the Voronoi cell of b that are within a distance of $y(b)$ from b, i.e.,

$$\mathrm{ResVor}_y(b) := \{a \in \mathrm{Vor}_y(b) \mid \mathrm{c}(a,b) \leq y(b)\}.$$

See Figure 5.1(b). We refer to the restricted Voronoi cell of any point $b \in B$ as *balanced* (resp.

surplus, *deficit*) if $\mu(\text{ResVor}_y(b)) = \nu(b)$ (resp. $\mu(\text{ResVor}_y(b)) < \nu(b)$, $\mu(\text{ResVor}_y(b)) > \nu(b)$).

Valid weights, cover, and cap. A weight function $y : B \to \mathbb{R}_{\geq 0}$ is called *valid* if for all points $b \in B$, the restricted Voronoi cell of b is either balanced or surplus and the points with a surplus cell have the maximum weight, i.e., for all points $b \in B$ with surplus restricted Voronoi cell, $y(b) = \max_{b' \in B} y(b')$. For a valid weight function y, we define the transport plan induced by y, denoted by $\tau_y : A \times B \to \mathbb{R}_{\geq 0}$, as the one that for each point $b \in B$, transports the mass of b to the continuous mass of μ inside the restricted Voronoi cell $\text{ResVor}_y(b)$. Since each point $b \in B$ has a surplus or balanced restricted Voronoi cell, there is enough mass at b to transport to $\text{ResVor}_y(b)$. We define the *cover* of $y(\cdot)$, denoted by $\text{Cover}(y)$, to be the total amount of continuous mass inside the restricted Voronoi cells of all points in B, i.e.,

$$\text{Cover}(y) := \sum_{b \in B} \mu(\text{ResVor}_y(b)).$$

Additionally, define the *cap* of a weight function $y(\cdot)$, denoted by $\text{Cap}(y)$, to be the maximum weight assigned by $y(\cdot)$ to the points in B, i.e.,

$$\text{Cap}(y) := \max_{b \in B} y(b).$$

Characterization of robust OT. Next, we characterize a λ-robust optimal transport plan using a set of valid weights with a cap of λ.

Lemma 5.1 (λ-ROT Characterization)**.** *Let μ be a continuous distribution over compact support $A \subset \mathbb{R}^d$, ν be a discrete distribution over a set B in n points in $\mathbb{R}^d$, and $\lambda > 0$ be a parameter. There exists a valid weight function $y : B \to \mathbb{R}_{\geq 0}$ with $\text{Cap}(y) = \lambda$ such that the transport plan induced by $y(\cdot)$ is a λ-robust optimal transport plan. Furthermore, this property holds for any valid weight function y with $\text{Cap}(y) = \lambda$.*

Proof. The primal linear optimization problem for λ-robust optimal transport is as follows.

$$\begin{aligned}
&\min_{\tau} \sum_{b\in B} \int_A \tau(a,b)\mathrm{c}(a,b)\,da + \lambda\left(1 - \sum_{b\in B}\int_A \tau(a,b)\,da\right)\\
&\text{s.t.} \quad \int_A \tau(a,b)da \le \nu(b) \quad \forall b \in B, && (5.1)\\
&\qquad \sum_{b\in B} \tau(a,b) \le \mu(a) \quad \forall a \in A, && (5.2)\\
&\qquad \tau \ge 0.
\end{aligned}$$

The corresponding dual linear optimization problem is

$$\begin{aligned}
&\max_{\phi,\psi} \quad -\sum_{b\in B}\phi(b)\nu(b) - \int_A \psi(a)\mu(a)da\\
&\text{s.t.} \quad -\phi(b) - \psi(a) \le \mathrm{c}(a,b) - \lambda \quad \forall a \in A, b \in B, && (5.3)\\
&\qquad \phi, \psi \ge 0.
\end{aligned}$$

Define $y(b) := \lambda - \phi(b)$. Then, since λ is a given parameter and the total mass at the points of B is 1, the dual problem can be rewritten as

$$\begin{aligned}
&\max_{y,\psi} \quad \sum_{b\in B} y(b)\nu(b) - \int_A \psi(a)\mu(a)da - \lambda\\
&\text{s.t.} \quad y(b) - \psi(a) \le \mathrm{c}(a,b) \quad \forall a \in A, b \in B, && (5.4)\\
&\qquad y(b) \le \lambda, \quad \forall b \in B, && (5.5)\\
&\qquad \psi \ge 0.
\end{aligned}$$

First, note that by strong duality of primal-dual linear programs, the optimal choice of dual variables $y(\cdot)$ exists. Furthermore, by the complementary slackness property, for any pair $(a,b) \in A \times B$ with $\tau(a,b) > 0$, the corresponding dual constraint (5.3) has to be tight and

$y(b) - \psi(a) = \mathrm{c}(a, b)$. Since $\psi(a) \geq 0$, we have

$$y(b) - \mathrm{c}(a, b) = \psi(a) \geq 0.$$

Therefore, the point a does not lie outside of $\mathrm{ResVor}_y(b)$. Furthermore, for any point $a \in A$ whose mass is not fully transported by τ, we have $\sum_{b \in B} \mu(a, b) < \mu(a)$ and from the complementary slackness property, its dual variable is zero, i.e., $\psi(a) = 0$. However, using constraint (5.4), for any point $b \in B$,

$$y(b) - \mathrm{c}(a, b) \leq \psi(a) = 0,$$

and the point a does not lie inside the restricted Voronoi cell of b. Consequently, all points $a \in A$ whose mass is not fully transported by τ have a zero dual weight and lie outside of all restricted Voronoi cells, while the points $a \in A$ whose mass is transported by some points $b \in B$ lie on the boundary or inside the restricted Voronoi cell of b.

Hence, for any optimal solution $y(\cdot), \psi(\cdot)$ to the dual formulation of our linear program, the corresponding optimal solution τ to the primal linear program is a transport plan that transports the mass of each point $b \in B$ to the continuous mass inside its restricted Voronoi cell, i.e., τ is a transport plan induced by the weights $y(\cdot)$.

For any point $b \in B$ whose mass is not fully transported (i.e., $\int_A \tau(a, b) da < \nu(b)$), the constraint (5.1) is not tight. Consequently, from the complementary slackness property, the corresponding dual variable has to be zero (i.e., $\phi(b) = 0$) and $y(b) = \lambda - \phi(b) = \lambda$. Since $\lambda \geq \max_{b' \in B} y(b')$ (from constraint (5.5)), we conclude that for any free point $b \in B$ (i.e., the points $b \in B$ with a surplus restricted Voronoi cell), $y(b) = \max_{b' \in B} y(b')$ and $\mathrm{Cap}(y) := \max_{b \in B} y(b) = \lambda$.

Note that if the mass of all points in B is fully transported (i.e., $\int_A \tau(a,b)da = \nu(b)$ for all $b \in B$), then we can simply shift the dual weights to obtain a set of weights with a cap of λ. To do so, let $y_{\max} := \max_{b\in B} y(b)$ and define the new dual weights $y'(b) := y(b) + (\lambda - y_{\max})$ for each point $b \in B$ and $\psi'(a) := \psi(a) + (\lambda - y_{\max})$ for each $a \in A$. It follows that y', ψ' are feasible and the dual objective value remains the same, since both μ and ν are probability distributions and have an equal amount of mass. Furthermore, $\mathrm{Cap}(y') := \max_{b\in B} y'(b) = \lambda$, as desired.

We finally show that for any valid weight function $y(\cdot)$ for B where $\mathrm{Cap}(y) = \lambda$ and all free points $b \in B$ have a maximum weight $y(b) = \lambda$, the transport plan τ induced by $y(\cdot)$ would be a λ-robust optimal transport plan. Define a weight function $\psi : A \to \mathbb{R}$ that assigns

$$\psi(a) := \max\{0, \max_{b\in B} y(b) - \mathrm{c}(a,b)\}$$

to each point $a \in A$. Let $\alpha = \mathrm{Cover}(y)$. Additionally, define $\nu_f^\tau(b) := \nu(b) - \int_A \tau(a,b)da$ and $\mu_f^\tau(a) := \mu(a) - \sum_{b\in B} \tau(a,b)$ as the free mass of b and a with respect to transport plan τ, respectively. Using the assumption that τ is induced by y, we can rewrite the robust cost of τ as:

$$\begin{aligned} w_\lambda^{\mathrm{ROT}}(\tau) &= w(\tau) + \lambda(1 - \mathcal{M}(\tau)) \\ &= \sum_{b\in B} \int_A \mathrm{c}(a,b)\,\tau(a,b)\,da + \lambda\,(1-\alpha) \\ &= \sum_{b\in B} \int_A (y(b) - \psi(a))\tau(a,b)\,da + \lambda\,(1-\alpha) \\ &= \sum_{b\in B} y(b)\nu(b) - \int_A \psi(a)\,d\mu(a) - \sum_{b\in B_f} y(b)\nu_f^\tau(b) + \int_{A_f} \psi(a)\,d\mu_f^\tau(a) + \lambda(1-\alpha). \end{aligned} \tag{5.6}$$

Since all free points $b \in B$ have a weight $y(b) = \lambda$ and that the total amount of mass that

is not transported by τ is $(1-\alpha)$, we have $\sum_{b\in B_f} y(b)\nu_f^\tau(b) = \lambda(1_\alpha)$. Furthermore, since all points $a \in A$ whose mass is not fully transported by τ are outside the restricted Voronoi cells of the points in B, $\psi(a) = 0$ and $\int_{A_f} \psi(a)\, d\mu_f^\tau(a) = 0$. Plugging into Equation (5.6),

$$\begin{aligned} w_\lambda^{\mathrm{ROT}}(\tau) &= \sum_{b\in B} y(b)\nu(b) - \int_A \psi(a)\, d\mu(a) - \sum_{b\in B_f} y(b)\nu_f^\tau(b) + \int_{A_f} \psi(a)\, d\mu_f^\tau(a) + \lambda(1-\alpha) \\ &= \sum_{b\in B} y(b)\nu(b) - \int_A \psi(a)\, d\mu(a) - \lambda(1-\alpha) + 0 + \lambda(1-\alpha) \\ &= \sum_{b\in B} y(b)\nu(b) - \int_A \psi(a)\, d\mu(a). \end{aligned}$$

Let τ^* denote any optimal λ-robust optimal transport plan between μ and ν. Define $\alpha^* = \mathcal{M}(\tau^*)$. Then, by the definition of dual weights for points in A,

$$\begin{aligned} w_\lambda^{\mathrm{ROT}}(\tau^*) &= \sum_{b\in B} \int_A \mathrm{c}(a,b)\, \tau^*(a,b)\, da + \lambda\,(1 - \mathcal{M}(\tau^*)) \\ &\geq \sum_{b\in B} \int_A (y(b) - \psi(a))\tau^*(a,b)\, da + \lambda(1-\alpha^*) \\ &= \sum_{b\in B} y(b)\nu(b) - \int_A \psi(a)\, d\mu(a) - \sum_{b\in B_f} y(b)\nu_f^{\tau^*}(b) + \int_{A_f} \psi(a)\, d\mu_f^{\tau^*}(a) + \lambda(1-\alpha^*). \end{aligned} \tag{5.7}$$

Note that $\psi(a) \geq 0$ for any point $a \in A$ and $y(b) \leq \lambda$ for any point $b \in B$. Plugging into Equation (5.7),

$$\begin{aligned} w_\lambda^{\mathrm{ROT}}(\tau^*) &\geq \sum_{b\in B} y(b)\nu(b) - \int_A \psi(a)\, d\mu(a) - \sum_{b\in B_f} y(b)\nu_f^{\tau^*}(b) + \int_{A_f} \psi(a)\, d\mu_f^{\tau^*}(a) + \lambda(1-\alpha^*) \\ &\geq \sum_{b\in B} y(b)\nu(b) - \int_A \psi(a)\, d\mu(a) - \lambda(1-\alpha^*) + \lambda(1-\alpha^*) \\ &= \sum_{b\in B} y(b)\nu(b) - \int_A \psi(a)\, d\mu(a), \end{aligned}$$

Combining the two bounds,

$$w_\lambda^{\mathrm{ROT}}(\tau) = \sum_{b\in B} y(b)\nu(b) - \int_A \psi(a)\, d\mu(a) \le w_\lambda^{\mathrm{ROT}}(\tau^*).$$

Since τ^* is a λ-robust OT plan, we conclude that τ is also a λ-robust OT plan. □

5.2 Our Combinatorial Framework

We begin by introducing the notion of a δ-restricted-feasible transport plan. Given a weight function $y(\cdot)$ on points of B and a parameter $\delta \ge 0$, the *δ-expanded restricted Voronoi cell* of b, denoted by $\mathrm{ResVor}_y^\delta(b)$ is defined as

$$\mathrm{ResVor}_y^\delta(b) := \{a \in A \mid \mathrm{c}(a,b) \le y(b) + \delta \text{ and } \mathrm{c}_y(a,b) \le \mathrm{c}_y(a,b') + \delta, \forall b' \in B\}.$$

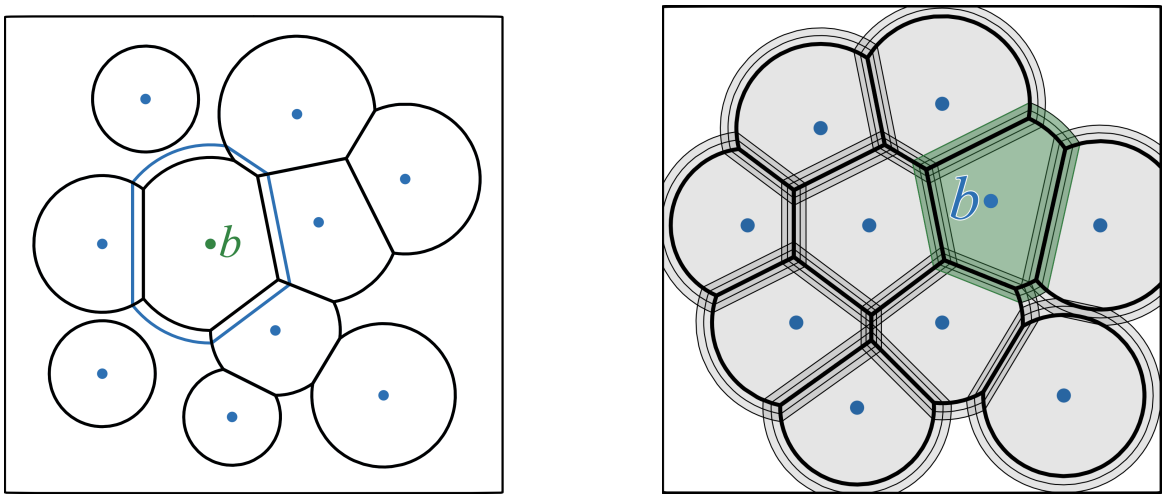
See Figure 5.2(a). The δ-expanded restricted Voronoi cell of b can be seen as the intersection of the δ-expanded Voronoi cell of b with a ball of radius $y(b) + \delta$ centered at b.

Analogous to the δ-feasibility defined in previous chapters, we next define δ-restricted-feasible transport plans. Any transport plan τ along with weights $y(\cdot)$ is *δ-restricted-feasible* if,

(F2) for any pair $(a,b) \in A \times B$ with $\tau(a,b) > 0$, the point a lies inside the 2δ-expanded restricted Voronoi cell $\mathrm{ResVor}_y^{2\delta}(b)$,

See Figure 5.2(b). We refer to any point $b \in B$ (resp. $a \in A$) as *surplus* (resp. *deficit*) if $\int_A \tau(a,b)\, da < \nu(b)$ (resp. $\sum_{b\in B} \tau(a,b) < \mu(a)$) and as *balanced* if $\int_A \tau(a,b)\, da = \nu(b)$ (resp. $\sum_{b\in B} \tau(a,b) = \mu(a)$).

Given a weight function $y(\cdot)$ for the points in B, we derive a weight $\bar{y}(a)$ for each point



(a) (b)

Figure 5.2: (a) The δ-expanded restricted Voronoi cell of $\mathrm{ResVor}_y^{\delta}(b)$, and (b) in a δ-restricted-feasible transport plan, the point $b \in B$ can transport mass to its 2δ-expanded restricted Voronoi cell.

$a \in A$, referred to by the *derived weight* of a, as

$$\bar{y}(a) := \max\{0, \max_{b \in B}\{y(b) - \mathrm{c}(a,b)\}\}. \tag{5.8}$$

Note that any 0-restricted-feasible transport plan transports the mass of each point $b \in B$ to regions inside its restricted Voronoi cell. From Lemma 5.1, a λ-robust optimal transport plan is a 0-restricted-feasible transport plan where (i) $\mathrm{Cap}(y) = \lambda$, (ii) all surplus points $b \in B$ have $y(b) = \lambda$, and (iii) all deficit points $a \in A$ have $\bar{y}(a) = 0$. The following lemma shows that any δ-restricted-feasible transport plan with similar conditions on surplus and deficit points is a 2δ-close λ-robust transport plan.

Lemma 5.2. *For any parameter $\delta > 0$, suppose $\tau, y(\cdot)$ denotes a δ-restricted-feasible transport plan, and let $\lambda := \mathrm{Cap}(y)$. Suppose*

(F3) for every deficit point $a \in A$, $\bar{y}(a) = 0$, and

(F4) for all surplus points $b \in B$, $y(b) = \lambda$.

Then τ is a 2δ-close λ-robust transport plan.

Proof. For any point $a \in A$, suppose $b_a \in B$ denotes the weighted nearest neighbor of a with respect to weights $y(\cdot)$, i.e., $b_a := \arg\min_{b\in B} \mathrm{c}_y(a,b)$. For any transport plan τ', let $\mu^{\tau'}$ (resp. $\nu^{\tau'}$) denote the sub-measure of μ (resp. ν) that is transported by τ', i.e., the part of the mass of μ (resp. ν) that is transported by τ'. Let

$$\mu_f^{\tau'} := \mu - \mu^{\tau'} \qquad \text{and} \qquad \nu_f^{\tau'} := \nu - \nu^{\tau'}$$

denote the sub-measures of μ and ν that are not transported by τ', i.e., $\mu_f^{\tau'}$ and $\nu_f^{\tau'}$ are the untransported part of the mass of μ and ν, respectively. We can rewrite the λ-robust cost of the transport plan τ as

$$\begin{aligned} w_\lambda^{\mathrm{ROT}}(\tau) = w(\tau) + \lambda(1-\mathcal{M}(\tau)) &= \sum_{b\in B}\int_A \mathrm{c}(a,b)\tau(a,b)\,da + \lambda(1-\mathcal{M}(\tau)) \\ &= \sum_{b\in B}\int_A (\mathrm{c}(a,b) - y(b) + \bar{y}(a))\tau(a,b)\,da + \lambda(1-\mathcal{M}(\tau)) \\ &\qquad + \sum_{b\in B} y(b)\nu^\tau(b) - \int_A \bar{y}(a)\mu^\tau(a)\,da. \end{aligned} \tag{5.9}$$

Note that from property (F4), any point $b \in B$ with $\nu_f^\tau(b) > 0$ (i.e., any surplus point $b \in B$) has a weight $y(b) = \lambda$; therefore,

$$\sum_{b\in B} y(b)\nu^\tau(b) = \sum_{b\in B} y(b)\nu(b) - \sum_{b\in B} y(b)\nu_f^\tau(b) = \sum_{b\in B} y(b)\nu(b) - \lambda(1-\mathcal{M}(\tau)). \tag{5.10}$$

Similarly, from property (F3), any point $a \in A$ with $\mu_f^\tau(a) > 0$ (i.e., any deficit point $a \in A$)

has a zero weight. Hence,

$$\int_A \bar{y}(a)\mu^\tau(a)\,da = \int_A \bar{y}(a)\mu(a)\,da - \int_A \bar{y}(a)\mu_f^\tau(a)\,da = \int_A \bar{y}(a)\mu(a)\,da. \tag{5.11}$$

Finally, due to the δ-restricted-feasibility of $\tau, y(\cdot)$, the point a lies inside the 2δ-expanded restricted Voronoi cell of b, and therefore, by increasing the weights of b by 2δ, the point b becomes the WNN of a and also contains a in its dual disc, i.e., $\bar{y}(a) \leq (y(b) + 2\delta) - \mathrm{c}(a, b)$. By rearranging this inequality,

$$\mathrm{c}(a, b) - y(b) + \bar{y}(a) \leq 2\delta. \tag{5.12}$$

Plugging Equations (5.10), (5.11), and (5.12) into Equation (5.9),

$$\begin{aligned}
w_\lambda^{\mathrm{ROT}}(\tau) &= \sum_{b\in B}\int_A (\mathrm{c}_y(a, b) + \bar{y}(a))\tau(a, b)\,da + \sum_{b\in B} y(b)\nu^\tau(b) - \int_A \bar{y}(a)\mu^\tau(a)\,da + \lambda(1 - \mathcal{M}(\tau)) \\
&= \sum_{b\in B}\int_A (\mathrm{c}_y(a, b) + \bar{y}(a))\tau(a, b)\,da + \sum_{b\in B} y(b)\nu(b) - \int_A \bar{y}(a)\mu(a)\,da \\
&\leq \sum_{b\in B}\int_A 2\delta\tau(a, b)\,da + \sum_{b\in B} y(b)\nu(b) - \int_A \bar{y}(a)\mu(a)\,da \\
&\leq 2\delta + \sum_{b\in B} y(b)\nu(b) - \int_A \bar{y}(a)\mu(a)\,da,
\end{aligned} \tag{5.13}$$

where the last inequality holds since τ transports at most 1 unit of mass, i.e., $\mathcal{M}(\tau) \leq 1$.

Next, let τ^* denote any λ-robust OT plan. We can rewrite the λ-robust cost of τ^* as

$$\begin{aligned} w_\lambda^{\mathrm{ROT}}(\tau^*) &= w(\tau^*) + \lambda(1 - \mathcal{M}(\tau^*)) = \sum_{b\in B} \int_A \mathrm{c}(a,b)\tau^*(a,b)\,da + \lambda(1 - \mathcal{M}(\tau^*)) \\ &= \sum_{b\in B} \int_A (\mathrm{c}(a,b) - y(b) + \bar{y}(a))\tau^*(a,b)\,da + \lambda(1 - \mathcal{M}(\tau^*)) \\ &\qquad + \sum_{b\in B} y(b)\nu^{\tau^*}(b) - \int_A \bar{y}(a)\mu^{\tau^*}(a)\,da. \end{aligned} \tag{5.14}$$

Since $\lambda = \mathrm{Cap}(y)$,

$$\sum_{b\in B} y(b)\nu^{\tau^*}(b) = \sum_{b\in B} y(b)\nu(b) - \sum_{b\in B} y(b)\nu_f^{\tau^*}(b) \geq \sum_{b\in B} y(b)\nu(b) - \lambda(1 - \mathcal{M}(\tau)). \tag{5.15}$$

Similarly, since all weights $\bar{y}(\cdot)$ are positive,

$$\int_A \bar{y}(a)\mu^{\tau^*}(a)\,da = \int_A \bar{y}(a)\mu(a)\,da - \int_A \bar{y}(a)\mu_f^{\tau^*}(a)\,da \leq \int_A \bar{y}(a)\mu(a)\,da. \tag{5.16}$$

Finally, by the definition of the derived weights, for any pair of points $(a,b) \in A \times B$,

$$\mathrm{c}(a,b) - y(b) + \bar{y}(a) \geq 0. \tag{5.17}$$

Plugging Equations (5.15), (5.16), and (5.17) into Equation (5.14),

$$\begin{aligned} w_\lambda^{\mathrm{ROT}}(\tau^*) &= \sum_{b\in B} \int_A (\mathrm{c}_y(a,b) + \bar{y}(a))\tau^*(a,b)\,da + \lambda(1 - \mathcal{M}(\tau^*)) \\ &\qquad + \sum_{b\in B} y(b)\nu^{\tau^*}(b) - \int_A \bar{y}(a)\mu^{\tau^*}(a)\,da \\ &\geq \sum_{b\in B} \int_A (\mathrm{c}_y(a,b) + \bar{y}(a))\tau^*(a,b)\,da + \sum_{b\in B} y(b)\nu(b) - \int_A \bar{y}(a)\mu(a)\,da \\ &\geq \sum_{b\in B} y(b)\nu(b) - \int_A \bar{y}(a)\mu(a)\,da, \end{aligned} \tag{5.18}$$

Combining Equations (5.13) and (5.18),

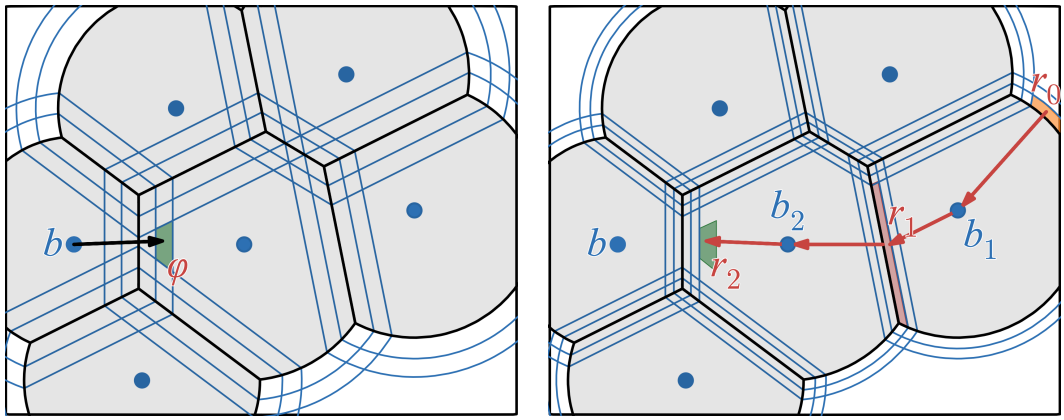
$$w_\lambda^{\mathrm{ROT}}(\tau) \leq 2\delta + \sum_{b\in B} y(b)\nu(b) - \int_A \bar{y}(a)\mu(a)\,da \leq w_\lambda^{\mathrm{ROT}}(\tau^*) + 2\delta,$$

and τ is a 2δ-close λ-ROT plan. □

Challenges in extending cost-scaling technique. Extending our combinatorial cost-scaling algorithm from Chapter 4 to the robust optimal transport problem creates new challenges. Unlike in the standard setting, our algorithm computes a δ-restricted-feasible transport plan that may leave some mass untransported. We must, therefore, cap the weights assigned to each point $b \in B$ by λ (Lemma 5.2 condition (F4)) and restrict the weights $\bar{y}(a)$ of all deficit regions in A to 0 (Lemma 5.2 condition (F3)). A naïve implementation of the scaling paradigm risks generating deficit regions within the restricted Voronoi cells – deficit points of A with positive weights – violating Lemma 5.2. For example, in Figure 5.3, the algorithm creates a deficit region (green region) inside the restricted Voronoi cells, illustrating such a violation. To resolve this, we introduce "consolidating paths", which carefully restructure the transport plan to relocate deficit regions from inside the restricted Voronoi cells to its exterior (Figure 5.3(b)).

Each scale of our algorithm consists of two steps. Step 1 updates the weights and the transport plan along a set of consolidating paths to restore condition (F3). Step 2 of our algorithm maintains δ-restricted-feasibility and condition (F3) as invariants, and carefully raises the weight of the surplus points to λ, enforcing condition (F4). From Lemma 5.2, the resulting transport plan is 2δ-close.

Decomposition of A. Given a δ-restricted-feasible transport plan $\tau, y(\cdot)$, let $\mathcal{X}_\delta$ denote all regions in the arrangement of the restricted Voronoi cell $\mathrm{ResVor}_y(b)$ and the δ- and 2δ-



(a) (b)

Figure 5.3: (a) In scale 2δ, the point b transports mass to a region φ inside its 4δ-expanded restricted Voronoi cell, and (b) in scale δ, the φ becomes a deficit region with a positive derived weight. A consolidating path $\langle r_0, b_1, r_1, b_2, r_2\rangle$ can be used to bring this deficit to the exterior of the restricted Voronoi cells.

expanded restricted Voronoi cells $\text{ResVor}_y^{\delta}(b)$ and $\text{ResVor}_y^{2\delta}(b)$ for all points $b \in B$. See Figure 5.4. For each region $\varphi \in \mathcal{X}_\delta$, pick an arbitrary representative point r_φ inside φ and assign it a mass of $\hat{\mu}(r_\varphi) := \mu(\varphi) = \int_\varphi \mu(a)\, da$. Let A_δ denote the set of representative points of all regions in $\mathcal{X}_\delta$. In this case, $\hat{\mu}$ is a (discrete) distribution over A_δ. We refer to any point $r_\varphi \in A_\delta$ as a deficit point if the corresponding region $\varphi \in \mathcal{X}_\delta$ is deficit with respect to τ. The point r_φ is *violating* if it is deficit and $\bar{y}(r_\varphi) > 0$.

In our algorithm, we store and represent a δ-restricted-feasible semi-discrete transport plan $\tau, y(\cdot)$ using a (discrete) transport plan $\hat{\tau}$ over $A_\delta \times B$, where for any point $b \in B$ and any region $\varphi \in \mathcal{X}_\delta$, we set $\hat{\tau}(r_\varphi, b) := \tau(\varphi, b)$. Note that any δ-restricted-feasible transport plan $\hat{\tau}, y(\cdot)$ over $A_\delta \times B$ can be converted back to a δ-restricted-feasible semi-discrete transport plan τ that arbitrarily transports a mass of $\hat{\tau}(r_\varphi, b)$ from b to φ for any pair $(\varphi, b) \in \mathcal{X}_\delta \times B$. We refer to $\hat{\tau}$ as an *implicit representation* of τ.

The following lemma shows that for any δ-restricted-feasible transport plan $\tau, y(\cdot)$, one can interchangeably use the implicit representation $\hat{\tau}$ without violating the δ-restricted-feasibility

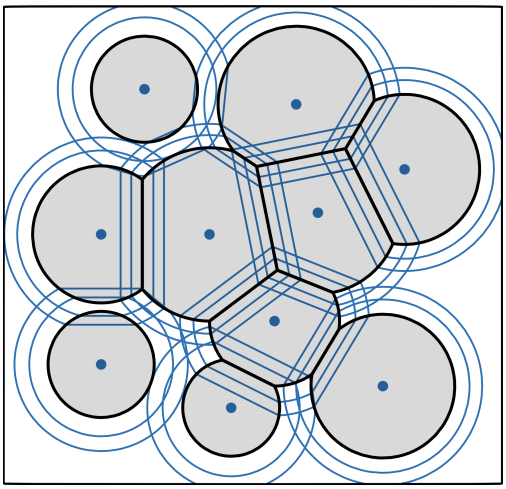


Figure 5.4: The decomposition $\mathcal{X}_\delta$.

conditions.

Lemma 5.3. *For any δ-restricted-feasible semi-discrete transport plan $\tau, y(\cdot)$ between μ and ν, the implicit representation $\hat{\tau}, y(\cdot)$ is also δ-restricted-feasible. Similarly, for any set of weights $y(\cdot)$, any δ-restricted-feasible transport plan $\hat{\tau}$ between $\hat{\mu}$ and ν can be converted into a δ-restricted-feasible semi-discrete transport plan $\tau, y(\cdot)$ between μ and ν.*

Proof. For each region $\varphi \in \mathcal{X}_\delta$, the representative point r_φ is inside φ. Furthermore, since $\mathrm{ResVor}_y^{2\delta}(b)$ is included in the set of cells used in the construction of $\mathcal{X}_\delta$, the region φ is either completely inside $\mathrm{ResVor}_y^{2\delta}(b)$ or is completely outside of it. Hence, due to the δ-restricted-feasibility of $\tau, y(\cdot)$, for any pair $(a, b) \in A \times B$ such that $\tau(a, b) > 0$ (since $a \in \mathrm{ResVor}_y^{2\delta}(b)$), the region $\varphi \in \mathcal{X}_\delta$ containing a also completely lies inside $\mathrm{ResVor}_y^{2\delta}(b)$ and $r_\varphi \in \mathrm{ResVor}_y^{2\delta}(b)$. Consequently, for any pair $(r, b) \in A_\delta \times B$ with $\hat{\tau}(r, b) > 0$, the point r lies inside the 2δ-expanded restricted Voronoi cell of b, and $\hat{\tau}, y(\cdot)$ is δ-restricted-feasible.

Next, suppose $\hat{\tau}$ is a δ-restricted-feasible transport plan between $\hat{\mu}$ and ν, i.e., for any pair $(r, b) \in A_\delta \times B$, if $\hat{\tau}(r, b) > 0$, then $r \in \mathrm{ResVor}_y^{2\delta}(b)$. As discussed above, for any region $\varphi \in \mathcal{X}_\delta$, if $r_\varphi \in \mathrm{ResVor}_y^{2\delta}(b)$, then φ would be completely inside $\mathrm{ResVor}_y^{2\delta}(b)$. Hence, if we

define a transport plan τ that transports, for each pair $(\varphi, b) \in \mathcal{X}_\delta \times B$, a total of $\hat{\tau}(r_\varphi, b)$ mass from the point b to the region φ (i.e., $\tau(\varphi, b) = \hat{\tau}(r_\varphi, b)$), then for any pair $(a, b) \in A \times B$ with $\tau(a, b) > 0$, then $a \in \mathrm{ResVor}_y^{2\delta}(b)$, and $\tau, y(\cdot)$ is δ-restricted-feasible. □

Residual Graph. Given a δ-restricted-feasible transport plan $\tau, y(\cdot)$, define a residual graph $\mathcal{G}_\delta$ over the point set $A_\delta \cup B$, where for each pair $(r_\varphi, b) \in A_\delta \times B$, we add a forward edge directed from b to r_φ if $r_\varphi \in \mathrm{ResVor}_y^{2\delta}(b)$. Additionally, if $\tau(\varphi, b) > 0$, we add a backward edge directed from r_φ to b. For any triple (b_1, r_φ, b_2) formed by a forward edge followed by a backward edge, the triple (b_1, r_φ, b_2) is *admissible* if $\mathrm{c}_y(r_\varphi, b_1) < \mathrm{c}_y(r_\varphi, b_2)$.

The following lemma, whose proof is provided in Appendix A.3, bounds the size of the residual graph.

Lemma 5.4. *For d-dimensional distributions, for any $d \geq 2$, the residual graph has $O(n^d)$ nodes and $O(n^{d+1})$ edges.*

Any directed path P in the residual graph is an *alternating path*. An alternating path P is *admissible* if all triples $(b_i, r_i, b_{i+1}) \in P$ are admissible. For any point $v \in A_\delta \cup B$, define the *residual capacity* of v, denoted by $\mathrm{rc}(v)$, as the amount of mass of v that is not transported by $\hat{\tau}$. For any alternating path P that starts at a point $s \in A_\delta \cup B$ and ends at a point $t \in A_\delta \cup B$, the *bottleneck capacity* of P, denoted by $\mathrm{bc}(P)$, is defined as

$$\mathrm{bc}(P) := \min \left\{ \mathrm{rc}(s), \mathrm{rc}(t), \min_{i \in [1, k-1]} \hat{\tau}(r_i, b_{i+1}) \right\}.$$

We *update* $\hat{\tau}$ along P by increasing (resp. decreasing) the mass transported along each forward (resp. backward) edge of P by the bottleneck capacity $\mathrm{bc}(P)$.

When P is a path from a surplus point $b \in B$ to a deficit point $r_\varphi \in A_\delta$, we refer to P as an *augmenting path* and to updating the transport plan τ along P as the *augment* process. Note

that augmenting a transport plan τ along an augmenting path P either removes a backward edge from the residual graph or makes one of the endpoints of P balanced.

Consolidating paths. Any alternating path $P = \langle r_0, b_1, r_1, \ldots, b_k, r_k \rangle$ in the residual graph is called a *consolidating path* if r_0 is a point with $\bar{y}(r_0) = 0$ and r_k is a deficit point with $\bar{y}(r_k) > 0$, i.e., P is a path from a point outside of all 2δ-expanded cells to a violating deficit point. The consolidating path P is admissible if all triples $(b_i, r_i, b_{i+1}) \in P$ are admissible. Note that updating $\hat{\tau}$ along P increases the total mass transported to the violating deficit point r_k by $\mathrm{bc}(P)$ (potentially making it balanced) and decreases the amount of mass transported to the zero-weight point r_0 by $\mathrm{bc}(P)$. Furthermore, updating a transport plan τ along a consolidating path P either removes a backward edge from the residual graph or makes the deficit endpoints of P balanced.

5.3 Our Scaling Algorithm

For parameters $\lambda, \varepsilon > 0$, in this section, we present an algorithm that computes an ε-close λ-robust transport plan between a continuous distribution μ and a discrete distribution ν in $\tilde{O}(n^4 \log \frac{C_{\max}}{\varepsilon})$ time in 2 dimensions and in $n^{O(d)} \log \frac{C_{\max}}{\varepsilon}$ time in d dimensions, where $C_{\max}$ is the diameter of $A \cup B$.

Our algorithm works in $O(\log(\frac{C_{\max}}{\varepsilon}))$ scales, where $C_{\max}$ is the diameter of $A \cup B$. Each scale is associated with a parameter $\delta > 0$ and computes a δ-restricted-feasible λ-robust transport plan. Our algorithm maintains a set of weights $y(\cdot)$ for B. At the beginning of the first scale, set $\delta = C_{\max}$ and $y(b) = 0$ for all $b \in B$. Execute the following steps while $\delta > \varepsilon/2$.

1. *Removing violating deficit points:* For any point $b \in B$, set $\tau_\delta(a, b) := \tau_{2\delta}(a, b)$ for each point $a \in \mathrm{ResVor}_y^{2\delta}(b)$ and $\tau_\delta(a, b) = 0$ for any point $a \notin \mathrm{ResVor}_y^{2\delta}(b)$. Compute $\mathcal{G}_\delta$

and $\hat{\tau}_\delta$ with respect to $\tau_\delta, y(\cdot)$. While there exists a violating deficit point $r \in A_\delta$:

(i) Execute the SEARCHANDCONSOLIDATE procedure, which computes a set of admissible augmenting and consolidating paths in $\mathcal{G}_\delta$ and updates $\hat{\tau}_\delta$ along these paths. At the end of this step, there are no admissible augmenting and consolidating paths to the violating deficit points in the residual graph.

(ii) Execute the REDUCEWEIGHTS procedure, which reduces by δ the weights of all points of B that have admissible paths to the violating deficit points in A_δ and recomputes $\mathcal{G}_\delta$ and $\hat{\tau}_\delta$.

(iii) Execute the ACYCLIFY procedure, which updates $\hat{\tau}_\delta$ and $\mathcal{G}_\delta$ so that the transport plan $\hat{\tau}_\delta$ is a forest and the residual graph does not have any admissible cycles.

2. *Raising surplus weights to λ:* Set all points $b \in B$ with $y(b) < \lambda$ as active. While there are active surplus points in B:

(i) Execute the SEARCHANDAUGMENT procedure, which computes a set of admissible augmenting paths and admissible alternating paths from surplus active points to inactive points in $\mathcal{G}_\delta$ and updates $\hat{\tau}_\delta$ along these paths. At the end of this step, there are no admissible augmenting paths in the residual graph.

(ii) Execute the INCREASEWEIGHTS procedure, which increases the weights of all active points of B which have admissible paths from the active surplus points by δ, marks any point $b \in B$ with $y(b) = \lambda$ as inactive, and recomputes $\mathcal{G}_\delta$ and $\hat{\tau}_\delta$.

(iii) Execute the ACYCLIFY procedure, which updates $\hat{\tau}_\delta$ and $\mathcal{G}_\delta$ 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 $\delta \leftarrow \delta/2$.

Next, we describe the details of the procedures.

5.3.1 SearchAndConsolidate Procedure

The SearchAndConsolidate procedure executes a DFS-style search to find admissible augmenting and consolidating paths in the residual graph and updates the transport plan along those paths. Define $\mathcal{V}$ as the set of all violating deficit points of A_δ. Mark all points of B and all backward edges as unvisited. Let $U := B$ denote the set of unvisited points, and at any point during the execution of the algorithm, let Q denote the search path. The SearchAndConsolidate procedure initiates a partial DFS from each point in $\mathcal{V}$ in the graph $\overleftarrow{\mathcal{G}_\delta}$ to find admissible consolidating and augmenting paths. For any point $r \in \mathcal{V}$, set $Q = \langle r = r_1 \rangle$ and execute the following steps until Q becomes empty.

1. If $Q = \langle r = r_1, b_1, \ldots, r_i, b_i \rangle$,

   (a) If b_i is a surplus point, then the reverse path $P = \langle b_i, r_i, b_{i-1}, \ldots, r_1 \rangle$ is an admissible augmenting path in $\mathcal{G}_\delta$. Augment $\hat{\tau}_\delta$ along P. If r remains a (violating) deficit point, then set $Q = \langle r = r_1 \rangle$ and continue. Otherwise, stop the search from r.

   (b) If b_i is not a surplus point, for any unvisited backward edge (b_i, r') in $\overleftarrow{\mathcal{G}_\delta}$, add r' as r_{i+1} to Q. If there are no unvisited backward edges from b_i in $\overleftarrow{\mathcal{G}_\delta}$, then mark b_i as visited and remove b_i from U and Q. If Q becomes empty, stop the search from b.

2. If $Q = \langle r = r_1, b_1, \ldots, b_i, r_{i+1} \rangle$, 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}.

   (a) If $\bar{y}(r_{i+1}) = 0$, then the reverse path $P = \langle r_{i+1}, b_i, r_i, \ldots, r_1 \rangle$ is an admissible consolidating path in $\mathcal{G}_\delta$. Update $\hat{\tau}_\delta$ along P. If r remains a (violating) deficit point, set $Q = \langle r = r_1 \rangle$ and continue. Otherwise, stop the search from b.

(b) Otherwise, if (b^*, r_{i+1}, b_i) is admissible, i.e., $\mathrm{c}_y(r_{i+1}, b^*) < \mathrm{c}_y(r_{i+1}, b_i)$, then add b^* as b_{i+1} to Q.

(c) Otherwise, remove r_{i+1} from Q and mark the backward edge (b_i, r_{i+1}) as visited.

The procedure terminates when all partial search procedures from all points in $\mathcal{V}$ are terminated.

Lemma 5.5. *Given a δ-restricted-feasible transport plan $\tau_\delta, y(\cdot)$, after the execution of the* SEARCHANDCONSOLIDATE *procedure,*

(SC1) the transport plan remains δ-restricted-feasible,

(SC2) each balanced point $b \in B$ remains balanced,

(SC3) there are no admissible consolidating paths and no admissible augmenting paths to violating deficit regions in the residual graph, and

(SC4) there are no admissible cycles in the residual graph.

Proof. Suppose τ_δ (resp. τ'_δ) denotes the transport plan maintained by the algorithm after (resp. before) the execution of the SEARCHANDCONSOLIDATE procedure.

For any edge $(r, b) \in A_\delta \times B$ with $\hat{\tau}_\delta(r, b) > 0$, if $\hat{\tau}'_\delta(r, b) > 0$, then by the δ-restricted-feasibility of $\hat{\tau}'_\delta, y(\cdot)$, the point r lies inside $\mathrm{ResVor}^{2\delta}_y(b)$. Otherwise, the edge (r, b) was a forward edge on an augmenting or consolidating path P so that after updating the transport plan along P, it has become a backward edge. In this case, there is a forward edge from b to r in the residual graph, which means that $r \in \mathrm{ResVor}^{2\delta}_y(b)$ and (SC1) holds.

Next, note that any augmenting path P is from a surplus point $b \in B$ to a point $r_\varphi \in A_\delta$. Furthermore, any consolidating path P is from a zero-weight point $r' \in A_\delta$ to a violating

deficit point $r_\varphi \in A_\delta$. For any point $b \in B$ other than the two endpoints of a path P, the amount of mass transported from b remains unchanged after updating the transport plan along P. Therefore, updating a transport plan along any augmenting or consolidating path does not change the amount of mass transported from any balanced point, leading to (SC2).

Next, we prove (SC4) and then, use property (SC4) to prove (SC3). Suppose $P^1, \ldots, P^k$ denotes the sequence of augmenting and consolidating paths computed by the SearchAndConsolidate procedure, and for any $i \in [0, k]$, let $\hat{\tau}_\delta^i$ and $\mathcal{G}_\delta^i$ denote the transport plan and residual graph maintained after updating the transport plan along P^i, respectively. Define V^i (resp. U^i) to be the set of visited (resp. unvisited) points of B with respect to $\hat{\tau}_\delta^i$.

We prove (SC4) using an induction on i. Since $\hat{\tau}_\delta^0$ is obtained from the transport plan computed by the Acyclify procedure, there are no cycles of admissible triples in the residual graph, and (SC4) holds for $\mathcal{G}_\delta^0$. For any $i \in [1, k]$, assuming (SC4) on $\mathcal{G}_\delta^0, \ldots, \mathcal{G}_\delta^{i-1}$, we show that (SC4) also holds for $\mathcal{G}^i$. For any admissible triple (b, r, b') in $\mathcal{G}^i$ formed after updating $\hat{\tau}_\delta^{i-1}$ along P^i, we show below that the triple (b, r, b') does not form admissible cycles and conclude (SC4) on $\mathcal{G}_\delta^i$.

For any visited point $b \in V^{i-1} \setminus V^{i-2}$, the search from b did not lead to the computation of an augmenting or consolidating path; therefore, any point b' that is reachable from b by an admissible path in the reverse of residual graph $\overleftarrow{\mathcal{G}^{i-1}}$ (and therefore, is reachable by our partial DFS procedure from b) would have also been marked as visited. In other words, 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 U^{i-1} to any visited point in V^{i-1}.

Below, we show that for any newly formed admissible triple (b, r, b'), $b \in V^{i-1}$ and $b' \in U^{i-1}$. Assuming this, there are no admissible triples from any visited point to any unvisited point

(in the reversed residual graph), and therefore, the newly formed admissible triples (which are from an unvisited point to a visited point in the reversed residual graph) do not form any admissible cycles and (SC4) holds for $\mathcal{G}_\delta^i$ as well. Consider any triple (b, r, b') that is admissible in $\mathcal{G}_\delta^i$ but not in $\mathcal{G}_\delta^{i-1}$. Recall that the SEARCHANDCONSOLIDATE procedure does not change the weights of points, and therefore, since (b, r, b') is not admissible in $\mathcal{G}_\delta^{i-1}$, (r, b') is not a backward edge in $\mathcal{G}_\delta^{i-1}$, i.e., $(r, b') \in P^i$ as a forward edge. From step 2(b) of the SEARCHANDCONSOLIDATE procedure, b' is the weighted nearest unvisited neighbor of r; therefore, since $\mathrm{c}_y(r, b) > \mathrm{c}_y(r, b')$ (from the admissibility of (b, 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'), $b \in V^{i-1}$ and $b' \in U^{i-1}$.

Finally, to prove property (SC3), we begin by showing the following property:

(SC5) any point $b \in B$ and any backward edge that are marked as visited does not form any admissible augmenting paths to violating deficit points or consolidating paths in the same execution of the SEARCHANDCONSOLIDATE procedure.

For any visited point $b \in V^i$, as discussed above, all vertices that are reachable from b by an admissible path in the reversed residual graph are also visited. Since all surplus points $b_s \in B$ (resp. points $b_z \in B$ transporting mass to a zero-weight point $r \in A_\delta$) are unvisited, b_s (resp. b_z) is not reachable from any visited point $b \in B$ by our search algorithm and therefore, the visited points do not participate in an admissible augmenting or consolidating 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 to a violating deficit point or a consolidating path, the visited backward edge (r, b) also does not form an admissible augmenting path to a violating deficit point or a consolidating path.

For any violating deficit point $r_\varphi \in A_\delta$ with respect to $\hat{\tau}_\delta, y(\cdot)$, the search from r_φ has been terminated since Q was empty; hence, all forward edges incident on r_φ are to visited points of B, which from (SC5) do not form admissible augmenting/consolidating paths; therefore, r_φ will not form any admissible augmenting paths to violating deficit points or consolidating paths during the same execution of the SearchAndConsolidate procedure, leading to (SC3). □

5.3.2 ReduceWeights Procedure

The ReduceWeights procedure reduces the weights of all points in B that have admissible alternating paths to some deficit points $r \in A$ with $\bar{y}(r) > 0$, leading to the formation of new admissible augmenting and consolidating paths to the violating free points in the residual graph. The procedure performs partial DFS from all violating deficit points in the reverse of the residual graph to compute a set $\mathcal{K}$ of all points of B that have admissible alternating paths to the violating deficit points. The procedure then decreases the weights of all points in $\mathcal{K}$ by δ and shrinks their restricted Voronoi cells. It finally recomputes A_δ, $\mathcal{G}_\delta$, and $\hat{\tau}_\delta$. We describe the steps of the DFS below.

Define $\mathcal{V} \subset A_\delta$ as the set of violating deficit points. Given the reversed residual graph $\overleftarrow{\mathcal{G}_\delta}$, mark all points $b \in B$ and all backward edges (b, r) as unvisited and set $\mathcal{K} = \emptyset$ and let $U = B$ denote the set of unvisited points of B. For any point $r \in \mathcal{V}$, start a partial DFS in the reverse residual graph $\overleftarrow{\mathcal{G}_\delta}$ by setting $Q := \langle r = r_1 \rangle$ as the search path that the algorithm grows. Execute the following steps.

1. If $Q = \langle r = r_1, \ldots, b_i \rangle$,

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

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

2. If $Q = \langle r = r_1, \ldots, b_i, r_i \rangle$, let $b^* := \arg\min_{b' \in U} \mathrm{c}_y(r_i, b')$.

   (a) If $\mathrm{c}_y(r_i, b^*) < \mathrm{c}_y(r_i, b_i)$, i.e., (b^*, r_i, b_i) is admissible, add b^* as b_{i+1} to Q.

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

After all DFS procedures from all points in $\mathcal{V}$ terminate, for each point $b \in \mathcal{K}$, set $y(b) \leftarrow y(b) - \delta$.

Lemma 5.6. *Given a δ-restricted-feasible transport plan $\tau_\delta, y(\cdot)$ such that there are no admissible consolidating paths and no admissible augmenting path to a violating deficit region, after the execution of the* ReduceWeights *procedure,*

(RW1) the transport plan remains δ-restricted-feasible,

(RW2) the weight of each point $b \in B$ containing violating deficit regions inside $\mathrm{ResVor}_y(b)$ *decreases by δ, and*

(RW3) the weight of each surplus point $b \in B$ remains unchanged.

Proof. Let $y(\cdot), \mathcal{X}_\delta, A_\delta$ (resp. $y'(\cdot), \mathcal{X}'_\delta, A'_\delta$) denote the set of weights of B, decomposition of A, and the set of representative points before (resp. after) the execution of the ReduceWeights procedure.

To prove property (RW1), first note that for each point $b \in \mathcal{K}$, there are no zero-weight points $r \in \mathrm{ResVor}_y^{2\delta}$ such that $\tau_\delta(r, b) > 0$, since otherwise, there would have been an admissible consolidating path in the residual graph, which is in contrast with Lemma 5.5 (SC3). Below, we show that for any pair of points $(a, b) \in A \times B$ such that $\tau_\delta(a, b) > 0$, we have $a \in \mathrm{ResVor}_{y'}^{2\delta}(b)$, and hence (RW1) holds. Let b_a (resp. b'_a) denote the weighted nearest neighbor of a with respect to weights $y(\cdot)$ (resp. $y'(\cdot)$). From the δ-restricted-feasibility of

$\tau_\delta, y(\cdot)$, $\mathrm{c}_y(a,b) \leq \mathrm{c}_y(a,b_a) + 2\delta$. Let r_a (resp. r'_a) denote the representative point of the region containing a. As discussed above, $\bar{y}(r_a) > 0$ and r_a lies inside the restricted Voronoi cell of b_a.

- If $b \in \mathcal{K}$,
  - if $b'_a \in \mathcal{K}$, then

$$\mathrm{c}_{y'}(a,b) = \mathrm{c}_y(a,b) + \delta \leq \mathrm{c}_y(a,b'_a) + 3\delta = \mathrm{c}_{y'}(a,b'_a) + 2\delta.$$

  - Otherwise, the triple (b'_a, r_a, b) would not be admissible (since otherwise $b'_a \in \mathcal{K}$), and $\mathrm{c}_y(r_a,b) \leq \mathrm{c}_y(r_a,b'_a)$. Hence, $\mathrm{c}_y(a,b) \leq \mathrm{c}_y(a,b'_a) + \delta$ and

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

- Otherwise, $b \notin \mathcal{K}$. In this case,

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

Therefore, for any pair (a,b) with $\tau_\delta(a,b) > 0$, we have $a \in \mathrm{ResVor}_y^{2\delta}(b)$, and $\tau_\delta, y'(\cdot)$ would be δ-restricted-feasible, i.e., (RW1) holds.

Since the ReduceWeights procedure initiates a partial DFS from each violating deficit point $r \in \mathcal{V}$, all points of B with violating deficit regions in their restricted Voronoi cell will be added to the search path and to the set $\mathcal{K}$ in step 1(b) of the procedure; hence, their weights will be reduced by δ (leading to (RW2)). Furthermore, for any surplus point $b \in B$, since there are no admissible augmenting paths from the violating deficit regions (from Lemma 5.5), the surplus point b will not be added to the search path and hence, cannot be

added to the set $\mathcal{K}$, leading to (RW3). □

5.3.3 ACYCLIFY Procedure

The changes in the weights of B in the REDUCEWEIGHTS and INCREASEWEIGHTS procedures require 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, ensuring the correctness and efficiency of our algorithm. Converting the transport plan into a forest is critical for the efficiency of the SEARCHANDCONSOLIDATE and SEARCHANDAUGMENT procedures while eliminating admissible cycles is essential for the correctness of those procedures. The procedure executes the following steps. First, use the dynamic tree structure by Sleator and Tarjan [149] to make the transport plan $\hat{\tau}_\delta$ a forest. Then, execute a partial DFS search from each unvisited point $b \in B$ 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 A.3.2 for details. The following lemma shows the properties of the ACYCLIFY procedure.

Lemma 5.7. *Given a δ-restricted-feasible transport plan $\tau_\delta, y(\cdot)$, after the execution of the* ACYCLIFY *procedure,*

(AC1) the transport plan $\hat{\tau}_\delta, y(\cdot)$ remains δ-restricted-feasible,

(AC2) the transport plan $\hat{\tau}_\delta$ is a forest and there are no admissible cycles in the residual graph, and

(AC3) if $\tau_\delta, y(\cdot)$ satisfy condition (F3) in Lemma 5.2, then condition (F3) remains satisfies.

5.3.4 SearchAndAugment Procedure

In the second step of our algorithm, the transport plan $\tau_\delta, y(\cdot)$ maintained by our algorithm is δ-restricted-feasible and satisfies (F1), i.e., all deficit points of A_δ have a zero derived weight. To retrieve property (F2) (that is all surplus points in B have a weight of λ), the SearchAndAugment procedure uses a DFS-style search procedure to identify admissible augmenting paths and admissible alternating paths from surplus active points to inactive points of B in the residual graph and updates the transport plan along those paths. More precisely, the SearchAndAugment starts a partial DFS procedure from each deficit points of A_δ in the reverse of the residual graph, similar to the one described for the SearchAndConsolidate procedure. As soon as the search path reaches a surplus or an inactive point $b \in B$ (i.e., the procedure have found an admissible augmenting path or an alternating path from an inactive point to a deficit point), the procedure updates the transport plan along the search path and continues the search. The details of the procedure are described in details in Appendix A.3.3. The following lemma states the useful properties of the SearchAndAugment procedure.

Lemma 5.8. *Given a δ-restricted-feasible transport plan $\tau_\delta, y(\cdot)$ satisfying condition (F3) in Lemma 5.2, after the execution of the* SearchAndAugment *procedure,*

(SA1) the transport plan $\hat{\tau}_\delta, y(\cdot)$ remains δ-restricted-feasible and (F3) remains satisfies, and

(SA2) there are no admissible cycles, admissible augmenting paths, and admissible alternating paths from inactive points to deficit regions in the residual graph.

5.3.5 IncreaseWeights Procedure

The IncreaseWeights procedure adjusts the weights of B leading to the formation of admissible augmenting paths and also deactivate the points of B whose dual weights have reached λ. The IncreaseWeights procedure performs a DFS on the residual graph $\mathcal{G}_\delta$ to compute a set $\mathcal{K}$ of all active 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 restricted Voronoi cells) and for the points $b \in B$ whose dual weight have reached λ, the IncreaseWeights procedure marks b as inactive. 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 A.3.4 for complete details of the IncreaseWeights procedure. The following lemma states the properties of the IncreaseWeights procedure.

Lemma 5.9. *Given a δ-restricted-feasible transport plan $\tau_\delta, y(\cdot)$ satisfying condition (F3) in Lemma 5.2, after the execution of the* IncreaseWeights *procedure,*

(IW1) the transport plan $\hat{\tau}_\delta, y(\cdot)$ remain δ-restricted-feasible and condition (F3) remains satisfies,

(IW2) the weight of each active surplus point $b \in B$ increases by δ, and

(IW3) the weights of all points $b \in B$ with deficit regions inside $\mathrm{ResVor}_y^{2\delta}(b)$ and all inactive points remain unchanged.

5.4 Analysis

5.4.1 Correctness Analysis

In each scale δ, our algorithm begins with a δ-restricted-feasible transport plan that might not satisfy properties (F3) and (F4) in Lemma 5.2. In the first step, our algorithm iteratively runs the SearchAndConsolidate, ReduceWeights, and Acyclify procedures until there are no violating points in A_δ remaining. Using Lemma 5.5 (SC1), Lemma 5.6 (RW1), and Lemma 5.7 (AC1), the transport plan $\tau_\delta, y(\cdot)$ remains δ-restricted-feasible. The first step of our algorithm terminates since, from Lemma 5.6 (RW2), the weight of each point $b \in B$ containing violating deficit regions inside their restricted Voronoi cells will be reduced in each iteration by δ and the restricted Voronoi cell of any point with a zero weight is empty. Therefore, the first step terminates with no violating points remaining in A_δ, i.e., the first step of our algorithm restores the condition (F3) in Lemma 5.2 while maintaining the δ-restricted-feasibility conditions.

In step 2, our algorithm iteratively executes the SearchAndAugment, IncreaseWeights, and Acyclify procedures. Using Lemma 5.8 (SA1), Lemma 5.9 (IW1), and Lemma 5.7 (AC1) and (AC3), the transport plan $\tau_\delta, y(\cdot)$ maintained by our algorithm during the second step remains δ-restricted-feasible satisfying condition (F3). Furthermore, from Lemma 5.9 (IW2), the weight of each active surplus point increases by δ in each iteration of the second step, and each point will be marked as inactive when its dual weight reaches λ; hence, the second step of our algorithm terminates with a transport plan $\tau_\delta, y(\cdot)$ that does not have any active surplus points. Therefore, after the second step terminates, $\tau_\delta, y(\cdot)$ is a δ-restricted-feasible satisfying conditions (F3) and (F4) and from Lemma 5.2, the computed transport plan at the end of each scale δ is a 2δ-close λ-ROT plan.

5.4.2 Efficiency Analysis

From Lemma 5.4, in 2 dimensions, the residual graph contains $O(n^2)$ nodes and $O(n^3)$ edges. Using this, we show in Appendix A.3 that each execution of the SEARCHANDCONSOLIDATE, ACYCLIFY, and SEARCHANDAUGMENT , procedures takes $O(n^3 \log n)$ time and each execution of the REDUCEWEIGHTS and INCREASEWEIGHTS procedures takes $O(n^2(\mathrm{Q} + n \log n))$ time, where Q denotes the query time of an oracle that computes the continuous mass of μ inside a query triangle. Consequently, each iteration of steps 1 and 2 of our algorithm takes $O(n^2(\mathrm{Q} + n \log n))$ time. In the following, we show that both steps of our algorithm run $O(n)$ iterations and therefore, have a total execution time of $O(n^3(\mathrm{Q} + n \log n))$. Summing over all $O(\log \frac{C_{\max}}{\varepsilon})$ scales, we get a total running time of $O(n^3(\mathrm{Q} + n \log n) \log \frac{C_{\max}}{\varepsilon})$ for our algorithm, as claimed.

Number of iterations of step 1. We begin by showing that the number of iterations of step 1 of our algorithm is $O(n)$. Suppose $y_{2\delta}(\cdot)$ denotes the set of weights of B maintained by our algorithm at the end of scale 2δ. For any iteration i of step 1, let $\tau^i, y^i_{\text{step1}}(\cdot)$ denote the δ-feasible transport plan maintained by our algorithm after the ith iteration and define $\gamma^i_{\text{step1}}(b) := y_{2\delta}(b) - y^i_{\text{step1}}(b)$.

Lemma 5.10. *For any subset $S \subset B$, suppose in an iteration i of step 1 of our algorithm, the reduction in the weight of any point in S is more than 6δ greater than the reduction in the weight of any point in $B \setminus S$, i.e., $\min_{b \in S} \gamma^i_{step1}(b) > \max_{b' \in B \setminus S} \gamma^i_{step1}(b') + 6\delta$. Then, there are no deficit regions inside the restricted Voronoi cells of the points in S.*

Proof. Let $\tau_{2\delta}, y_{2\delta}(\cdot)$ denote the 2δ-restricted-feasible transport plan computed by our algorithm at the end of scale 2δ. Recall that $\tau^i, y^i_{\text{step1}}(\cdot)$ denotes the δ-restricted-feasible transport plan maintained after iteration i of the initialization step and $\gamma^i_{\text{step1}}(b) = y_{2\delta}(b) - y^i_{\text{step1}}(b)$

for each point $b \in B$. Define $\mathcal{R}_S^\delta := \bigcup_{b \in S} \mathrm{ResVor}^{2\delta}_{y^i_{\mathrm{step1}}}(b)$ as the union of all 2δ-expanded restricted Voronoi cells of the points in S.

To prove this lemma, we use the gap of 6δ in the reduction in the weights of the points in S and the points in $B \setminus S$ to show that all the continuous mass inside $\mathcal{R}_S^\delta$ is transported to the points in S in $\tau_{2\delta}$. Assuming this, the total continuous mass inside $\mathcal{R}_S^\delta$ would be no more than the discrete mass at the points in S. Furthermore, by the δ-restricted-feasibility conditions, τ^i transports the mass of the points in S only to the regions in $\mathcal{R}_S^\delta$. Since the ReduceWeights procedure does not reduce the weight of any surplus points, no points in S can be surplus; therefore, since no points in S can be deficit, all points in S are balanced and their mass is fully transported to the regions inside $\mathcal{R}_S^\delta$, i.e., there are no violating free regions inside the 2δ-expanded Voronoi cells of S and the procedure terminates. Consequently, to complete the proof, we show in the following that all the continuous mass inside $\mathcal{R}_S^\delta$ is transported to the points in S in $\tau_{2\delta}$.

Consider any point $a \in \mathcal{R}_S^\delta$, and let $b_a \in B$ be any 4δ-weighted nearest neighbor of a with respect to weights $y_{2\delta}(\cdot)$. Below, we show that $b_a \in S$. For the sake of contradiction, suppose $b_a \notin S$, and let $b' \in S$ denote any point in the set S. Since b_a is a 4δ-WNN of a,

$$\mathrm{c}_{y_{2\delta}}(a, b_a) \leq \mathrm{c}_{y_{2\delta}}(a, b') + 4\delta. \tag{5.19}$$

Furthermore, by the assumption of the lemma, since $b' \in S$ and $b_a \notin S$

$$\gamma^i_{\mathrm{step1}}(b') < \gamma^i_{\mathrm{step1}}(b_a) + 6\delta. \tag{5.20}$$

Therefore,

$$\begin{aligned}
\mathrm{c}_{y^i_{\text{step1}}}(a, b') &= \mathrm{c}_{y_{2\delta}}(a, b') + y_{2\delta}(b') - y^i_{\text{step1}}(b') \\
&\geq \Big(\mathrm{c}_{y_{2\delta}}(a, b_a) - 4\delta\Big) + \gamma^i_{\text{step1}}(b') && \text{(from Equation (5.19))} \\
&= \Big(\mathrm{c}_{y^i_{\text{step1}}}(a, b_a) + y^i_{\text{step1}}(b_a) - y_{2\delta}(b_a) - 4\delta\Big) + \gamma^i_{\text{step1}}(b') \\
&= \mathrm{c}_{y^i_{\text{step1}}}(a, b_a) - \gamma^i_{\text{step1}}(b_a) + \gamma^i_{\text{step1}}(b') - 4\delta \\
&> \mathrm{c}_{y^i_{\text{step1}}}(a, b_a) + 2\delta. && \text{(from Equation (5.20))}
\end{aligned}$$

In words, since the decrease in the weights of the points in S is more than 6δ greater than the decrease in the weights of the points in $B \setminus S$, the increase in the weighted distance of any point $b' \in S$ to a would be more than 6δ greater than the increase in the weighted distance of b_a to a. Hence, the weighted distance of any point $b' \in B \setminus S$ to a would be more than 2δ greater than the weighted distance of a to b_a; consequently, the point a cannot lie inside the union of the 2δ expanded restricted Voronoi cells of S (i.e., $a \notin \mathcal{R}^\delta_S$), which is a contradiction. Hence, any point $b_a \in B$ that transports mass to a in $\tau_{2\delta}$ (i.e., any 4δ-WNN of a with respect to weights $y_{2\delta}(\cdot)$) has to be in the set S.

Next, note that any point $a \in \mathcal{R}^\delta_S$ lies inside the 2δ-expanded Voronoi cell of at least one point in S with respect to weights $y^i_{\text{step1}}(\cdot)$ and therefore has a positive derived weight, i.e., $y^{i^-}_{\text{step1}}(a) > 0$. Since no points in B have their weights increased, $\bar{y}_{2\delta}(a) \geq y^{i^-}_{\text{step1}}(a)$ and the derived weight $\bar{y}_{2\delta}(a)$ with respect to weights $y_{2\delta}(\cdot)$ would also be positive. Consequently, from the 2δ-restricted-feasibility of $\tau_{2\delta}, y_{2\delta}(\cdot)$, the point a is not a deficit point in $\tau_{2\delta}$, and all continuous mass inside $\mathcal{R}^\delta_S$ is transported to the points in S in $\tau_{2\delta}$, as claimed. □

For any iteration i of step 1, let $A^i_F \subset A_\delta$ (resp. $B^i_S \subset B$) denote the subset of violating deficit points (resp. surplus points) with respect to τ^i. Note that step 1 terminates as soon

as A_F^i becomes empty. Since the total amount of continuous mass transported by τ^i is no more than the total discrete mass at the points in B, if $A_F^i \neq \emptyset$, then there exists at least one surplus point $b_s \in B_S^i$. Since step 1 does not create new surplus points (from Lemma 5.5 (SC2), balanced points remain balanced), the point b_s was surplus in all previous iterations as well, and $\gamma^i_{\text{step1}}(b_s) = 0$. Furthermore, from Lemma 5.6 (RW2), the weight of the point $b_r \in B$ containing any deficit point $r \in A_F^i$ reduces by δ in each iteration of step 1, and therefore, $\gamma^i_{\text{step1}}(b_r) = \delta i$. Consequently, if $i > 6n$, then $\gamma^i_{\text{step1}}(b_r) - \gamma^i_{\text{step1}}(b_s) > 6n\delta$, and there will be a subset $S \subset B$ such that $b_r \in S$ for all points $b_r \in B$ containing a violating deficit point $r \in A_F^i$, $b_s \notin S$ for all surplus points $b_s \in B_S^i$, and that S satisfies the conditions of Lemma 5.10. In this case, from Lemma 5.10, there are no deficit regions inside the restricted Voronoi cells of the points in S, which is a contradiction. Hence, there will be no violating deficit points in A_δ after $6n$ iterations of step 1, i.e., step 1 will terminate after at most $6n$ iterations.

Number of iterations of step 2. Next, using a similar approach as our analysis for the number of iterations of step 1, we show that, in each scale δ, the second step of our algorithm executes $O(n)$ iterations. For any iteration i, let $\tau^i_{\text{step2}}, y^i_{\text{step2}}(\cdot)$ denote the δ-restricted-feasible transport plan maintained after iteration i of step 2. Define $\gamma^i_{\text{step2}}(b) := y^i_{\text{step2}}(b) - y_{2\delta}(b)$.

Lemma 5.11. *For any subset $S \subset B$, suppose in an iteration $i > 12n$ of step 2, the increase in the weight of any point $b \in S$ is more than 6δ greater than the increase in the weight of any point $b' \in B \setminus S$, i.e., $\min_{b\in S} \gamma^i_{step2}(b) > \max_{b'\in B\setminus S} \gamma^i_{step2}(b') + 6\delta$. Then, all points in S are balanced in τ^i_{step2}.*

Proof. First, recall that step 2 terminates when there are no active surplus points. Hence, we assume that there exists an active surplus point $b_s \in B$ in iteration i. Since in each iteration, the weight of b_s increases by δ, $\gamma^i_{\text{step2}}(b_s) = i\delta$. Also, b_s has the highest increase

in weight and therefore, $b_s \in S$. Note that any point $b \in B$ that is surplus with respect to the transport plan τ^i_{step2} was also a surplus point in all previous iterations of steps 1 and 2. Hence, the weight of b has not been reduced in step 1. Recall that if b was a surplus point in $\tau_{2\delta}$, then b has been deactivated by our algorithm, which means $y_{2\delta}(b) = \lambda$. However, step 1 has not reduced the weight of b, and therefore, the point b would have been deactivated in step 2. As a result, since b_s is an active surplus point in iteration $i > 12n$, the surplus point b_s cannot be a surplus point in $\tau_{2\delta}$ and therefore, is a balanced point in $\tau_{2\delta}$.

Second, note that any surplus point $b \in B$ with respect to $\tau_{2\delta}, y_{2\delta}(\cdot)$ has a weight $y_{2\delta}(b) = \lambda$, and since the number of iterations of step 1 is at most $6n$ (in which the weight of b_s might have decreased by at most $6n\delta$ while the weight of b has remained unchanged) and step 2 has executed $i > 12n$ iterations (in which the weight of b_s have increased by $12n\delta$ while the weight of b has remained unchanged), $\gamma^i_{\text{step2}}(b_s) \geq \gamma^i_{\text{step2}}(b) + 6n\delta$, and b cannot be in S. Therefore, all points $b \in S$ are balanced in $\tau_{2\delta}$.

Define $\text{ResVor}^{-2\delta}_{y^i_{\text{step2}}}(b)$ as the 2δ-shrunken restricted Voronoi cells of any point $b \in B$, i.e., the restricted Voronoi cell of b with respect to a weight function ${y^i_{\text{step2}}}'(\cdot)$ that assigns ${y^i_{\text{step2}}}'(b) = y^i_{\text{step2}}(b) - 2\delta$ and ${y^i_{\text{step2}}}'(b') = y^i_{\text{step2}}(b')$ to any point $b' \neq b$ in B. Define $\mathcal{R}^\delta_S := \bigcup_{b\in S} \text{ResVor}^{-2\delta}_{y^i_{\text{step2}}}(b)$ as the union of all 2δ-shrunken restricted Voronoi cells of the points in S. From the δ-restricted-feasibility of $\tau^i_{\text{step2}}, y^i_{\text{step2}}(\cdot)$, the continuous mass inside $\mathcal{R}^\delta_S$ is transported only to the point in S. To prove this lemma, we show that the amount of continuous mass inside the set $\mathcal{R}^\delta_S$ is no less than the discrete mass at the points in S. Assuming this, no points in S can be surplus, and all points in S are balanced. In the following, we show that the total continuous mass inside $\mathcal{R}^\delta_S$ is equal to $\nu(S)$.

Consider any point $b \in S$ and any point $a \in A$ such that $\tau_{2\delta}(a, b) > 0$. Below, we show that $a \in \mathcal{R}^\delta_S$. Let $b' \notin S$ be any other point. Since $\tau_{2\delta}(a, b) > 0$, then b is a 4δ-weighted nearest

neighbor of a with weights $y_{2\delta}(\cdot)$ and therefore,

$$\mathrm{c}_{y_{2\delta}}(a,b) \leq \mathrm{c}_{y_{2\delta}}(a,b') + 4\delta. \tag{5.21}$$

Furthermore, since $b \in S$ and $b' \notin S$,

$$\gamma_{\text{step2}}(b) > \gamma_{\text{step2}}(b') + 6\delta. \tag{5.22}$$

Therefore,

$$\begin{aligned}
\mathrm{c}_{y^i_{\text{step2}}}(a,b) &= \mathrm{c}_{y_{2\delta}}(a,b) + y_{2\delta}(b) - y^i_{\text{step2}}(b) \\
&\leq \mathrm{c}_{y_{2\delta}}(a,b') + 4\delta - \gamma^i_{\text{step2}}(b) && \text{(from Equation (5.21))} \\
&= \mathrm{c}_{y^i_{\text{step2}}}(a,b') + y^i_{\text{step2}}(b') - y_{2\delta}(b') + 4\delta - \gamma^i_{\text{step2}}(b) \\
&= \mathrm{c}_{y^i_{\text{step2}}}(a,b') + \gamma^i_{\text{step2}}(b') + 4\delta - \gamma^i_{\text{step2}}(b) \\
&< \mathrm{c}_{y^i_{\text{step2}}}(a,b') - 2\delta. && \text{(from Equation (5.22))}
\end{aligned}$$

In words, with respect to weights $y^i_{\text{step2}}(\cdot)$, the weighted distance of b to a would be more than 2δ less than the weighted distance of a to any point $b' \notin S$, i.e., $a \in \mathcal{R}^\delta_S$.

Consequently, all continuous mass inside $\mathcal{R}^\delta_S$ is transported to the points in S in $\tau_{2\delta}$, as claimed. Since all points in S are balanced in $\tau_{2\delta}$, the total discrete mass at the points of S is no more than the continuous mass inside $\mathcal{R}^\delta_S$, and since there are no deficit points in B, there are also no surplus points in S and all points in S are balanced. □

From Lemma 5.9, the weight of each active surplus point $b \in B$ increases by δ in each iteration of step 2. Furthermore, for any point $b \in B$ that had a weight of λ in the previous scale, the weight of b has been reduced in step 1 by at most $6n\delta$, and in step 2, it can increase

by at most $6n\delta$ before being marked as inactive. Assume, for the sake of contradiction, that the number of iterations of step 2 has exceeded $12n$, and consider any iteration $i > 12n$. In this case, for any surplus point $b_s \in B$, since b_s was a surplus point in all previous iterations, including the iterations of step 1, we have

$$\gamma^i_{\text{step2}}(b_s) = y^i_{\text{step2}}(b_s) - y_{2\delta}(b_s) = (y^i_{\text{step2}}(b_s) - y^i_{\text{step1}}(b_s)) - (y_{2\delta}(b_s) - y^i_{\text{step1}}(b_s)) = 12n\delta - 0 = 12n\delta.$$

Furthermore, for any inactive point $b' \in B$ with respect to $\tau_{2\delta}, y_{2\delta}(\cdot)$, after increasing the weight of b' by at most $6n\delta$, it becomes inactive; hence, the difference in the increases in the weights of b_s and b' is more than $6n\delta$, and there exists a subset S of B where all active surplus points are in S, all inactive points are in $B \setminus S$, and that S satisfies the condition of Lemma 5.11. However, from Lemma 5.11, all points in S have to be balanced, which is a contradiction. Therefore, the step 2 of our algorithm terminates in at most $12n$ iterations.

Chapter 6

A Sub-Quadratic Algorithm for Stochastic Euclidean Matching

Given two sets A and B of n points inside the unit square, in this chapter, we present an algorithm for computing a minimum-cost perfect matching between A and B. Our algorithm as well as its analysis extends to any dimension $d \geq 2$; however, to simplify the presentation, we present and analyze our algorithm for two dimensions and provide its analysis for d-dimensional settings in Appendix B.1.

6.1 Geometric Primal-Dual Framework

6.1.1 Hierarchical Partitioning

Using $\lambda := 9n^{-1/4}$, we construct a hierarchical partitioning $\mathcal{H}$ recursively. Each node of $\mathcal{H}$ is an axis-parallel rectangle, referred to as a *cell*. The root node of $\mathcal{H}$, denoted by $\square^*$, is the square $\square^* := [-3, 3]^2$ containing all points in $A \cup B$. For each node $\square$, let $A_\square$ and $B_\square$ be the points of A and B inside $\square$ and define $n_\square := |A_\square \cup B_\square|$ as the number of points of $A \cup B$ that lie inside $\square$. If $n_\square \leq 1$, i.e., $\square$ contains at most 1 point of $A \cup B$, then $\square$ is marked as a leaf node. Otherwise, we partition $\square$ into two smaller rectangles as follows. Let $\ell_\square$ be the larger of the length and width of rectangle $\square$. Without loss of generality, assume that $\ell_\square$

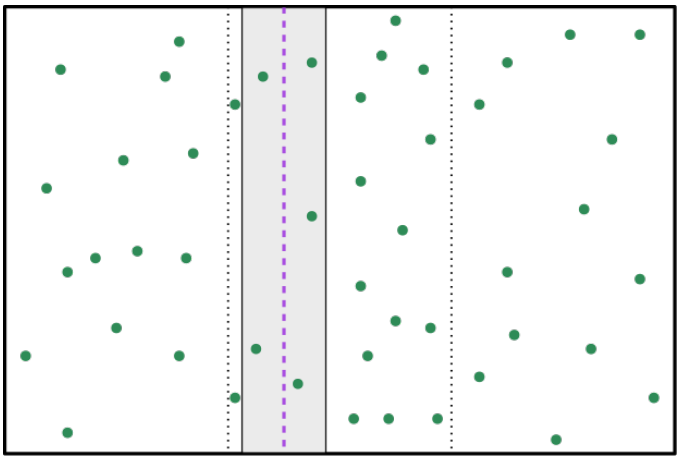


Figure 6.1: Partitioning a rectangle into two children (the purple dashed vertical line is the divider).

is the width of $\square$ and let $x_{\min}$ be the x-coordinate of the bottom-left corner of $\square$. For any value $\hat{x} \in [x_{\min} + \frac{\ell_\square}{3}, x_{\min} + \frac{2\ell_\square}{3}]$, define

$$\Lambda(\hat{x}) := \{u \in A_\square \cup B_\square : |u_x - \hat{x}| \leq \ell_\square \lambda\};$$

here, u_x denotes the x coordinate of the point u. Let

$$x^* := \arg \min_{\hat{x} \in [x_{\min} + \frac{\ell_\square}{3}, x_{\min} + \frac{2\ell_\square}{3}]} |\Lambda(\hat{x})|.$$

We partition $\square$ into two smaller rectangles by using a vertical line defined by $x = x^*$ and add them as the children of $\square$ to $\mathcal{H}$. We refer to the segment partitioning $\square$ into its two children as its *divider* and denote it by $\Gamma_\square$. See Figure 6.1. For any cell $\square$, the four sides of its rectangle are defined by the dividers of its ancestor or the boundaries of the root square. This completes the construction of $\mathcal{H}$. Note that the height of the tree is $O(\log n\Delta)$.

A simple sweep-line algorithm can compute x^* in $O(n_\square \log n_\square)$ time. At a high level, the sweep-line algorithm moves a vertical sweep line from left to right, maintaining the number

of points that are at a distance at most $\ell_\square \lambda$ from the sweep line. Using this procedure, we construct $\mathcal{H}$ in $\tilde{O}(n \log(n\Delta))$ time. We describe the sweep-line algorithm in detail in Appendix B.1.1.

Lemma 6.1. *For each cell $\square$ of $\mathcal{H}$, the ratio of the largest to the smallest side of $\square$ is at most 3. Furthermore, the number of points of $A_\square \cup B_\square$ with a distance smaller than $\ell_\square \lambda$ to the divider $\Gamma_\square$ is $O(n_\square \lambda)$.*

Proof. We use an inductive proof to show that the aspect ratio of all cells of $\mathcal{H}$ is at most 3. Note that the root cell $\square^*$ is square and has a unit aspect ratio. For any non-root cell $\square \in \mathcal{H}$, let $\square'$ denote the parent of $\square$ in $\mathcal{H}$. By the inductive hypothesis, the aspect ratio of $\square'$ is at most 3. Let ℓ_x (resp. ℓ_y) denote the width (resp. length) of $\square$. Similarly, let ℓ'_x (resp. ℓ'_y) denote the width (resp. length) of $\square'$. Without loss of generality, assume $\ell'_x \geq \ell'_y$. In this case, we split $\square'$ into two rectangles using a vertical line in the middle third part of the x side of $\square'$, i.e.,

$$\ell_y = \ell'_y \quad \text{and} \quad \ell_x \in \left[\frac{\ell'_x}{3}, \frac{2\ell'_x}{3}\right].$$

In this case, if $\ell_x \geq \ell_y$, then

$$\frac{\ell_x}{\ell_y} \leq \frac{\ell'_x}{\ell'_y} \leq 3.$$

Otherwise, $\ell_x < \ell_y$ and

$$\frac{\ell_y}{\ell_x} \leq \frac{\ell'_y}{\ell'_x/3} \leq 3.$$

Next, we show that the number of points of $A_\square \cup B_\square$ at a distance at most $\ell_\square \lambda$ from $\Gamma_\square$ is $O(n_\square \lambda)$. Consider the set

$$\zeta = \{x_{\min} + \frac{\ell_\square}{3} + 3i\ell_\square \lambda : 0 \leq i \leq \lfloor \frac{1}{9\lambda} \rfloor\}.$$

Recall that for any value $x' \in [x_{\min} + \frac{\ell_\square}{3}, x_{\min} + \frac{2\ell_\square}{3}]$, $\Lambda(x')$ denotes the set of all points of

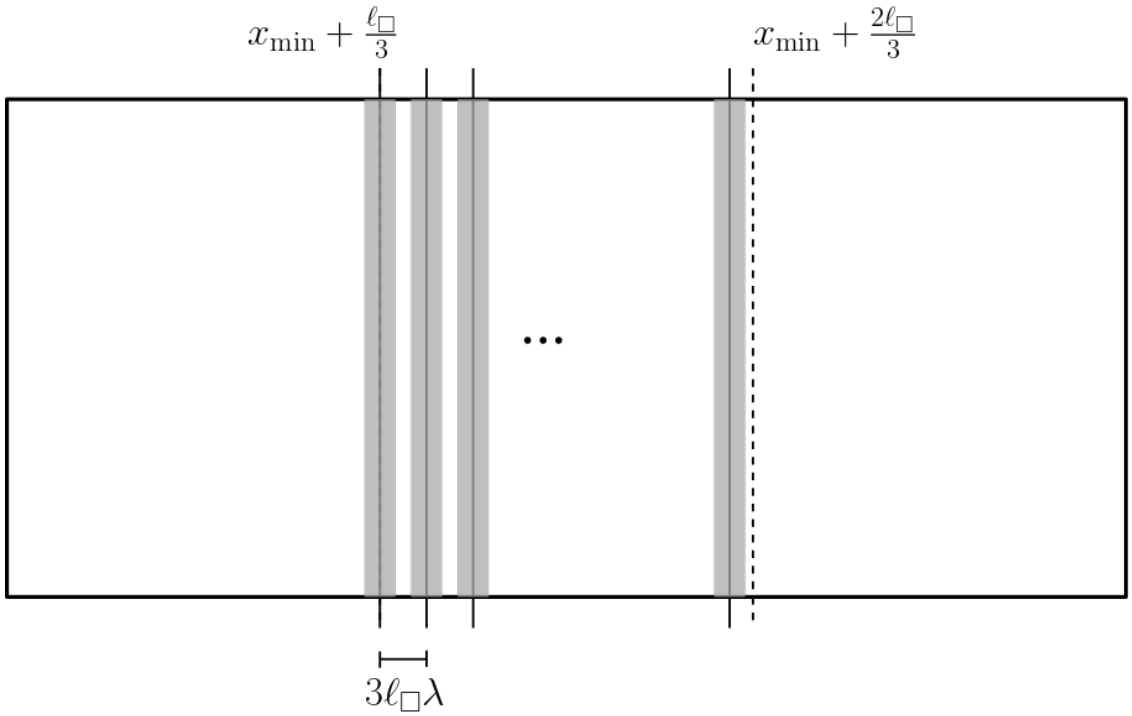


Figure 6.2: The solid vertical line shows the set ζ.

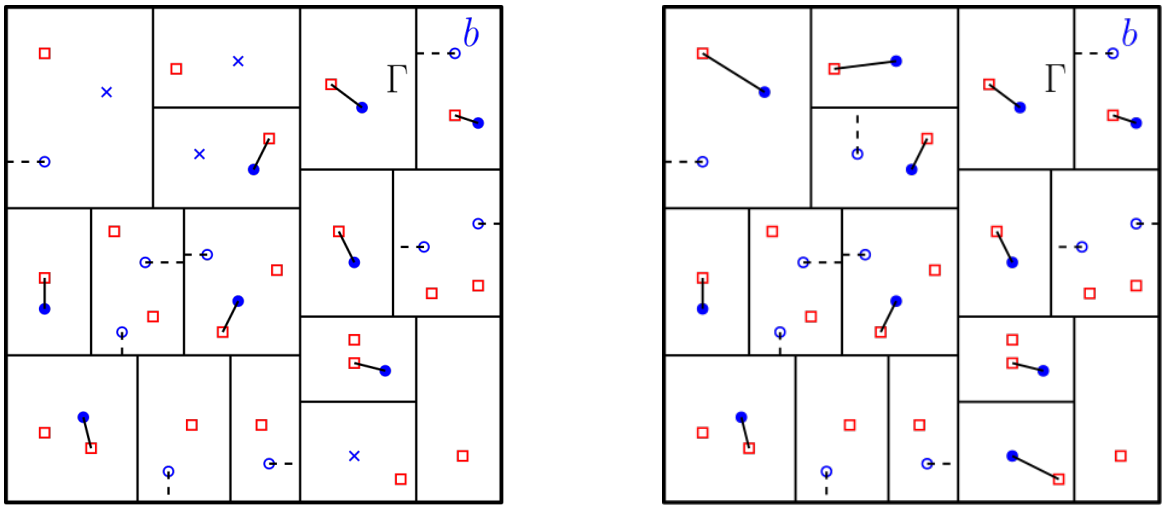
$A_\square \cup B_\square$ that are at a distance at most $\ell_\square\lambda$ from the vertical line $x = x'$. Note that for any pair of distinct values $x_1, x_2 \in \zeta$, $\Lambda(x_1) \cap \Lambda(x_2) = \emptyset$. See Figure 6.2. Furthermore,

$$\sum_{x' \in \zeta} |\Lambda(x')| \leq n_\square.$$

Define $x^* := \arg\min_{x' \in \zeta} |\Lambda(x')|$. In this case, the size of $\Lambda(x^*)$ is no more than the average size of the $\Lambda(x')$ for the values $x' \in \zeta$; more precisely,

$$|\Lambda(x^*)| \leq \frac{\sum_{x' \in \zeta} |\Lambda(x')|}{|\zeta|} \leq 9\lambda n_\square.$$

In words, the number of points of $A \cup B$ that are close to the divider $\Gamma_\square$ is $O(n_\square\lambda)$ $\square$



(a) (b)

Figure 6.3: (a) a $\mathcal{C}$-extended 17-matching with 8 matched points (blue discs), 9 boundary-matched points (blue circles), and 4 free points (blue crosses). The matching cost is the total length of the solid and dashed lines. The boundary-matched point b is matched to the divider Γ, and (b) a perfect $\mathcal{C}$-extended matching, where all points of B are either matched to a point in A or to the boundaries of the cells in $\mathcal{C}$.

6.1.2 Extended Bipartite Matching

$$w_{\mathcal{C}}(M^{\mathcal{C}}) := \sum_{(a,b)\in M} \mathrm{c}(a,b) + \sum_{b\in B^{\mathcal{C}}} \mathrm{c}(b,\mathcal{C}). \tag{6.1}$$

We refer to all points of $B^{\mathcal{C}}$ as *boundary-matched*. All points of B that are neither matched in M nor boundary-matched are considered free. All points of A that are not matched in M are also considered free. The size of $M^{\mathcal{C}}$ is equal to $|M| + |B^{\mathcal{C}}|$. When $\mathcal{C}$ is clear from the context, we refer to $M^{\mathcal{C}}$ as an extended matching. An extended matching $M^{\mathcal{C}}$ is called *minimum-cost $\mathcal{C}$-extended t-matching* if it has a size t and has a minimum cost, and is called *minimum-cost perfect $\mathcal{C}$-extended matching* if it has a size n and has a minimum cost. In a minimum-cost perfect extended matching, all points of B are either matched to another point in A or to the boundaries of the cells in $\mathcal{C}$.

Consider the partitioning $\mathcal{C}^* = \{\square^*\}$, where $\square^*$ is the root cell of $\mathcal{H}$. Using the fact that

the input points are far from the boundaries of $\square^*$, we show in Lemma 6.2 below that any minimum-cost $\mathcal{C}^*$-perfect extended matching $M^{\mathcal{C}^*}$ is also a minimum-cost perfect matching between A and 4B, i.e., no points in B are boundary-matched in $M^{\mathcal{C}^*}$.

Lemma 6.2. *Suppose $\mathcal{C}^* = \{\square^*\}$, where $\square^*$ is the root cell of $\mathcal{H}$. Let $M^{\mathcal{C}^*} = (M, B^{\mathcal{C}^*})$ be a minimum-cost perfect extended matching. Then, the matching M is a minimum-cost perfect matching between A and B.*

Proof. Intuitively, our construction guarantees that for any point $b \in B$, the distance of b to the boundaries of $\square^*$ is more than the distance of b to any point $a \in A$. In this case, any boundary-matched point $b \in B^{\mathcal{C}^*}$ will contribute a high cost of $\mathrm{c}(b, \mathcal{C}^*)$ to $w_{\mathcal{C}^*}M^{\mathcal{C}^*}$ and therefore, matching b to any free point of A will reduce the cost of $M^{\mathcal{C}^*}$, which is a contradiction to the assumption that $M^{\mathcal{C}^*}$ is a minimum-cost $\mathcal{C}^*$-perfect extended matching. We give the details below.

For the sake of contradiction, suppose the matching M is not a perfect matching, and there exists a boundary-matched point $b \in B^{\mathcal{C}^*}$. By construction, the distance of b to the boundary of $\square^*$ is at least 2, i.e., $\mathrm{c}(b, \mathcal{C}^*) \geq 2$. Therefore, the contribution of b to $w_{\mathcal{C}^*}(M^{\mathcal{C}^*})$ is $\mathrm{c}(b, \mathcal{C}^*) \geq 2$. Since $|A| = |B|$, there exists at least one free point in A. For any free point $a \in A$, since both a and b are inside the unit square, $\|a - b\| \leq \sqrt{2} < 2$; therefore, by adding the edge (a, b) to the matching M of $M^{\mathcal{C}^*}$, the change in the cost of $M^{\mathcal{C}^*}$ would be $\mathrm{c}(a, b) - \mathrm{c}(b, \mathcal{C}^*) < 0$, i.e., matching the points b to the free point a results in an extended matching with a lower cost, which is a contradiction. Therefore, the matching $M^{\mathcal{C}^*}$ cannot have any free points, and the matching M of $M^{\mathcal{C}^*}$ is a perfect matching.

Finally, note that since the matching M of $M^{\mathcal{C}^*}$ is a perfect matching and there are no boundary-matched points in B, the cost of $M^{\mathcal{C}^*}$ and M are the same. Since $M^{\mathcal{C}^*}$ is a minimum-cost perfect extended matching, the matching M is also a minimum-cost perfect

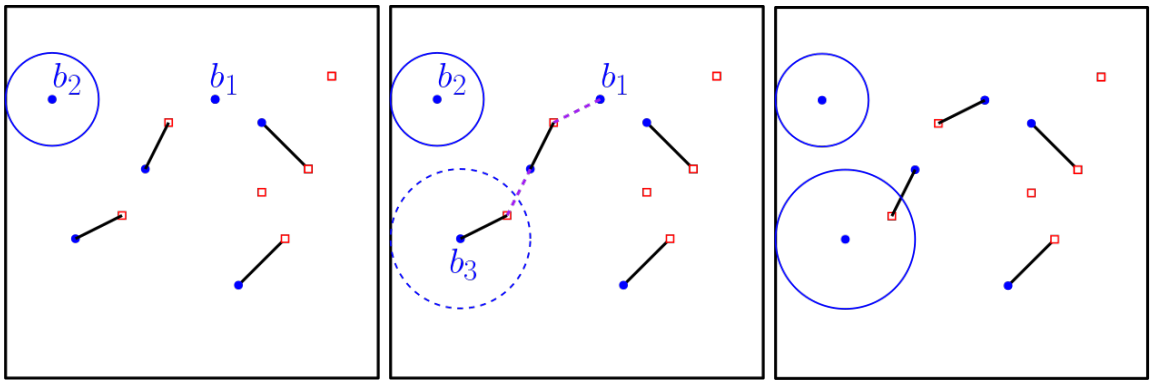


Figure 6.4: (a) The subset of a $\mathcal{C}$-feasible extended matching $M^{\mathcal{C}}$ that lies inside a cell $\square \in \mathcal{C}$, where b_1 is a free point and b_2 is a boundary-matched point, (b) an admissible extended augmenting path P (dashed line) from b_1 to a point b_3 whose vertex slack is zero, and (c) the $\mathcal{C}$-feasible extended matching obtained from augmenting $M^{\mathcal{C}}$ along P.

matching. □

Let $M^{\mathcal{C}} = (M, B^{\mathcal{C}})$ denote a $\mathcal{C}$-extended matching. Any path P whose edges alternate between matching and non-matching edges in M is called an alternating path. An alternating path P is called an *extended augmenting path* if P starts from a free point $b \in B$ and ends with either (i) a free point $a \in A$, or (ii) a matched or boundary-matched point $b' \in B$. We *augment* the extended matching $M^{\mathcal{C}}$ along an extended augmenting path P by updating its matching $M \leftarrow M \oplus P$, and for case (ii), we match b' to the boundary and update $B^{\mathcal{C}}$ to include b'. See Figure 6.4.

From Lemma 6.2, one can compute a minimum-cost perfect matching between A and B by computing a minimum-cost $\mathcal{C}^*$-perfect extended matching between A and B. In the next section, we present a primal-dual framework that our algorithm uses to efficiently compute minimum-cost extended matchings.

6.1.3 A Constrained Dual Formulation for Extended Matchings

Suppose $\mathcal{C}$ is a set of cells of $\mathcal{H}$ partitioning the root cell $\square^*$. Consider a $\mathcal{C}$-extended matching $M^{\mathcal{C}} = (M, B^{\mathcal{C}})$ between A and B along with a set of non-negative dual weights $y : A \cup B \to \mathbb{R}_{\geq 0}$. Let A_F be the set of free points of A with respect to $M^{\mathcal{C}}$. We say that $M^{\mathcal{C}}, y(\cdot)$ is *$\mathcal{C}$-feasible* if,

$$y(b) - y(a) \leq \mathrm{c}(a, b), \qquad \forall (a, b) \in A \times B, \tag{6.2}$$

$$y(b) - y(a) = \mathrm{c}(a, b), \qquad \forall (a, b) \in M, \tag{6.3}$$

$$y(b) \leq \mathrm{c}(b, \mathcal{C}), \qquad \forall b \in B, \tag{6.4}$$

$$y(b) = \mathrm{c}(b, \mathcal{C}), \qquad \forall b \in B^{\mathcal{C}}, \tag{6.5}$$

$$y(a) = 0, \qquad \forall a \in A_F. \tag{6.6}$$

The following lemma shows that any $\mathcal{C}$-feasible perfect extended matching has a minimum cost.

Lemma 6.3. *Suppose $M^{\mathcal{C}} = (M, B^{\mathcal{C}}), y(\cdot)$ is a $\mathcal{C}$-feasible perfect extended matching. Then, $M^{\mathcal{C}}$ is a minimum-cost perfect $\mathcal{C}$-extended matching.*

Proof. Using the $\mathcal{C}$-feasibility of the extended matching $M^{\mathcal{C}}, y(\cdot)$, we rewrite the cost of $M^{\mathcal{C}}$

as follows:

$$
\begin{aligned}
w_{\mathcal{C}}(M^{\mathcal{C}}) &= \sum_{(a,b)\in M} \mathrm{c}(a,b) + \sum_{b\in B^{\mathcal{C}}} \mathrm{c}(b,\mathcal{C}) \\
&= \left(\sum_{(a,b)\in M} y(b) - y(a)\right) + \sum_{b\in B^{\mathcal{C}}} y(b) && \text{From (6.3) and (6.5)} \\
&= \left(\sum_{(a,b)\in M} y(b) - y(a)\right) + \sum_{b\in B^{\mathcal{C}}} y(b) - \sum_{a\in A_F} y(a) && \text{From (6.6)} \\
&= \sum_{b\in B} y(b) - \sum_{a\in A} y(a) && \text{Since } M^{\mathcal{C}} \text{ is perfect.} \quad (6.7)
\end{aligned}
$$

Let $\hat{M}^{\mathcal{C}} = (\hat{M}, \hat{B}^{\mathcal{C}})$ denote any minimum-cost perfect $\mathcal{C}$-extended matching. We rewrite the cost of $\hat{M}^{\mathcal{C}}$ as follows:

$$
\begin{aligned}
w_{\mathcal{C}}(\hat{M}^{\mathcal{C}}) &= \sum_{(a,b)\in \hat{M}} \mathrm{c}(a,b) + \sum_{b\in \hat{B}^{\mathcal{C}}} \mathrm{c}(b,\mathcal{C}) \\
&\geq \left(\sum_{(a,b)\in \hat{M}} y(b) - y(a)\right) + \sum_{b\in \hat{B}^{\mathcal{C}}} y(b) && \text{From (6.2) and (6.4)} \\
&\geq \left(\sum_{(a,b)\in \hat{M}} y(b) - y(a)\right) + \sum_{b\in \hat{B}^{\mathcal{C}}} y(b) - \sum_{a\in A_F} y(a) && \text{Since } y(a) \geq 0 \forall a \in A \\
&= \sum_{b\in B} y(b) - \sum_{a\in A} y(a) && \text{Since } \hat{M}^{\mathcal{C}} \text{ is perfect.} \quad (6.8)
\end{aligned}
$$

Combining Equations (6.7) and (6.8),

$$
w_{\mathcal{C}}(M^{\mathcal{C}}) = \sum_{b\in B} y(b) - \sum_{a\in A} y(a) \leq w_{\mathcal{C}}(\hat{M}^{\mathcal{C}}).
$$

Since $\hat{M}^{\mathcal{C}}$ is a minimum-cost perfect extended matching, we conclude that $M^{\mathcal{C}}$ is also a minimum-cost perfect extended matching. □

For any pair $(a, b) \in A \times B$, we define the *slack* of (a, b) as $s(a, b) := \mathrm{c}(a, b) - y(b) + y(a)$. The edge (a, b) is *admissible* if $s(a, b) = 0$. We extend the definition of slacks to the points of B and define the *slack* of any point $b \in B$ as

$$s(b) := \mathrm{c}(b, \mathcal{C}) - y(b).$$

For any $\mathcal{C}$-feasible extended matching $M^{\mathcal{C}}, y(\cdot)$, the slack of every edge as well as every point $b \in B$ is non-negative. Recall that an extended augmenting path P is a path from a free point $b \in B$ to either (i) a free point $a \in A$ or (ii) a point $b' \in B$. The path P is *admissible* if all of its edges are admissible, and in case (ii), the slack of the end-point b' is $s(b') = 0$. The following properties of $\mathcal{C}$-feasible extended matchings are critical in the design of an efficient and correct algorithm.

Lemma 6.4. *For any $\mathcal{C}$-feasible extended matching $M^{\mathcal{C}} = (M, B^{\mathcal{C}}), y(\cdot)$,*

(a) no matching edges in M cross the boundaries of the cells in $\mathcal{C}$,

(b) all points in P lie in the same cell of $\mathcal{C}$, and

(c) the matching obtained after augmenting $M^{\mathcal{C}}$ along P remains $\mathcal{C}$-feasible.

Proof. Assume, for contradiction, that there exists a matching edge $(a, b) \in M$ such that a and b lie inside different cells $\square_a$ and $\square_b$ of $\mathcal{C}$. In this case, from condition (6.3),

$$\mathrm{c}(a, b) = y(b) - y(a) \leq y(b).$$

On the other hand, if a lies outside of $\square_b$, then

$$\mathrm{c}(b, \mathcal{C}) = \mathrm{c}(b, \square_b) < \mathrm{c}(a, b).$$

This, however, violates condition (6.4), since

$$y(b) \geq \mathrm{c}(a,b) > \mathrm{c}(b,\mathcal{C}),$$

which is a contradiction. Hence, no matching edges cross the boundaries of the cells in $\mathcal{C}$.

Next, consider an admissible extended augmenting path P. Assume, for contradiction, that there is an edge $(a,b) \in P$ such that a and b lie inside different cells $\square_a$ and $\square_b$ of $\mathcal{C}$. In this case, from the admissibility of P, the edge (a,b) is admissible, and therefore,

$$\mathrm{c}(a,b) = y(b) - y(a) \leq y(b).$$

Since a lies outside of $\square_a$,

$$\mathrm{c}(b,\mathcal{C}) = \mathrm{c}(b,\square_b) < \mathrm{c}(a,b).$$

Hence, the $\mathcal{C}$-feasibility on the point b is violated since

$$y(b) \geq \mathrm{c}(a,b) > \mathrm{c}(b,\mathcal{C}),$$

which is a contradiction.

Finally, we show that augmenting an extended matching along an admissible extended augmenting path does not violate the $\mathcal{C}$-feasibility conditions. For any edge $(a,b) \in P$, due to the admissibility of the edge, $y(b) - y(a) = \mathrm{c}(a,b)$. If $(a,b) \notin M$ prior to augmentation, it is a matching edge after the augmentation and condition (6.3) holds for (a,b). Otherwise, $(a,b) \in M$ prior to augmentation, it is a non-matching edge after the augmentation, and condition (6.2) holds for (a,b). Furthermore, if P is an extended augmenting path that ends at a point $b' \in B$, then b' has a zero slack, i.e., $y(b') = \mathrm{c}(b',\mathcal{C})$, and matching b' to the boundaries does not violate condition (6.5). For all other conditions, note that all dual

weights remain unchanged and consequently, conditions (6.4)–(6.6) remain satisfied. □

The properties described in Lemma 6.4 are important for the design of an efficient algorithm. Unlike the Hungarian algorithm, which searches the entire graph for admissible augmenting path, our algorithm can find an admissible extended augmenting path by conducting the search inside a cell $\square \in \mathcal{C}$ and augmenting the matching along the computed path. Thus, we can replace a global search with a search inside each cell of $\mathcal{C}$.

The next lemma provides critical properties that allow for correctly and efficiently merging cells in $\mathcal{C}$.

Lemma 6.5. *For a cell $\square$ of $\mathcal{H}$, suppose $\square'$ and $\square''$ denote its two children, and let $\mathcal{C}$ denote a partitioning containing $\square'$ and $\square''$. Let $\mathcal{C}' = \mathcal{C} \cup \{\square\} \setminus \{\square', \square''\}$. Given a $\mathcal{C}$-feasible extended matching $(M, B^{\mathcal{C}}), y(\cdot)$, let $B^{\mathcal{C}}_{\square} \subseteq B^{\mathcal{C}}$ denote the subset of boundary-matched points that are matched to the divider $\Gamma_{\square}$ of $\square$. Then, the $\mathcal{C}'$-extended matching $(M, B^{\mathcal{C}} \setminus B^{\mathcal{C}}_{\square}), y(\cdot)$ is also $\mathcal{C}'$-feasible.*

Proof. To prove this lemma, we show that the matching $\hat{M}^{\mathcal{C}} = (M, B^{\mathcal{C}} \setminus B^{\mathcal{C}}_{\square})$ along with the dual weights $y(\cdot)$ satisfy the extended feasibility conditions (6.2)–(6.6). First, note that for any edge $(a, b) \in A \times B$ (resp. matching edge $(a, b) \in M$), the dual weights of a and b, as well as their distance, is unchanged; hence, Condition (6.2) (resp. (6.3)) holds trivially. Similarly, for any free point $a \in A_F$, the dual weight of a remains zero, and for any point $b \in B \setminus B_{\square}$ outside of $\square$, the dual weight of b, as well as its distance to the boundaries of the partitioning, remains unchanged; therefore, Conditions (6.4)–(6.6) hold for all free points of A and all points of B that reside outside of $\square$. Next, we show that Conditions (6.4) and (6.5) are also true for the points of $B_{\square}$.

For any point $b \in B_{\square}$, let $\square_b \in \{\square', \square''\}$ be the child of $\square$ containing b. From the $\mathcal{C}$-feasibility

of $M^{\mathcal{C}}, y(\cdot)$,

$$y(b) \leq \mathrm{c}(b, \mathcal{C}) = \mathrm{c}(b, \square_b) \leq \mathrm{c}(b, \square) = \mathrm{c}(b, \mathcal{C}'),$$

and Condition (6.4) holds. For any boundary-matched point $b \in B_{\square} \cap (B^{\mathcal{C}} \setminus B^{\mathcal{C}}_{\square})$, since b is matched to the boundaries of $\square_b$ that are different from the divider $\Gamma_{\square}$, then $\mathrm{c}(b, \mathcal{C}) = \mathrm{c}(b, \mathcal{C}')$ and therefore, Condition (6.5) holds for all boundary-matched points of $\hat{M}^{\mathcal{C}}$ inside $\square$ as well. $\square$

From Lemma 6.5, erasing a divider does not cause the extended feasibility conditions to be violated. Despite preserving the extended feasibility conditions, erasing the boundary may result in the formation of free points in B, i.e., the points that were matched to the divider of the cell will now be free. In our algorithm in Section 6.2, we describe a process to adjust the matching $M^{\mathcal{C}}$ and dual weights $y(\cdot)$ and obtain a minimum-cost perfect extended matching.

Residual Graph. Similar to the Hungarian algorithm, we define a residual graph that assists in finding admissible extended augmenting paths. Consider a $\mathcal{C}$-feasible extended matching $M^{\mathcal{C}} = (M, B^{\mathcal{C}})$ and a set of dual weights $y(\cdot)$ for $A \cup B$. For each cell $\square \in \mathcal{C}$, we define a residual graph $\mathcal{G}_{\square}$. The vertex set of $\mathcal{G}_{\square}$ is a source vertex s and the points in $A_{\square} \cup B_{\square}$. For any edge $(a, b) \in A_{\square} \times B_{\square}$ inside $\square$, if $(a, b) \in M$ (resp. $(a, b) \notin M$), there is an edge with a weight $s(a, b)$ from a to b (resp. from b to a) in $\mathcal{G}_{\square}$. Additionally, there is an edge directed from s to all free points $b \in B$ with a weight $y(b)$.

6.2 Algorithm

Initialize $\mathcal{C}$ to the leaf cells of $\mathcal{H}$. Let $M^{\mathcal{C}} = (M, B^{\mathcal{C}})$ be the extended matching maintained by the algorithm and initialized to $M = \emptyset$ and $B^{\mathcal{C}} = B$, i.e., initially, we match all points

of B to the boundaries of the cells in $\mathcal{C}$. For each point $v \in A \cup B$, let $y(v)$ denote its dual weight. We initialize the dual weight of each point $b \in B$ to $y(b) = \mathrm{c}(b, \mathcal{C})$ and each point $a \in A$ to $y(a) = 0$. Execute the following steps until $\mathcal{C}$ contains only the root cell $\square^*$:

- Pick a cell $\square' \in \mathcal{C}$ with the smallest perimeter and let $\square$ and $\square''$ be the parent and sibling of $\square'$ in $\mathcal{H}$, respectively. Erase the divider of $\square$, i.e., set $\mathcal{C} = \mathcal{C} \setminus \{\square', \square''\} \cup \{\square\}$ and add all points $b \in B_\square$ that were matched to the divider $\Gamma_\square$ to the free points. While there exists a free point $b \in B_\square$, execute the EXTENDEDHUNGARIANSEARCH procedure, which updates the dual weights, computes an admissible extended augmenting path inside $\square$, and augments the extended matching $M^{\mathcal{C}}$ along the computed path. After this step, the extended matching $M^{\mathcal{C}}$ is perfect and $M^{\mathcal{C}}, y(\cdot)$ is $\mathcal{C}$-feasible.

When $\mathcal{C} = \{\square^*\}$, return the matching M of the extended matching $M^{\mathcal{C}}$ maintained by our algorithm as a minimum-cost perfect matching between A and B.

6.2.1 EXTENDEDHUNGARIANSEARCH Procedure

Given a $\mathcal{C}$-feasible matching $M^{\mathcal{C}}, y(\cdot)$ and a cell $\square \in \mathcal{C}$ such that $B_\square$ contains free points with respect to $M^{\mathcal{C}}$, the EXTENDEDHUNGARIANSEARCH procedure computes an admissible extended augmenting path $P_\square$ and augments $M^{\mathcal{C}}$ along $P_\square$. This procedure is similar to the classical Hungarian search procedure executed on $\mathcal{G}_\square$ and is mildly modified to include the augmenting paths that end at the boundary of $\square$. Details of the procedure are as follows.

1. *Update duals:* With s as the source vertex, execute Dijkstra's shortest path algorithm on $\mathcal{G}_\square$. Let P_v be the shortest path from s to each $v \in A_\square \cup B_\square$ and let κ_v be its cost. Define

$$\kappa = \min\{\min_{b \in B_\square} \kappa_b + s(b), \min_{a \in A_\square^F} \kappa_a\}. \tag{6.9}$$

For any $v \in A_\square \cup B_\square$, if $\kappa_v < \kappa$, set $y(v) \leftarrow y(v) + \kappa - \kappa_v$.

2. *Augment:* Let $v \in A_\square^F \cup B_\square$ be the point realizing the minimum distance in Equation (6.9). Define P as the augmenting path obtained by removing s from P_v. Augment $M^{\mathcal{C}}$ along P.

Lemma 6.6 establishes the properties of the ExtendedHungarianSearch procedure.

Lemma 6.6. *After the execution of the* ExtendedHungarianSearch *procedure for a cell* $\square$*, the extended matching* $M^{\mathcal{C}}, y(\cdot)$ *remains* $\mathcal{C}$*-feasible. Furthermore, the number of free points of* B *is reduced by one.*

Proof. In our proof, we first show that after the update duals step, the matching $M^{\mathcal{C}}$ along with dual weights $y(\cdot)$ remain $\mathcal{C}$-feasible and the path P is an admissible extended augmenting path.

Feasibility of points. For any point $b \in B$, if $b \in B \setminus B_\square$ is outside of $\square$, then $y(b) = y'(b)$; therefore, conditions (6.4) and (6.5) remain satisfied. Otherwise, $b \in B_\square$ and $\kappa \leq \kappa_b + s(b)$. Therefore, condition (6.4) holds for b since

$$y(b) = y'(b) + \kappa - \kappa_b \leq y'(b) + s(b) = \mathrm{c}(b, \square). \tag{6.10}$$

Similarly, for any free point $a \in A_F$, if $a \in A \setminus A_\square$ is a free point outside of $\square$, then $y(a) = y'(a) = 0$ and condition (6.6) remains true. Otherwise, $a \in A_\square$ is a free point inside $\square$. In this case, $\kappa_a \geq \kappa$ and the procedure does not update the dual weight of a, i.e., $y(a) = y'(a) = 0$, satisfying condition (6.6).

Feasibility of edges. For any edge $(a, b) \in A \times B$, let $s(a, b)$ denote the slack of (a, b) before updating the dual weights. For any edge $(a, b) \in A \times B$:

1. if $a \in A \setminus A_\square$ and $b \in B \setminus B_\square$, then $y(a) = y'(a)$ and $y(b) = y'(b)$; furthermore, the ExtendedHungarianSearch procedure does not add (resp. remove) the edge (a, b) to (resp. from) the matching and therefore, the feasibility conditions (6.2) and (6.3) remains satisfied for (a, b).

2. if $a \in A_\square$ and $b \in B \setminus B_\square$, then $y(b) = y'(b)$ and $y(a) \geq y'(a)$, since the ExtendedHungarianSearch procedure does not decrease the dual weight of any point inside $\square$; hence,

$$y(b) - y(a) \leq y'(b) - y'(a) \leq \mathrm{c}(a, b)$$

and conditions (6.2) is satisfied. (Note that by Lemma 6.4(a), the edge (a, b) cannot be a matching edge).

3. if $a \in A \setminus A_\square$ and $b \in B_\square$, then $y(b) \leq \mathrm{c}(b, \mathcal{C}') \leq \mathrm{c}(a, b)$; hence,

$$y(b) - y(a) \leq y(b) \leq \mathrm{c}(a, b)$$

and conditions (6.2) is satisfied. (Note that by Lemma 6.4(a), the edge (a, b) cannot be a matching edge).

4. if $a \in A_\square$ and $b \in B_\square$:

   - If $(a, b) \in M$ is a matching edge, then $\kappa_b = \kappa_a$ since the only edge directed to b in the residual graph is the zero-slack edge (a, b). Thus, the feasibility condition (6.3) holds since

   $$y(b) - y(a) = (y'(b) + \kappa - \kappa_b) - (y'(a) + \kappa - \kappa_a) = y'(b) - y'(a) = \mathrm{c}(a, b).$$

   - Otherwise, for any non-matching edge (a, b), $\kappa_a \leq \kappa_b + s(b, a)$, and the feasibility

condition (6.2) holds since

$$y(b) - y(a) = (y'(b) + \kappa - \kappa_b) - (y'(a) + \kappa - \kappa_a) \leq y'(b) - y'(a) + s(b,a) = \mathrm{c}(a,b). \tag{6.11}$$

Finally, note that each execution of the ExtendedHungarianSearch procedure augments the matching along an admissible extended augmenting path from a free point in $B_\square$ to either a free point in $A_\square$ or a matched point in $B_\square$. Since all points of B along the path will be matched or boundary-matched after augmenting the extended matching, the number of free points in $B_\square$ would be reduced by one.

□

In the next section, we provide the correctness and efficiency analysis of our algorithm.

6.3 Analysis

6.3.1 Correctness Analysis

Recollect that for a partitioning $\mathcal{C} = \{\square^*\}$ consisting only of the root square $\square^*$ of $\mathcal{H}$, from Lemma 6.2, a minimum-cost perfect $\mathcal{C}$-extended matching $M^{\mathcal{C}} = (M, B^{\mathcal{C}})$ does not match any point of B to the boundaries of $\square^*$ (i.e., $B^{\mathcal{C}}$ is empty) and therefore, the matching M of $M^{\mathcal{C}}$ is a minimum-cost perfect matching between A and B. Furthermore, from Lemma 6.3, for any $\mathcal{C}$-feasible perfect extended matching $M^{\mathcal{C}}, y(\cdot)$, the extended matching $M^{\mathcal{C}}$ is a minimum-cost perfect extended matching. Hence, it remains to show that at each step, the extended matching $M^{\mathcal{C}}, y(\cdot)$ maintained by our algorithm is a $\mathcal{C}$-feasible perfect extended matching.

Initially, our algorithm starts with a partitioning $\mathcal{C}$ consisting of all leaf cells of $\mathcal{H}$. It matches

all points of B to the boundaries of the leaf cells containing them by setting $M = \emptyset$ and $B^{\mathcal{C}} = B$, and assigning a dual weight $y(b) = \mathrm{c}(b, \mathcal{C})$ for all points $b \in B$ and $y(a) = 0$ for all points $a \in A$. Since each leaf cell contains only one point of $A \cup B$, this initial extended matching does not violate condition (6.2), and since the matching M is empty, it does not violate condition (6.3). Furthermore, the initial dual weight assignment to the points of A and B trivially satisfies the $\mathcal{C}$-feasibility conditions (6.4)–(6.6), and since all points of B are boundary-matched, the initial extended matching $M^{\mathcal{C}}, y(\cdot)$ is $\mathcal{C}$-feasible perfect extended matching.

Our algorithm then iteratively erases the divider of the cells in $\mathcal{C}$ and uses the ExtendedHungarianSearch procedure to match the newly formed free points of B (the points that were matched to the erased boundary) until all dividers are erased and the partitioning contains only the root cell. From Lemma 6.5, the matching obtained after erasing the divider of a cell does not violate the $\mathcal{C}$-feasibility conditions, and from Lemma 6.6, the extended matching maintained by the algorithm after executing the ExtendedHungarianSearch procedure remains $\mathcal{C}$-feasible. Also, from Lemma 6.6, the number of free points of B reduces by one after each execution of the ExtendedHungarianSearch procedure; hence, at the end of each step, our algorithms computes a $\mathcal{C}$-feasible perfect extended matching, as desired.

6.3.2 Efficiency Analysis

The ExtendedHungarianSearch procedure requires an execution of Dijkstra's shortest path algorithm on $\mathcal{G}_{\square}$. Using a DWBCP-DS structure with a query/update time of $\Phi(n)$, for any cell $\square$ of $\mathcal{H}$, each execution of the ExtendedHungarianSearch procedure inside $\square$ can be done in $O(n_{\square}\Phi(n_{\square}))$ time. In general settings, for any point sets A and B, when erasing the divider of any cell $\square$, we might introduce $O(n)$ free points, requiring $O(n)$ executions of the

ExtendedHungarianSearch procedure. Hence, our algorithm processes any cell $\square$ of $\mathcal{H}$ in $O(n_\square n\Phi(n))$ time, and an overall execution time of $O(n\Phi(n)\sum_{\square\in\mathcal{H}} n_\square)$. Since each point participates in at most $O(\log\Delta)$ cells in $\mathcal{H}$ (one cell per level), the total size of all cells of the hierarchical partitioning $\mathcal{H}$ is $\sum_{\square\in\mathcal{H}} n_\square = O(n\log\Delta)$. Consequently, our algorithm would run in $O(n^2\Phi(n)\log\Delta)$ time for any point sets A and B. In the following, assuming that the ground distance is the squared Euclidean distances, we show that for any set of $2n$ points U, a random subset A of n points of U, and the set $B = U \setminus A$, the efficiency of our algorithm improves to $O(n^{7/4}\Phi(n)\log\Delta)$. Our analysis easily extends to the pth power Euclidean distances. We present our analysis for any constant dimension $d \geq 2$ in Appendix B.1.2.

Recall that our algorithm maintains a perfect extended matching after each step. As a result, for any cell $\square$ of $\mathcal{H}$, once the divider $\Gamma_\square$ of $\square$ is removed, the only free points in the extended matching are those that were matched to the divider $\Gamma_\square$. In the remainder of this section, we show that the number of points that are matched to the divider of $\square$ is $\tilde{O}(n^{3/4})$; assuming this, the time to merge the children of $\square$ by erasing its divider and recomputing a $\mathcal{C}$-feasible perfect extended matching would be $O(n_\square n^{3/4}\Phi(n))$, leading to an overall running time of

$$O\left(n^{3/4}\Phi(n)\sum_{\square\in\mathcal{H}}\right) = O\left(n^{7/4}\Phi(n)\log\Delta\right).$$

Constructing a Partial Matching. In this part, we construct a matching M' of $A_\square \cup B_\square$ that, in expectation, has a cost $\tilde{O}(\ell_\square n_\square^{-1/4})$ and matches all except $\tilde{O}(n_\square^{3/4})$ points of $B_\square$. We begin by introducing a set of notations. Let $\square$ be any cell of $\mathcal{H}$, and let $\mathbb{G}$ be a grid dividing $\square$ into smaller squares. For any square $\xi \in \mathbb{G}$, let $A_\square^\xi$ (resp. $B_\square^\xi$) denote the subset of points of $A_\square$ (resp. $B_\square$) that lie inside ξ. Let the *excess* of ξ be $\text{exc}(\xi) := |B_\square^\xi| - |A_\square^\xi|$. Define $\text{exc}(G) := \sum_{\xi\in\mathcal{G}} \text{exc}(\xi)$ as the excess of $\mathbb{G}$. We next show an important property of randomly colored point sets, which is critical in constructing the matching.

Lemma 6.7. *For any square $\square$ of $\mathcal{H}$ and a grid $\mathbb{G}$ inside $\square$ with cell side length $O(\ell_\square n_\square^{-\alpha})$, $\mathbb{E}[\text{exc}(\mathbb{G})] = \tilde{O}(n_\square^{\alpha+\frac{1}{2}})$.*

Proof. If $\alpha \geq \frac{1}{2}$, the lemma statement holds trivially since $n_\square^{\alpha+\frac{1}{2}} \geq n_\square$. Therefore, we assume $\alpha \leq \frac{1}{2}$. Using the Hoeffding's inequality [84], for any hypercube ξ of G,

$$\Pr\left[|B_\square^\xi| - |A_\square^\xi| \geq c_1\sqrt{n_\square log n}\right] \leq n^{-c_2}$$

for some constants $c_1, c_2 > 1$. Since $\text{exc}(\xi) \leq n_\square$,

$$\mathbb{E}[\text{exc}(\xi)] = O(\sqrt{n_\square \log n}).$$

Summing over all squares of G,

$$\mathbb{E}[\text{exc}(G)] = \sum_{\xi \in G} \mathbb{E}[\text{exc}(\xi)] = O\left(\sum_{\xi \in G} \sqrt{n_\square \log n}\right) = O\left(\sqrt{n \log n \times |G|}\right) = \tilde{O}(n^{\alpha+\frac{1}{2}}).$$

□

We next use Lemma 6.7 to show that there exists a low-cost high-cardinality matching inside each sub-problem.

Lemma 6.8. *For any cell $\square$ of $\mathcal{H}$, there exists a matching M' that, in expectation, has a cost $O(\ell_\square^2 n_\square^{1/4})$ and matches all except $O(n_\square^{3/4})$ points of $B_\square$.*

Proof. Let $\langle \mathbb{G}_1, \ldots, \mathbb{G}_t \rangle$ denote a sequence of grids, where $t = \lceil \log \log n \rceil$ and each grid $\mathbb{G}_i$ has a side-length $O(\ell_\square n_\square^{-\alpha_i})$ for

$$\alpha_i := \frac{1}{2} - \frac{2^t}{2^{t+2} - 3}\left(1 - \frac{1}{2^i}\right).$$

Using these grids, we construct a matching M' as follows. Let $A_\square^0 := A_\square$ and $B_\square^0 := B_\square$. Starting from $i = 1$, we compute a matching M_i from $B_\square^{i-1}$ to $A_\square^{i-1}$ that for each square of $\mathbb{G}_i$ matches as many points as possible inside the square. Define $A_\square^i$ (resp. $B_\square^i$) as the set of free points of $A_\square^{i-1}$ (resp. $B_\square^{i-1}$) and process the next grid $\mathbb{G}_{i+1}$. We continue this procedure until the last grid $\mathbb{G}_t$ is processed. Define $M' := \sum_{i=1}^{t} M_i$. In the following, we show that the total number of free points with respect to M' is $O\left(n_\square^{3/4}\right)$; we then show that $w(M') = O\left(\ell_\square^2 n_\square^{1/4}\right)$ and conclude the lemma statement.

The matching M matches as many points as possible inside each square of the grid $\mathbb{G}_t$. Therefore, the total number of unmatched points is equal to the excess of $\mathbb{G}_t$, which by Lemma 6.7 is

$$\mathbb{E}[\mathrm{exc}(\mathbb{G}_t)] = \tilde{O}\left(n_\square^{\alpha_t+\frac{1}{2}}\right) = O\left(n_\square^{\frac{3}{4}+\frac{1}{4(2^{t+2}-3)}}\right) = O\left(n_\square^{\frac{3}{4}+\frac{1}{16\log n-12}}\right) = O\left(n_\square^{\frac{3}{4}}\right). \tag{6.12}$$

We next analyze the expected cost of M'. By the linearity of expectation,

$$\mathbb{E}[w(M')] = \sum_{i=1}^{t} \mathbb{E}[w(M_i)]. \tag{6.13}$$

For $i = 1$, since all matching edges in M_1 have a squared Euclidean cost at most $O((\ell_\square n_\square^{-\alpha_1})^2)$, the cost of M_1 would be

$$w(M_1) = O\left(n_\square \times (\ell_\square n_\square^{-\alpha_1})^2\right) = O\left(\ell_\square^2 n_\square^{\frac{2^t}{2^{t+2}-3}}\right) = O\left(\ell_\square^2 n_\square^{\frac{1}{4}}\right). \tag{6.14}$$

Finally, for each $1 < i \leq t$, the matching M_i consists of matching edges with squared Euclidean costs of at most $O((\ell_\square n_\square^{-\alpha_i})^2)$. By Lemma 6.7, the number of matching edges in

M_i is $\tilde{O}(n_{\square}^{\alpha_{i-1}+\frac{1}{2}})$ in expectation; therefore,

$$\mathbb{E}[w(M_i)] = \tilde{O}\left(n_{\square}^{\alpha_{i-1}+\frac{1}{2}} \times (\ell_{\square} n_{\square}^{-\alpha_i})^2\right) = \tilde{O}\left(\ell_{\square}^2 n_{\square}^{\frac{2^t}{2^{t+2}-3}}\right) = \tilde{O}\left(\ell_{\square}^2 n_{\square}^{\frac{1}{4}}\right). \tag{6.15}$$

Combining Equations (6.13), (6.14), and (6.15), $\mathbb{E}[w(M)] = \tilde{O}\left(\ell_{\square}^2 n_{\square}^{\frac{1}{4}}\right)$. □

Next, for any cell $\square$ of $\mathcal{H}$, using Lemma 6.8, we show that the number of executions of the ExtendedHungarianSearch procedure on $\square$ is $\tilde{O}(n^{3/4})$.

Number of Iterations. Let $B_{\square}^{\mathcal{C}}$ denote the set of all boundary-matched points of $B_{\square}$ that are matched to the divider $\Gamma_{\square}$ in the extended matching maintained by our algorithm before erasing the divider of $\square$. When erasing the divider of $\square$, our algorithm creates $|B_{\square}^{\mathcal{C}}|$ free points. Each iteration of the ExtendedHungarianSearch procedure finds an extended augmenting path, which reduces the number of free points by one (see Lemma 6.6). Therefore, after erasing the divider of $\square$, the number of free points remaining inside $\square$ is $|\mathbb{B}_{\square}|$, and our algorithm executes one ExtendedHungarianSearch procedure for each remaining free point. To complete the proof, we next show that $|B_{\square}^{\mathcal{C}}| = O(n^{3/4})$.

Lemma 6.9. *For any cell $\square$ of $\mathcal{H}$, $|B_{\square}^{\mathcal{C}}| = O(n^{3/4})$.*

Proof. Let M' denote the matching constructed in Lemma 6.8, and let M denote the matching of the extended matching $M^{\mathcal{C}}$ maintained by our algorithm. Let $M_{\square}$ denote the subset of the matching edges of M that lie inside $\square$. Note that by lemma 6.4, no matching edges can cross the boundaries of the cells in $\mathcal{C}$. Let $\mathcal{P}_{\text{aug}}$ (resp. $\mathcal{P}_{\text{alt}}$) denote the set of augmenting paths (resp. alternating paths) with an endpoint in $B_{\square}^{\mathcal{C}}$ in the symmetric difference $M_{\square} \oplus M'$. Clearly, $|B_{\square}^{\mathcal{C}}| = |\mathcal{P}_{\text{alt}}| + |\mathcal{P}_{\text{aug}}|$. For the alternating paths, $|\mathcal{P}_{\text{alt}}| = O(n^{3/4})$ since each alternating path has one free endpoint in M' and the number of free points with respect to M'

is $O(n^{3/4})$. Next, we show that $|\mathcal{P}_{\text{aug}}| = O(n^{3/4})$.

Using the definition of the net-cost of an augmenting path,

$$\sum_{P \in \mathcal{P}_{\text{aug}}} \phi(P) = \sum_{P \in \mathcal{P}_{\text{aug}}} \left(\sum_{(a,b) \in P \cap M'} \mathrm{c}(a,b) - \sum_{(a,b) \in P \cap M_\square} \mathrm{c}(a,b) \right)$$
$$\leq \sum_{P \in \mathcal{P}_{\text{aug}}} \left(\sum_{(a,b) \in P \cap M'} \mathrm{c}(a,b) \right) \leq w(M'). \tag{6.16}$$

For each path $P \in \mathcal{P}_{\text{aug}}$, let $b_P \in B_\square$ and $a_P \in A_\square$ denote the two end-points of P. Define $B_{\text{aug}} := \{b_P : P \in \mathcal{P}_{\text{aug}}\}$. Recall that in Lemma 1.2, we showed that

$$\phi(P) = y(b_P) - y(a_P) + \sum_{(a,b) \in P} s(a,b).$$

Since the endpoint a_P is a free point with respect to the matching M of the extended matching $M^{\mathcal{C}}$ maintained by our algorithm, from the $\mathcal{C}$-feasibility condition (6.6), $y(a_P) = 0$. Using Equation (6.16) and the fact that the slacks are non-negative,

$$\sum_{b \in B_{\text{aug}}} y(b) = \sum_{P \in \mathcal{P}_{\text{aug}}} y(b_P) \leq \sum_{P \in \mathcal{P}_{\text{aug}}} \phi(P) \leq w(M'). \tag{6.17}$$

Define $\alpha := \ell_\square n^{-1/4}$. Each free point $b \in B_{\text{aug}}$ is called a *close* (resp. *far*) point if the Euclidean distance of b to the divider of $\square$ is at most (resp. more than) α. Define $B^{\text{far}}_{\text{aug}}$ (resp. $B^{\text{close}}_{\text{aug}}$) as the set of all far (resp. close) points of B_{aug}. By Lemma 6.1,

$$|B^{\text{close}}_{\text{aug}}| = O(n_\square n^{-1/4}) = O(n^{3/4}). \tag{6.18}$$

For each far point $b \in B^{\text{far}}_{\text{aug}}$, $y(b) = \mathrm{c}(b, \Gamma_\square) \geq \alpha^2$ since b is matched to the divider $\Gamma_\square$ (note

that the distance function $c(\cdot,\cdot)$ is the squared Euclidean distance). Therefore,

$$\sum_{b\in B_{\text{aug}}} y(b) \geq \sum_{b\in B^{\text{far}}_{\text{aug}}} y(b) \geq \alpha^2 \times |B^{\text{far}}_{\text{aug}}|. \tag{6.19}$$

Combining Equations (6.17) and (6.19),

$$|B^{\text{far}}_{\text{aug}}| \leq \frac{\sum_{b\in B_{\text{aug}}} y(b)}{\alpha^2} \leq \frac{w(M')}{\alpha^2} = O(n^{3/4}). \tag{6.20}$$

Combining with Equation (6.18),

$$|B^{\mathcal{C}}_{\square}| \leq |B_{\text{aug}}| + |B_{\text{alt}}| \leq |B^{\text{close}}_{\text{aug}}| + |B^{\text{far}}_{\text{aug}}| + |B_{\text{alt}}| = \tilde{O}(n^{3/4}).$$

□

Each iteration of the merge step and each execution of the search procedure takes $\tilde{O}(n_{\square}\Phi(n))$ time and therefore, the total execution time of our algorithm on $\square$ would be $\tilde{O}(n^{3/4}n_{\square}\Phi(n))$. Since each point participates in $O(\log\Delta)$ cells in $\mathcal{H}$, the total execution time of our algorithm across all cells would be $\tilde{O}(n^{7/4}\Phi(n)\log\Delta)$, proving Theorem 2.5 for 2-dimensional point sets. We prove Theorem 2.5 for any dimension $d \geq 2$ in Appendix B.1.2.

Chapter 7

A Partial-Matching-Based Exact Algorithm for the Server Problems

In this chapter, we extend our algorithm from Chapter 6 to compute a minimum-cost t-matching, for any $t \in [1, n]$, between two point sets A and B, each of size n. We show that for the instances of the bipartite matching generated by the k-sequence partitioning problem, our algorithm computes a minimum-cost $(n-k)$-matching (and consequently, the optimal solution to the k-SP problem) in $\tilde{O}(n^{1.8}\Phi(n)\log\Delta)$ time, where Δ is the spread of the points and $\Phi(n)$ is the query/update time of a **DWBCP-DS**Before describing our framework and algorithm, we provide an overview of our algorithm from Chapter 6 (from now on referred to by the GRS algorithm) and describe the challenges in extending the GRS algorithm to computing partial matchings.

Overview of the GRS algorithm. The GRS algorithm constructs a deterministic hierarchical partitioning $\mathcal{H}$ of the space, where each cell of $\mathcal{H}$ containing more than one point of $A \cup B$ is divided into two child cells using a vertical or a horizontal line, called the divider $\Gamma_\square$. The GRS algorithm uses this hierarchical partitioning to localize the search for augmenting paths. It maintains a subset of cells $\mathcal{C} \subset \mathcal{H}$ that partitions the root cell of $\mathcal{H}$ (and consequently, partitions the points in $A \cup B$). Recall that an extended matching $M^{\mathcal{C}} = (M, B^{\mathcal{C}})$ consists of a matching M in addition to a set of boundary-matched points $B^{\mathcal{C}}$, which are

a subset of points of B that are matched to the boundaries of the cells in $\mathcal{C}$. The algorithm starts with a partitioning $\mathcal{C}$ consisting of all leaf cells of $\mathcal{H}$ and initializes an extended matching that matches all points of B to the boundaries of the cells in $\mathcal{C}$. At each step, the algorithm merges two sibling cells in $\mathcal{C}$ by erasing their shared boundary and then iteratively computes admissible extended augmenting paths and augments the extended matching along such paths until all newly formed free points are matched or boundary-matched. Recall that an extended augmenting path is an alternating path from a free point $b \in B$ to either (i) a free point $a \in A$, or (ii) a point $b' \in B$. The GRS algorithm follows the steps mentioned above until all of the dividers of the cells are erased, and the partitioning $\mathcal{C}$ consists solely of the root cell of $\mathcal{H}$. We showed that, since the boundaries of the root cell are far from the points in $A \cup B$, no points of B will be matched to the boundaries of the root cell in any minimum-cost extended matching, and therefore, the matching maintained by the GRS algorithm when $\mathcal{C}$ contains only the root cell is a minimum-cost perfect matching.

Extension to partial matchings. Recall that the Hungarian algorithm computes a minimum-cost t-matching, for some parameter $t \in [1, n]$, by iteratively computing a minimum net-cost augmenting path and augmenting a matching along such a path until the size of the matching reaches t. The search for minimum net-cost augmenting paths requires a global search across all points, which takes $O(n\Phi(n))$ time, leading to an overall running time of $O(nt\Phi(n))$ time, or $O(n^2\Phi(n))$ when $t = \Theta(n)$.

In this chapter, we aim to extend our framework from Chapter 6 to computing minimum-cost t-matchings, where we localize the search for augmenting paths using the cells of the hierarchical partitioning $\mathcal{H}$. Similar to the GRS algorithm, we design a primal-dual framework to ensure the optimality of the middle-step solutions maintained by our algorithm. Unlike in perfect matchings, where the GRS algorithm matches all points of B inside the cells of the

partitioning $\mathcal{C}$, for computing minimum-cost t-matchings, the algorithm needs to leave $(n-t)$ points unmatched. To coordinate the placement of these $(n-t)$ unmatched points across all cells of $\mathcal{C}$, our algorithm maintains, for each cell $\square$ of $\mathcal{C}$, the net-cost of the minimum net-cost extended augmenting path on the points inside $\square$ and uses these stored values to find the next minimum net-cost extended augmenting path. In this way, our algorithm localizes the search for augmenting paths inside the cells of the partitioning $\mathcal{C}$ while ensuring the global optimality of the extended t-matching.

k-Sequence Partitioning Problem. Given a sequence $\sigma = \langle r_1, \ldots r_n \rangle$ of n requests and an integer $1 \leq k \leq n$, the k-SP problem asks for partitioning the requests in σ into k subsequences $\varsigma_1, \ldots, \varsigma_k$ such that the total cost of servicing each subsequence using a single server is minimized; more precisely, for each subsequence $\varsigma_i = \langle r'_1, \ldots, r'_m \rangle$, the cost of servicing ς_i by a single server is $w(\varsigma_i) = \sum_{i=1}^{m-1} \mathrm{c}(r'_i, r'_{i+1})$, and the cost of partitioning requests in σ into k subsequences $\varsigma_1, \ldots, \varsigma_k$ is $\sum_{i=1}^{k} w(\varsigma_i)$. The problem asks for a partitioning that minimizes this cost.

Given an input sequence σ of requests to the k-SP problem, recall that we construct a bipartite graph G_σ with a vertex set $A \cup B$ and a set of edges as follows. *Vertex Set:* For each request r_i, we create a vertex b_i (resp. a_i) in B (resp. A) and designate it as the entry (resp. exit) gate for request r_i. *Edge Set:* The exit gate a_i of request r_i is connected to the entry gate b_j of every subsequent request r_j with $j > i$ with an edge. The cost of this edge is $\mathrm{c}(a_i, b_j) = \|r_i - r_j\|_p$. Any minimum-cost $(n-k)$-matching M in G_σ can be used to find an optimal solution to the k-SP problem. See Figure 7.1.

In the remainder of this chapter, we explain our algorithm and its analysis for the bipartite matching instances generated by the k-SP problem. Our algorithm, however, easily extends to computing a minimum-cost t-matching in the geometric settings.

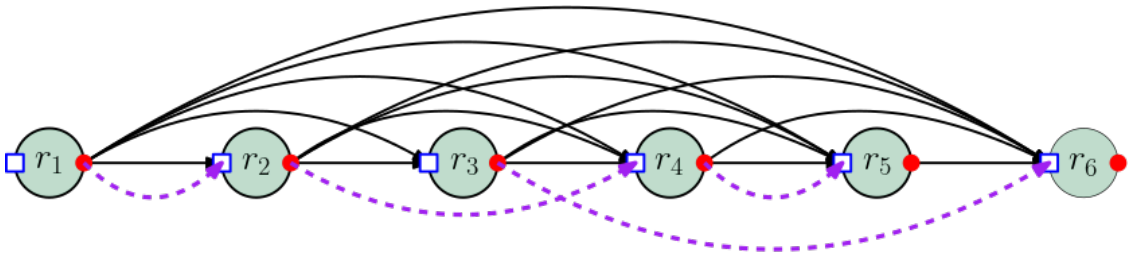


Figure 7.1: The graph $\mathcal{G}_\sigma$ constructed for $\sigma = \langle r_1, r_2, \ldots, r_6 \rangle$. The vertex set A (red disks) and B (blue squares) represent the exit and entry gates of each request, and the purple dashed lines show an $(n-2)$-partial matching on $\mathcal{G}_\sigma$ representing a 2-partitioning $\langle r_1, r_2, r_4, r_5 \rangle$ and $\langle r_3, r_6 \rangle$.

7.1 Geometric Primal-Dual Framework

Given a sequence of requests $\sigma = \langle r_1, r_2, \ldots, r_n \rangle$ inside the unit square and a parameter $k \in [1, n]$, in this section, we extend our primal-dual framework from Chapter 6 to compute a minimum-cost $(n-k)$-matching on the graph representation G_σ for the k-SP problem on the requests σ. We begin by describing the hierarchical partitioning scheme.

7.1.1 Hierarchical Partitioning

Our hierarchical partitioning in this section is similar to the one constructed in Section 7.1.1 with slight modifications. For completeness, we describe the construction next. Set $\lambda := 9n^{-1/5}$ and construct a hierarchical partitioning $\mathcal{H}$ recursively, where each node of $\mathcal{H}$ is an axis-parallel rectangle called a *cell*. The root node of $\mathcal{H}$ is a cell $\square^*$ defined as $\square^* := [-3n, 3n]^2$, containing all points in $A \cup B$. For each node $\square$, let $A_\square$ and $B_\square$ be the points of A and B inside $\square$ and let $n_\square = |A_\square \cup B_\square|$. If $n_\square \leq 2$ (i.e., $\square$ is empty or contains the entry and exit gates of a single request), then $\square$ is marked as a leaf node. Otherwise, we partition $\square$ into two smaller rectangles as follows. Let $\ell_\square$ be the larger of the length and width of rectangle $\square$. Assume, without loss of generality, that $\ell_\square$ is the width of $\square$. Let $x_{\min}$ be the

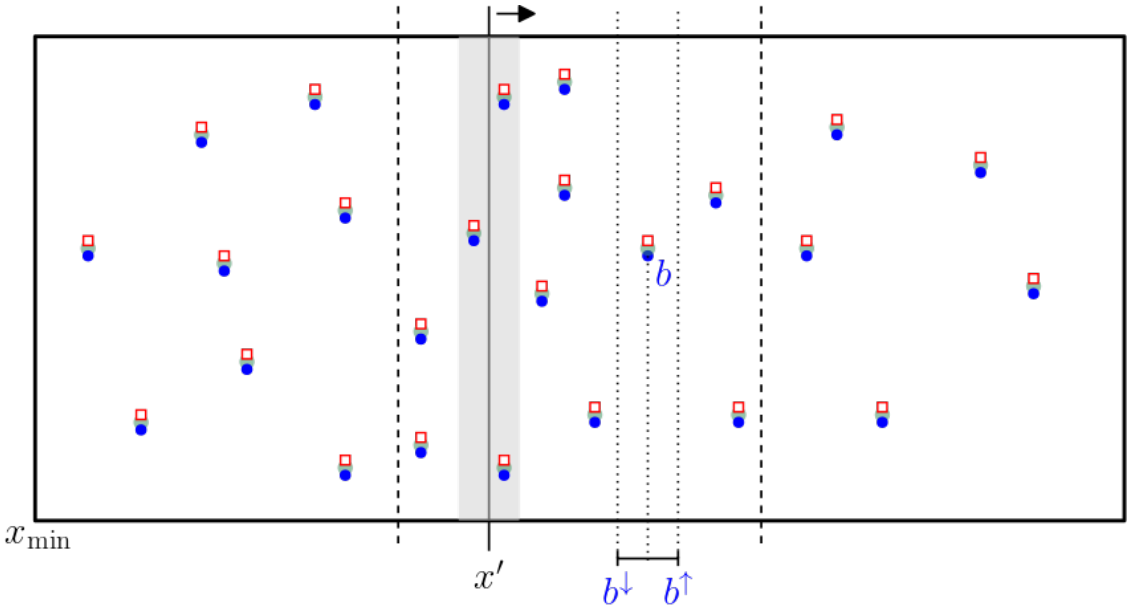


Figure 7.2: Partitioning a rectangle into two children using a sweep-line algorithm.

x-coordinate of the bottom-left corner of $\square$. For any value $\hat{x} \in [x_{\min} + \frac{\ell_\square}{3}, x_{\min} + \frac{2\ell_\square}{3}]$, define

$$\Lambda(\hat{x}) := \{u \in A_\square \cup B_\square : |u_x - \hat{x}| \leq \ell_\square \lambda\};$$

here, u_x denotes the x coordinate of the point u. Let

$$x^* := \arg \min_{\hat{x} \in [x_{\min} + \frac{\ell_\square}{3}, x_{\min} + \frac{2\ell_\square}{3}]} |\Lambda(\hat{x})|.$$

We partition $\square$ into two smaller rectangles by using a vertical line defined by $x = x^*$ and add them as the children of $\square$ to $\mathcal{H}$. We refer to the segment partitioning $\square$ into its two children as its *divider* and denote it by $\Gamma_\square$. This completes the construction of $\mathcal{H}$. Note that the height of the tree is $O(\log(n\Delta))$. A simple sweep-line algorithm, similar to the one described in Appendix B.1.1, can compute x^* in $O(n_\square \log n_\square)$ time. See Figure 7.2. Using this procedure, we construct $\mathcal{H}$ in $\tilde{O}(n \log(n\Delta))$ time.

We restate the important properties of the hierarchical partitioning that help analyze the

efficiency of our algorithm.

Lemma 6.1. *For each cell $\square$ of $\mathcal{H}$, the ratio of the largest to the smallest side of $\square$ is at most 3. Furthermore, the number of points of $A_\square \cup B_\square$ with a distance smaller than $\ell_\square \lambda$ to the divider $\Gamma_\square$ is $O(n_\square \lambda)$.*

The following structural property of the k-SP problem will be critical in bounding the efficiency of our algorithm.

Lemma 7.1. *For any cell $\square$ of $\mathcal{H}$, there exists a matching M' between $A_\square \cup B_\square$ that has a cost $O(\ell_\square n_\square^{3/5})$ and matches all except $O(n_\square^{4/5})$ points of $B_\square$.*

Proof. We place a grid $\mathbb{G}$ of side-length $\ell_\square n_\square^{-2/5}$ inside $\square$. The cells of $\mathbb{G}$ partition the requests of σ into $O(n_\square^{4/5})$ sub-sequences. For each cell ξ of $\mathbb{G}$, define $\sigma_\xi = \langle r'_1, \ldots, r'_m \rangle$ as the sub-sequence of σ inside ξ, and let A_ξ (resp. B_ξ) denote the entry and exit gates of σ_ξ. Let G_ξ be the graph representation of the requests in σ_σ. The graph G_ξ is a sub-graph of G_σ induced by $A_\xi \cup B_\xi$. Define a matching M_ξ corresponding to the 1-partitioning of σ_ξ; more precisely, $M_\xi := \{(a_{r'_i}, b_{r'_{i+1}})\}_{i \in [1, m-1]}$. Since each edge of M_ξ has a cost of at most $\ell_\square n_\square^{-2/5}$, we have $w(M_\xi) = O(\ell_\square |B_\xi| n_\square^{-2/5})$.

Define $M' := \bigcup_{\xi \in \mathbb{G}} M_\xi$. Since the total number of cells in the grid $\mathbb{G}$ is $O(n_\square^{4/5})$ and each matching M_ξ matches all except one point of B_ξ, the matching M' leaves $O(n_\square^{4/5})$ points of $B_\square$ unmatched. Furthermore,

$$w(M') = \sum_{\xi \in \mathbb{G}} w(M_\xi) = O\left(\ell_\square n_\square^{-2/5} \sum_{\xi \in \mathbb{G}} |B_\xi|\right) = O(\ell_\square n_\square^{3/5}).$$

This concludes the proof of the lemma. $\square$

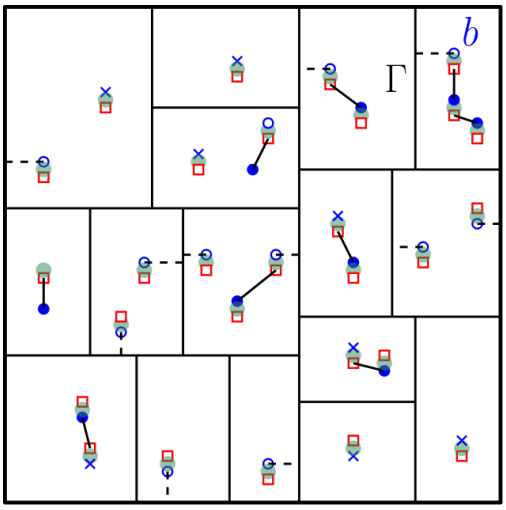


Figure 7.3: A $\mathcal{C}$-extended 20-matching with 9 matched points (blue discs), 11 boundary-matched points (blue circles), and 8 free points (blue crosses). The matching cost is the total length of the solid and dashed lines. The boundary-matched point b is matched to the divider Γ.

7.1.2 A Primal-Dual Framework for Partial Extended Matchings

Suppose $\mathcal{C}$ is a subset of cells of $\mathcal{H}$ that partitions $\square^*$. Recall that an extended matching $M^{\mathcal{C}} = (M, B^{\mathcal{C}})$ consists of a matching M between A and B and a subset $B^{\mathcal{C}}$ of the unmatched points of B with respect to M that are matched to the boundaries of cells in $\mathcal{C}$. See Figure 7.3. For any point $b \in B$, recall that $\mathrm{c}(b, \mathcal{C})$ denotes the shortest distance from b to the boundaries of the cells in $\mathcal{C}$. The cost of a $\mathcal{C}$-extended matching $M^{\mathcal{C}}$ is

$$w_{\mathcal{C}}(M^{\mathcal{C}}) := \sum_{(a,b)\in M} \mathrm{c}(a,b) + \sum_{b\in B^{\mathcal{C}}} \mathrm{c}(b,\mathcal{C}). \tag{7.1}$$

For each point $b \in B$ that is not matched in M, we refer to b as boundary-matched and free otherwise. All points of A that are not matched in M are also considered free. The size of $M^{\mathcal{C}}$ is defined as $|M| + |B^{\mathcal{C}}|$. See Figure 7.3. We refer to any extended matching of size t by *$\mathcal{C}$-extended t-matching*.

Consider the partitioning $\mathcal{C}^* = \{\square^*\}$, where $\square^*$ is the root cell of $\mathcal{H}$. Using the construction of $\mathcal{H}$, we show in Lemma 7.2 below that any minimum-cost $\mathcal{C}^*$-extended matching $M^{\mathcal{C}^*}$ of size t is also a minimum-cost t-matching, i.e., no point of B is boundary-matched in $M^{\mathcal{C}^*}$.

Lemma 7.2. *Suppose $\mathcal{C}^* = \{\square^*\}$, where $\square^*$ is the root cell of $\mathcal{H}$. Let $M^{\mathcal{C}^*} = (M, B^{\mathcal{C}^*})$ be a minimum-cost extended t-matching on G_σ, for $t \in [1, n]$. Then, the matching M is a minimum-cost t-matching between A and B.*

Proof. Assume, for contradiction, that the matching M is not a t-matching, and there exist boundary-matched points in $B^{\mathcal{C}^*}$. To prove this lemma, we show that there exists an extended t-matching $\hat{M}^{\mathcal{C}^*} = (\hat{M}, \hat{B}^{\mathcal{C}^*})$ such that $|\hat{M}| > |M|$ and $w_{\mathcal{C}^*}(\hat{M}^{\mathcal{C}^*}) < w_{\mathcal{C}^*}(M^{\mathcal{C}^*})$, contradicting the assumption that $M^{\mathcal{C}^*}$ is a min-cost extended t-matching.

Consider any augmenting path P with respect to the matching M, and let $b \in B$ denote the unmatched endpoint of P. Since the path P has a length of at most $2n - 1$, the net-cost of P is

$$\phi(P) = \sum_{(a,b)\in P\setminus M} \mathrm{c}(a, b) - \sum_{(a,b)\in P\cap M} \mathrm{c}(a, b) \leq \sum_{(a,b)\in P\setminus M} \mathrm{c}(a, b) \leq 2n;$$

here, the last inequality is resulted since all points are in the unit square and the ℓ_p distance of each pair is at most 2. By augmenting the matching M along the path P, we get a matching $\hat{M}$ with a cost $w(\hat{M}) = w(M) + \phi(P) \leq w(M) + 2n$. Also note that by the construction of $\square^*$, for any point $b' \in B$, $\mathrm{c}(b', \square^*) \geq 3n - 1 > 2n$. Let $\hat{b} \in B^{\mathcal{C}^*}$ denote an arbitrary boundary-matched point of $M^{\mathcal{C}^*}$. Consider any augmenting path P starting from $\hat{b}$ with respect to the matching M, and suppose $\hat{M}$ denotes the matching obtained by

augmenting M along P. Then, for the extended matching $\hat{M}^{\mathcal{C}^*} = (\hat{M}, B^{\mathcal{C}^*} \setminus \{\hat{b}\})$,

$$w_{\mathcal{C}^*}(\hat{M}^{\mathcal{C}^*}) = w(\hat{M}) + \sum_{b' \in B^{\mathcal{C}^*} \setminus \{\hat{b}\}} \mathrm{c}(b', \mathcal{C}^*) \leq w(M) + 2n + \sum_{b' \in B^{\mathcal{C}^*} \setminus \{\hat{b}\}} \mathrm{c}(b', \mathcal{C}^*)$$
$$< w(M) + \mathrm{c}(\hat{b}, \mathcal{C}^*) + \sum_{b' \in B^{\mathcal{C}^*} \setminus \{\hat{b}\}} \mathrm{c}(b', \mathcal{C}^*) = w_{\mathcal{C}^*}(M^{\mathcal{C}^*}).$$

Note that $\hat{M}^{\mathcal{C}^*}$ is also of size t, which is a contradiction to the assumption that $M^{\mathcal{C}^*}$ is a minimum-cost extended t-matching. □

Let $M^{\mathcal{C}} = (M, B^{\mathcal{C}})$ denote a $\mathcal{C}$-extended matching. Any path P on the graph G_σ whose edges alternate between matching and non-matching edges in M is called an *alternating path*. An alternating path P is called an *augmenting path* if P starts from a free point $b \in B$ and ends with either (i) a free point $a \in A$, or (ii) a point $b' \in B$. We *augment* the extended matching $M^{\mathcal{C}}$ along an augmenting path P by adding (resp. removing) any non-matching edge (resp. matching edge) of that path P to the matching M, and for case (ii), we match b' to the boundary and update $B^{\mathcal{C}}$ to include b'. The *net-cost* of P in case (i) is

$$\phi(P) := \sum_{(a,b) \in P \setminus M} \mathrm{c}(a, b) - \sum_{(a,b) \in P \cap M} \mathrm{c}(a, b),$$

and in case (ii) is

$$\phi(P) := \mathrm{c}(b', \mathcal{C}) + \sum_{(a,b) \in P \setminus M} \mathrm{c}(a, b) - \sum_{(a,b) \in P \cap M} \mathrm{c}(a, b).$$

Next, we present a dual formulation for computing minimum-cost extended t-matchings. The formulation is almost identical to the one presented in Section 6.1.3, with an additional condition that is required to guarantee the optimality of partial extended matchings.

For a partitioning $\mathcal{C}$ of the root cell $\square^*$ and a $\mathcal{C}$-extended t-matching $M^{\mathcal{C}} = (M, B^{\mathcal{C}})$ on G_σ, let A_F denote the free points of A with respect to $M^{\mathcal{C}}$. The extended matching $M^{\mathcal{C}}$ and a set of non-negative dual weights $y(\cdot)$ for $A \cup B$ is $\mathcal{C}$-feasible if,

$$y(b) - y(a) \leq \mathrm{c}(a, b), \qquad \forall (a, b) \in E, \tag{7.2}$$

$$y(b) - y(a) = \mathrm{c}(a, b), \qquad \forall (a, b) \in M, \tag{7.3}$$

$$y(b) \leq \mathrm{c}(b, \mathcal{C}), \qquad \forall b \in B, \tag{7.4}$$

$$y(b) = \mathrm{c}(b, \mathcal{C}), \qquad \forall b \in B^{\mathcal{C}}, \tag{7.5}$$

$$y(a) = 0, \qquad \forall a \in A_F. \tag{7.6}$$

For any edge $(a, b) \in E$, the *slack* of (a, b) is defined as $s(a, b) := \mathrm{c}(a, b) - y(b) + y(a)$. The edge (a, b) is *admissible* if $s(a, b) = 0$. We extend the definition of slacks to the points in B and define, for any point $b \in B$, a slack $s(b) := \mathrm{c}(b, \mathcal{C}) - y(b)$ for b. For any feasible extended matching $M^{\mathcal{C}}, y(\cdot)$, the slack of every edge as well as every point $b \in B$ is non-negative.

The following lemma shows that no edges of a $\mathcal{C}$-feasible extended matching crosses the boundaries of cell in $\mathcal{C}$.

Lemma 7.3. *For any $\mathcal{C}$-feasible extended matching $M^{\mathcal{C}} = (M, B^{\mathcal{C}}), y(\cdot)$, no edges of the matching M cross the boundaries of the cells in $\mathcal{C}$.*

Proof. For the sake of contradiction, suppose there is an edge $(a, b) \in M$, where a and b lie inside a cell $\square_a$ and $\square_b$ of $\mathcal{C}$ and $\square_a \neq \square_b$. By the $\mathcal{C}$-feasibility condition (7.3),

$$\mathrm{c}(a, b) = y(b) - y(a) \leq y(b).$$

Since a is outside of $\square_b$, then $\mathrm{c}(b, \square_b) < \mathrm{c}(a, b)$. Therefore,

$$y(b) \geq \mathrm{c}(a, b) \geq \mathrm{c}(b, \square_b) = \mathrm{c}(b, \mathcal{C}),$$

which is a contradiction to the assumption that $M^{\mathcal{C}}, y(\cdot)$ is feasible (condition (7.4) is violated). □

The following lemma is useful in proving important properties of extended matchings.

Lemma 7.4. *Given a feasible $\mathcal{C}$-extended matching $M^{\mathcal{C}} = (M, B^{\mathcal{C}}), y(\cdot)$ on $\mathcal{G}_\sigma$, let P be any augmenting path from a free point $b \in B$ with respect to $M^{\mathcal{C}}$. If P ends with a free point of A, then $\phi(P) = y(b) + \sum_{(a'',b'')\in P} s(a'', b'')$. If P ends with a matched point $b' \in B$, then*

$$\phi(P) = y(b) + s(b') + \sum_{(a'',b'')\in P} s(a'', b'').$$

Proof. Note that for any non-matching edge $(a, b) \in P$, $\mathrm{c}(a, b) = s(a, b) + y(b) - y(a)$. Furthermore, since the slacks of the matching edges are 0, for each matching edge $(a, b) \in M$, $-\mathrm{c}(a, b) = s(a, b) - y(b) + y(a)$. If P is an augmenting path from b to a free point $a \in A$ (case (i)), then

$$\begin{aligned}
\phi(P) &= \sum_{(a'',b'')\in P\setminus M} \mathrm{c}(a'', b'') - \sum_{(a'',b'')\in P\cap M} \mathrm{c}(a'', b'') \\
&= \sum_{(a'',b'')\in P\setminus M} (s(a'', b'') + y(b'') - y(a'')) + \sum_{(a'',b'')\in P\cap M} (s(a'', b'') - y(b'') + y(a'')) \\
&= \sum_{(a'',b'')\in P} s(a'', b'') + y(b) - y(a) = y(b) \sum_{(a'',b'')\in P} s(a'', b''),
\end{aligned}$$

where the last inequality holds since $y(a) = 0$ by Condition (7.6). Otherwise, P is an

alternating path from b to a matched point $b' \in B$ (case (ii)), and

$$\begin{aligned}\phi(P) &= \mathrm{c}(b', \square) + \sum_{(a'',b'')\in P\setminus M} \mathrm{c}(a'', b'') - \sum_{(a'',b'')\in P\cap M} \mathrm{c}(a'', b'') \\ &= \mathrm{c}(b', \square) + \sum_{(a'',b'')\in P\setminus M} (s(a'', b'') + y(b'') - y(a'')) + \sum_{(a'',b'')\in P\cap M} (s(a'', b'') - y(b'') + y(a'')) \\ &= \mathrm{c}(b', \square) + \sum_{(a'',b'')\in P} s(a'', b'') + y(b) - y(b') = y(b) + s(b') + \sum_{(a'',b'')\in P} s(a'', b''),\end{aligned}$$

where the last inequality holds since $s(b') = \mathrm{c}(b', \square) - y(b')$ by the definition of the slack of a point. □

The following is a straightforward corollary from Lemma 7.4 and the definition of admissible augmenting paths.

Corollary 7.5. *Given a feasible $\mathcal{C}$-extended matching $M^{\mathcal{C}} = (M, B^{\mathcal{C}}), y(\cdot)$ on $\mathcal{G}_\sigma$, let P be an admissible augmenting path from a free point $b \in B$ with respect to $M^{\mathcal{C}}$. Then, $\phi(P) = y(b)$.*

Recall that an extended augmenting path P starts at a free point $b \in B$ and ends at (i) a free point $a \in A$ or (ii) a point $b' \in B$. The path P is *admissible* if all edges of P are admissible and in case (ii), the slack of the end-point b' is $s(b') = 0$. The following properties of extended feasible matchings are critical in the design of an efficient and correct algorithm.

Lemma 7.6. *Given a feasible $\mathcal{C}$-extended t-matching $M^{\mathcal{C}} = (M, B^{\mathcal{C}})$ and a set of non-negative dual weights $y(\cdot)$ on $A \cup B$, let P be a minimum net-cost extended augmenting path with respect to $M^{\mathcal{C}}$. Let, for any $\square \in \mathcal{C}$, $y_\square = \max_{b'\in B_\square} y(b')$. Then,*

(a) all points of P lie inside a single cell of $\mathcal{C}$, and

(b) if, for every cell $\square \in \mathcal{C}$, $y_\square \le \phi(P)$ and for all free points $b \in B_\square$, $y(b) = y_\square$, then $M^{\mathcal{C}}$ is a minimum-cost extended t-matching.

Proof. Proof of (a). For the sake of contradiction, suppose $P = \langle b_1, a_1, b_2, \ldots, b_m, a_m \rangle$ be a minimum net-cost extended augmenting path that does not lie inside a single cell of $\mathcal{C}$. Let $\square$ denote the cell of $\mathcal{C}$ containing b_1, and let (b_i, a_i) be the first edge that goes outside of $\square$, i.e., all vertices $\{b_1, a_1, \ldots, b_i\}$ reside inside $\square$. For the alternating path $P' = \langle a_i, b_{i+1}, \ldots, a_m \rangle$, the net-cost of P' is

$$\begin{aligned}
\phi(P') &= \sum_{j=i+1}^{m} \mathrm{c}(a_j, b_j) - \sum_{j=i}^{m-1} \mathrm{c}(a_j, b_{j+1}) \\
&= \sum_{j=i+1}^{m} (s(a_j, b_j) + y(b_j) - y(a_j)) + \sum_{j=i}^{m-1} (s(a_j, b_{j+1}) - y(b_{j+1}) + y(a_j)) \\
&= \sum_{j=i+1}^{m} s(a_j, b_j) + \sum_{j=i}^{m-1} s(a_j, b_{j+1}) + y(a_i) - y(a_m) \geq 0,
\end{aligned}$$

where the last inequality holds since all edges have non-negative slack, all points have non-negative dual weights, and a_m is an unmatched point and by condition (7.6), has a zero dual weight.

In addition, the cost of matching b_i to the boundaries of $\square$ is less than the cost of matching it to a point a_i outside of $\square$, i.e., $\mathrm{c}(b_i, \mathcal{C}) \leq \mathrm{c}(a_i, b_i)$. Define $P'' = \langle b_1, a_1, \ldots, b_i \rangle$. Then,

$$\phi(P) = \phi(P'') + \mathrm{c}(a_i, b_i) + \phi(P') \geq \phi(P'') + \mathrm{c}(a_i, b_i) > \phi(P'') + \mathrm{c}(b_i, \mathcal{C}).$$

Therefore, the extended augmenting path P'', which is a path from the free point b_1 to the matched point b_i (case (ii)) has a lower net-cost than P, contradicting the assumption that P is a minimum net-cost extended augmenting path.

Proof of (b). We use Lemma 7.8 below to compute dual weights $y'(\cdot)$ such that $M^{\mathcal{C}}, y'(\cdot)$ is feasible, $y'(b) = \phi(P)$ for all free points $b \in B$, and $y'(b) \leq \phi(P)$ for all points $b \in B$. Let A_F (resp. B_F) denote the free points of A (resp. B) in the extended matching $M^{\mathcal{C}}$. Let

$y_{\max} := \max_{b \in B} y'(b)$. In this case, we have

$$\sum_{b \in B_F} y'(b) = |B_F| \cdot y_{\max}. \tag{7.7}$$

Using the $\mathcal{C}$-feasibility conditions, we rewrite the cost of $M^{\mathcal{C}}$ as

$$\begin{aligned} w_{\mathcal{C}}(M^{\mathcal{C}}) &= \sum_{(a,b) \in M} \mathrm{c}(a,b) + \sum_{b \in B^{\mathcal{C}}} \mathrm{c}(b, \mathcal{C}) \\ &= \sum_{(a,b) \in M} y'(b) - y'(a) + \sum_{b \in B^{\mathcal{C}}} y'(b) && \text{from (7.3) and (7.5)} \\ &= \left(\sum_{b \in B} y'(b) - \sum_{a \in A} y'(a) \right) - |B_F| \cdot y_{\max}. && \text{from (7.6) and (7.7)} \end{aligned} \tag{7.8}$$

Let $\hat{M}^{\mathcal{C}} = (\hat{M}, \hat{B}^{\mathcal{C}})$ denote any minimum-cost extended t-matching on G_σ. Let $\hat{A}_F$ (resp. $\hat{B}_F$) denote the free points of A (resp. B) in $\hat{M}^{\mathcal{C}}$. Note that both $M^{\mathcal{C}}$ and $\hat{M}^{\mathcal{C}}$ are t-matchings and have the same number of free points, i.e., $|B_F| = |\hat{B}_F|$. From the $\mathcal{C}$-feasibility conditions, we rewrite the cost of $\hat{M}^{\mathcal{C}}$ as

$$\begin{aligned} w_{\mathcal{C}}(\hat{M}^{\mathcal{C}}) &= \sum_{(a,b) \in \hat{M}} \mathrm{c}(a,b) + \sum_{b \in \hat{B}^{\mathcal{C}}} \mathrm{c}(b, \mathcal{C}) \\ &\geq \sum_{(a,b) \in \hat{M}} y'(b) - y'(a) + \sum_{b \in \hat{B}^{\mathcal{C}}} y'(b) && \text{from (7.2) and (7.4)} \\ &= \left(\sum_{b \in B} y'(b) - \sum_{a \in A} y'(a) \right) - \sum_{b \in \hat{B}_F} y'(b) + \sum_{a \in \hat{A}_F} y'(a) \\ &\geq \left(\sum_{b \in B} y'(b) - \sum_{a \in A} y'(a) \right) - |\hat{B}_F| \cdot y_{\max}, \end{aligned} \tag{7.9}$$

where the last inequality holds since all points have non-negative weights bounded by $y_{\max}$.

Combining Equations (7.8) and (7.9),

$$w_{\mathcal{C}}(M^{\mathcal{C}}) = \sum_{b \in B} y'(b) - \sum_{a \in A} y'(a) - |B_F| \cdot y_{\max} \leq w_{\mathcal{C}}(\hat{M}^{\mathcal{C}}).$$

Since $\hat{M}^{\mathcal{C}}$ is of minimum-cost, $w_{\mathcal{C}}(M^{\mathcal{C}}) = w_{\mathcal{C}}(\hat{M}^{\mathcal{C}})$, and we conclude that the extended matching $M^{\mathcal{C}}$ is also a minimum-cost extended t-matching. □

The property described in Lemma 7.6(a) is important for the design of an efficient algorithm. Unlike the Hungarian algorithm, which searches the entire graph for the minimum net-cost augmenting path, our algorithm can find the minimum net-cost extended augmenting path by searching for the cheapest extended augmenting path inside each cell $\square \in \mathcal{C}$ and then taking the smallest among them. Thus, we can replace a global search with a search inside each cell of $\mathcal{C}$. The property in Lemma 7.6(b) is important for the design of a correct algorithm since it provides conditions under which an extended matching is a minimum-cost extended matching. Our algorithm is designed to maintain these conditions as invariants during its execution.

Residual Graph. Similar to the Hungarian algorithm, we define a residual graph that assists in finding the minimum net-cost extended augmenting path. Consider a feasible $\mathcal{C}$-extended matching $M^{\mathcal{C}} = (M, B^{\mathcal{C}})$ along with a set of dual weights $y(\cdot)$ on points of $A \cup B$. For each cell $\square \in \mathcal{C}$, we define a residual graph $G_{\square}$. The vertex set of $G_{\square}$ is a source vertex s and the points in $A_{\square} \cup B_{\square}$. For any edge $(a, b) \in E$ inside $\square$, if $(a, b) \in M$ (resp. $(a, b) \notin M$), there is an edge directed from a to b (resp. from b to a) with a weight $s(a, b)$ in $G_{\square}$. Furthermore, there is an edge directed from s to every free point $b \in B$ with a weight $y(b)$.

Lemma 7.7 provides a method to update the dual weights, which is essential during the

process of merging cells.

Lemma 7.7. *Suppose $M^{\mathcal{C}}, y(\cdot)$ is a feasible $\mathcal{C}$-extended matching and P is a minimum net-cost extended augmenting path. For any cell $\square \in \mathcal{C}$, define $y_\square = \max_{b' \in B_\square} y(b')$, and suppose $y_\square \leq \phi(P)$ and $y(b_f) = y_\square$ for all free points $b_f \in B_\square$. Then, one can update the dual weights in $\tilde{O}(n_\square \Phi(n_\square))$ time such that $M^{\mathcal{C}}, y(\cdot)$ remains feasible, $y(b) \leq \phi(P)$ for all $b \in B_\square$, and $y(b_f) = \phi(P)$ for all free points $b_f \in B_\square$.*

Proof. Recall that $G_\square$ denotes the residual graph of G_σ inside $\square$. Define κ_v as the shortest path distance of each point $v \in A_\square \cup B_\square$ from the source vertex s. Define $\kappa = \phi(P)$. For any vertex $v \in A_\square \cup B_\square$ with $\kappa_v < \kappa$, set

$$y'(v) \leftarrow y(v) - \kappa_v + \kappa;$$

otherwise, set $y'(v) \leftarrow y(v)$. As discussed in Section 7.4, the shortest path distances κ_v can be computed in $\tilde{O}(n_\square \Phi(n_\square))$ time. Given the distances from the source, computing the set $y'(\cdot)$ of dual weights from $y(\cdot)$ takes $O(n_\square)$ time, and therefore, the total construction takes $\tilde{O}(n_\square \Phi(n_\square))$ time.

We next show that $M^{\mathcal{C}}, y'(\cdot)$ is feasible, $y'(b) \leq \phi(P)$ for all points $b \in A_\square \cup B_\square$, and $y'(b_f) = \phi(P)$ for all free points $b_f \in B_\square$. Note that by Lemma 7.3, since the extended matching $M^{\mathcal{C}}, y(\cdot)$ is feasible, then no matching edges of $M^{\mathcal{C}}$ cross the boundaries of the cells in $\mathcal{C}$.

Feasibility of edges. For any edge $(a, b) \in E$, let $s(a, b)$ denote the slack of (a, b) with respect to $y(\cdot)$. For any edge $(a, b) \in E$:

1. if $a \in A \setminus A_\square$ and $b \in B \setminus B_\square$, then $y'(a) = y(a)$ and $y'(b) = y(b)$; therefore, conditions (7.2) and (7.3) remains satisfied for (a, b).

2. if $a \in A_\square$ and $b \in B \setminus B_\square$, then the edge (a, b) is a non-matching edge, $y'(b) = y(b)$, and $y'(a) \geq y(a)$, since we do not decrease the dual weight of any point; hence, $y'(b) - y'(a) \leq y(b) - y(a) \leq \mathrm{c}(a, b)$ and condition (7.2) is satisfied.

3. if $a \in A \setminus A_\square$ and $b \in B_\square$, then (a, b) is a non-matching edge and $y'(b) \leq \mathrm{c}(b, \mathcal{C}') \leq \mathrm{c}(a, b)$; hence, $y'(b) - y'(a) \leq y'(b) \leq \mathrm{c}(a, b)$ and condition (7.2) is satisfied.

4. if $a \in A_\square$ and $b \in B_\square$:

   - If $(a, b) \in M$ is a matching edge, then $\kappa_b = \kappa_a$ since the only edge directed to b in the residual graph is the zero-slack edge (a, b). Thus, condition (7.3) holds since

   $$y'(b) - y'(a) = (y(b) + \kappa - \kappa_b) - (y(a) + \kappa - \kappa_a) = y(b) - y(a) = \mathrm{c}(a, b).$$

   - Otherwise, for any non-matching edge (a, b), $\kappa_a \leq \kappa_b + s(b, a)$, and the feasibility condition (7.2) holds since

   $$y'(b) - y'(a) = (y(b) + \kappa - \kappa_b) - (y(a) + \kappa - \kappa_a) \leq y(b) - y(a) + s(b, a) = \mathrm{c}(a, b).$$

   Note that if (a, b) is a non-matching edge on the shortest path tree, then $\kappa_a = \kappa_b + s(b, a)$ and $y'(b) - y'(a) = \mathrm{c}(a, b)$, i.e., (a, b) is admissible.

Feasibility of points.

1. For any point $b \in B \setminus B_\square$, $y'(b) = y(b)$ and conditions (7.4) and (7.5) holds. Similarly, for any free point $a \in A \setminus A_\square$, $y'(a) = y(a) = 0$ and condition (7.6) holds.

2. For any point $b \in B_\square$:

   - if b is a free point, then $\kappa_b = y(b)$, since the only path from s to b is an edge from s to b with a weight $y(b)$. In this case, $y'(b) = y(b) - \kappa_b + \kappa = \kappa$.

- if b is a boundary-matched point, then there are no edges coming into b and therefore, $y'(b) = y(b) = \mathrm{c}(b, \mathcal{C})$ and condition (7.5) holds.
- Otherwise, b is a matched point. In this case, we show that $\kappa \leq \kappa_b + s(b)$: Let P_b denote the shortest path from s to b, and let P' denote the path obtained by removing s from P_b, which has a free endpoint $b' \in B_\square$. If $\kappa > \kappa_b + s(b)$, then net-cost of the extended augmenting path P' is, by Lemma 7.4,

$$\phi(P') = y(b') + s(b) + \sum_{(a'',b'')\in P} s(a'', b'') = \kappa_b + s(b) < \kappa = \phi(P),$$

which contradicts the assumption that P is a minimum net-cost extended augmenting path. Therefore, $\kappa \leq \kappa_b + s(b)$ and condition (7.4) holds for b since

$$y'(b) = y(b) + \kappa - \kappa_b \leq y(b) + s_\square(b) = \mathrm{c}(b, \square). \tag{7.10}$$

3. For any free point $a \in A^F_\square$, $\kappa_a \geq \kappa$, since otherwise, if $\kappa_a < \kappa$, then the path from the source to a defines an extended augmenting path whose net-cost is $\kappa_a < \kappa = \phi(P)$, which contradicts the assumption that P is a minimum net-cost extended augmenting path. Thus, $\kappa_a \geq \kappa$ and the procedure does not update the dual weight of a, i.e., $y'(a) = 0$ satisfying condition (7.6).

Finally, we show that the dual weight of each point $v \in A_\square \cup B_\square$ is at most $\phi(P)$. For each point $b \in B_\square$ such that $y'(b) > y(b)$, we have $\kappa_b < \kappa$. Suppose P_b denotes the shortest path from the source s to b, and let P' be the path obtained by removing s from P_b. Let b' denote the free endpoint of P'. By the assumption of the lemma, $y(b') \geq y(b)$, and since all slacks are non-negative, $\kappa_{b'} \leq \kappa_b$; therefore,

$$y'(b) = y(b) - \kappa_b + \kappa \leq y(b') - \kappa_{b'} + \kappa = \kappa.$$

□

The following lemma is resulted from applying Lemma 7.7 on all cells $\square \in \mathcal{C}$.

Lemma 7.8. *Given a partitioning $\mathcal{C}$ and a feasible extended matching $M^{\mathcal{C}}, y(\cdot)$, let P denote a minimum net-cost extended augmenting path. Suppose $y(b) \leq \phi(P)$ for all points $b \in B$ and $y(b) = \max_{b' \in B_{\square}} y(b')$ for all cells $\square \in \mathcal{C}$ and all free points $b \in B_{\square}$. Then, there exists a set of dual weights $y'(\cdot)$ such that $M^{\mathcal{C}}, y'(\cdot)$ is feasible, $y'(b) \leq \phi(P)$ for all $b \in B$, and $y'(b) = \phi(P)$ for all free points $b \in B$.*

We also recall an important property of extended matchings, which is discussed and shown in Chapter 6.

Lemma 6.5. *For a cell $\square$ of $\mathcal{H}$, suppose $\square'$ and $\square''$ denote its two children, and let $\mathcal{C}$ denote a partitioning containing $\square'$ and $\square''$. Let $\mathcal{C}' = \mathcal{C} \cup \{\square\} \setminus \{\square', \square''\}$. Given a $\mathcal{C}$-feasible extended matching $(M, B^{\mathcal{C}}), y(\cdot)$, let $B^{\mathcal{C}}_{\square} \subseteq B^{\mathcal{C}}$ denote the subset of boundary-matched points that are matched to the divider $\Gamma_{\square}$ of $\square$. Then, the $\mathcal{C}'$-extended matching $(M, B^{\mathcal{C}} \setminus B^{\mathcal{C}}_{\square}), y(\cdot)$ is also $\mathcal{C}'$-feasible.*

Based on the property of extended matchings in Lemma 6.5, our algorithm iteratively computes a minimum-cost $\mathcal{C}$-extended $(n-k)$-matching, erases a divider to merge two sibling cells, and iteratively computes minimum net-cost extended augmenting paths to retrieve a minimum-cost extended $(n-k)$-matching. We describe the details in the next section.

7.2 Algorithm

In this section, we describe an algorithm that, given a sequence σ of n requests in 2-dimensions, computes the optimal solution to the k-SP problem in $\tilde{O}(n^{1.8}\Phi(n)\log(n\Delta))$

time.

Initialize $\mathcal{C}$ to the leaf cells of $\mathcal{H}$. Let $M^{\mathcal{C}} = (M, B^{\mathcal{C}})$ be the extended matching maintained by the algorithm and initialized to $M = \emptyset$ and $B^{\mathcal{C}} = \emptyset$. For each point $v \in A \cup B$, let $y(v)$ denote its dual weight initialized to $y(v) = 0$. Let B_F, initialized to B, be the free points of B with respect to $M^{\mathcal{C}}$. For each cell $\square$ that contains at least one free point $b \in B_F$, let $P_{\square}$ denote the minimum net-cost augmenting path inside $\square$. Initially, since $\square$ is a leaf of $\mathcal{H}$, it contains only one point $b \in B_F$, and therefore, $P_{\square}$ is this point with a net-cost equal to $\mathrm{c}(b, \mathcal{C})$. Our algorithm maintains a priority queue PRIORITYQUEUE storing every leaf cell $\square \in \mathcal{C}$ with at least one free vertex with a key of $\phi(P_{\square})$. At any time during the execution of our algorithm, let $\square_{\min}$ be the cell with the smallest key in PRIORITYQUEUE and let λ be the key of $\square_{\min}$. Execute the following steps until PRIORITYQUEUE becomes empty:

- While $|B_F| > k$,
  - *Search step:* Execute the EXTENDEDHUNGARIANSEARCH procedure, which extracts the cell $\square$ with the minimum key of λ from PRIORITYQUEUE, augments the matching $M^{\mathcal{C}}$ along $P_{\square}$, and updates the key of $\square$ in PRIORITYQUEUE (See Section 7.2.1 for details).
- *Merge step:* If $\mathcal{C} = \{\square^*\}$, remove $\square^*$ from PRIORITYQUEUE and return the matching M of $M^{\mathcal{C}}$. Otherwise, pick a cell $\square' \in \mathcal{C}$ with the smallest perimeter and let $\square$ and $\square''$ be the parent and sibling of $\square'$ in $\mathcal{H}$, respectively. Erase the divider of $\square$, i.e., set $\mathcal{C} = \mathcal{C} \setminus \{\square', \square''\} \cup \{\square\}$. We execute a MERGE procedure that updates the matching $M^{\mathcal{C}}$ inside the cell $\square$ so that $\phi(P_{\square})$ is at least λ (See Section 7.2.2 for details). At this point, the size of the updated extended matching $M^{\mathcal{C}}$ may not be $(n - k)$, i.e., there may be more than k free points.

Invariants. For each cell $\square \in \mathcal{C}$, let $y_\square = \max_{b \in B_\square} y(b)$. During the execution of our algorithm,

(I1) the extended matching $M^{\mathcal{C}}, y(\cdot)$ is feasible,

(I2) For each cell $\square \in \mathcal{C}$, $y_\square \leq \lambda$ and for all free point $b \in B_\square$, $y(b) = y_\square$, and,

(I3) The λ-value is non-decreasing. Furthermore, after each step of the algorithm, λ denotes the smallest net-cost of all augmenting paths with respect to $M^{\mathcal{C}}$.

7.2.1 ExtendedHungarianSearch Procedure

Given a feasible extended matching $M^{\mathcal{C}}, y(\cdot)$ and a cell $\square \in \mathcal{C}$, the extended Hungarian search procedure computes the minimum net-cost augmenting path $P_\square$ and augments $M^{\mathcal{C}}$ along $P_\square$. It then computes the new minimum net-cost augmenting path and updates the key of $\square$ in PriorityQueue to be its net-cost. This procedure is similar to the classical Hungarian search procedure executed on $G_\square$ and is mildly modified to include the augmenting paths that end at the boundary of $\square$. Details of the procedure are as follows.

1. *Update duals:* With s as the source vertex, use Dijkstra's algorithm on $G_\square$ to compute the shortest path P_v from s to each $v \in A_\square \cup B_\square$, and let κ_v be the cost of P_v. Let

$$\kappa = \min\{\min_{a \in A_\square^F} \kappa_a, \min_{b \in B_\square} \kappa_b + s(b)\}. \tag{7.11}$$

   For any $v \in A_\square \cup B_\square$, if $\kappa_v < \kappa$, set $y(v) \leftarrow y(v) + \kappa - \kappa_v$.

2. *Augment:* Let $u \in A_\square^F \cup B_\square$ be the point realizing the minimum distance in Equation (7.11). Let P be the augmenting path obtained by removing s. Augment $M^{\mathcal{C}}$ along P.

3. *Update key:* If $B_\square$ has no free points, then remove $\square$ from PRIORITYQUEUE. Otherwise,

   (a) Recompute the residual graph $\mathcal{G}_\square$ with respect to the updated matching,

   (b) With s as the source, execute Dijkstra's shortest path algorithm on $G_\square$. For each $v \in A_\square \cup B_\square$, let κ_v be the distance from s to v.

   (c) Update the key of $\square$ in PRIORITYQUEUE to

$$\kappa_\square = \min\{\min_{a \in A_\square^F} \kappa_a, \min_{b \in B_\square} \kappa_b + s(b)\}.$$

Lemma 7.9 establishes the properties of the extended Hungarian search procedure.

Lemma 7.9. *After the execution of the extended Hungarian search procedure for a cell $\square$, the extended matching $M^{\mathcal{C}}, y(\cdot)$ remains feasible, $y(v) \leq \lambda$ for all points $v \in A_\square \cup B_\square$, and $y(b_f) = \lambda$ for all free points $b_f \in B_\square$. Furthermore, the path P computed by the procedure is a minimum net-cost augmenting path inside $\square$. After augmenting along P, the updated key for $\square$ is the smallest net-cost of all augmenting paths inside $\square$ and is at least λ.*

7.2.2 MERGE Procedure

Given a feasible extended matching $M^{\mathcal{C}} = (M, B^{\mathcal{C}}), y(\cdot)$, any cell $\square$ of $\mathcal{H}$ where both of its children $\square'$ and $\square''$ are in $\mathcal{C}$, and a value λ, the MERGE procedure uses the algorithm in Lemma 7.7 to update the dual weights $y(\cdot)$ inside $\square'$ (resp. $\square''$) so that $M^{\mathcal{C}}, y(\cdot)$ remains feasible, the dual weights of all points in $\square'$ (resp. $\square''$) are at most λ, and the dual weight of all free points $b'_f \in B_{\square'}$ (resp. $b''_f \in B_{\square''}$) is $y(b'_f) = \lambda$ (resp. $y(b''_f) = \lambda$). The procedure then updates $\mathcal{C} \leftarrow \mathcal{C} \cup \{\square\} \setminus \{\square', \square''\}$ and makes the points matched to the divider $\Gamma_\square$ free. While there exists a free point $b \in B_\square$ with $y(b) < \lambda$,

1. With s as the source vertex, use Dijkstra's algorithm on $G_\square$ to compute the shortest

path P_v from s to each $v \in A_\square \cup B_\square$, and let κ_v be the cost of P_v. Let

$$\kappa = \min\{\min_{a \in A^F_\square} \kappa_a, \min_{b \in B_\square} \kappa_b + s(b), \min_{b \in B_\square} \kappa_b + \lambda - y(b)\}. \tag{7.12}$$

2. Let $u \in A^F_\square \cup B_\square$ be the point realizing the minimum value in Equation (7.12). Let P be the path obtained by removing s from the path P_u.

   (a) If $u \in B_\square$ and $\kappa = \kappa_u + \lambda - y(u)$ (i.e., κ is determined by the third element in the RHS of Equation (7.12)), then P is a path from a free point $b \in B_\square$ to the matched point u. For each $v \in A_\square \cup B_\square$, if $\kappa_v < \kappa$, update its dual weight to $y(v) \leftarrow y(v) + \kappa - \kappa_v$, which makes P admissible. Set $M \leftarrow M \oplus P$. Note that u is now a free point with $y(u) = \lambda$.

   (b) Otherwise, P is an augmenting path. For each $v \in A_\square \cup B_\square$, if $\kappa_v < \kappa$, update its dual weight to $y(v) \leftarrow y(v) + \kappa - \kappa_v$, which makes P admissible. Augment M along P.

At the end of this execution, all free points of B in $\square$ have a dual weight of λ. Finally, we update the key for $\square$ by executing steps 3(a)–(c) from the extended Hungarian search step.

Lemma 7.10. *After the execution of the* MERGE *procedure on a cell $\square$, the updated extended matching $M^{\mathcal{C}}, y(\cdot)$ is feasible, the dual weights $y(v)$ for every $v \in A_\square \cup B_\square$ is at most λ, and $y(b_f) = \lambda$ for all free points $b_f \in B_\square$. Furthermore, the updated key for $\square$ is the smallest net-cost of all augmenting paths within $\square$ and is at least λ.*

7.3 Analysis

We begin by showing in Section 7.3.1 that the three invariants (I1)–(I3) hold during the execution of our algorithm and use them to show the correctness of our algorithm. We then show in Section 7.3.2 that the running time of our algorithm is $\tilde{O}(n^{9/5}\Phi(n)\log n\Delta)$.

7.3.1 Correctness Analysis

Our algorithm initializes $M^{\mathcal{C}}$ with a feasible $\mathcal{C}$-extended matching and sets all dual weights and λ to 0. Therefore, (I1) and (I2) hold at the start of the algorithm. The EXTENDED-HUNGARIANSEARCH procedure (Lemma 7.9) as well as the MERGE procedure (Lemma 7.10) do not violate the feasibility of the extended matching and therefore (I1) holds during the execution of the algorithm. The EXTENDEDHUNGARIANSEARCH (Lemma 7.9) and the MERGE (Lemma 7.10) procedures keep the dual weight of every point inside $\square$ at or below λ, while ensuring that the dual weight of free points inside $\square$ is λ; hence, (I2) holds. Finally, both the EXTENDEDHUNGARIANSEARCH (Lemma 7.9) and the MERGE (Lemma 7.10) procedures update the key of $\square$ to the smallest net-cost of all augmenting paths inside $\square$ and do not decrease the key of any cell. From this observation, (I3) follows in a straightforward way.

From (I1) and (I2), our algorithm maintains a feasible $\mathcal{C}$-extended matching where, for each cell $\square \in \mathcal{C}$, $y_{\square} \leq \lambda$. From (I3), λ is equal to the minimum net-cost of all augmenting paths with respect to $M^{\mathcal{C}}$; combining this with Lemma 7.6(b), we conclude that $M^{\mathcal{C}}$ is a minimum-cost $\mathcal{C}$-extended matching. Upon termination, the algorithm returns a minimum-cost $\mathcal{C}^*$-extended $(n-k)$-matching for $\mathcal{C}^* = \{\square^*\}$, which from Lemma 7.2 has no boundary-matched points. Therefore, this matching is also a minimum-cost $(n-k)$-matching, as desired.

7.3.2 Efficiency Analysis

Both the merge step and the extended Hungarian search step require an execution of Dijkstra's shortest path algorithm on $G_\square$. In the full version of our paper, we show that this execution takes $\tilde{O}(n_\square \Phi(n_\square))$ time. Using this, we analyze the execution time.

We begin by establishing a bound on the execution time of the merge step, which combines $\square'$ and $\square''$ into a single cell $\square$. At the start of this step, the dual weight of all free points in $\square'$ and $\square''$ are raised to λ (from Lemma 7.7). As a result, once the divider is removed, the only free points with dual weights below λ are those that are matched to the divider $\Gamma_\square$. The following lemma bounds the number of points matched to $\Gamma_\square$ by $O(n^{4/5})$.

Lemma 7.11. *For a cell $\square$ of $\mathcal{H}$, suppose $\square'$ and $\square''$ denote its two children, and let $\mathcal{C}$ denote a partitioning containing $\square'$ and $\square''$. Let $\mathcal{C}' = \mathcal{C} \cup \{\square\} \setminus \{\square', \square''\}$. Given a feasible $\mathcal{C}$-extended matching $(M, B^{\mathcal{C}}), y(\cdot)$, let $B^{\mathcal{C}}_\square \subseteq B^{\mathcal{C}}$ denote the subset of boundary-matched points that are matched to the divider $\Gamma_\square$ of $\square$. Then, $|B^{\mathcal{C}}_\square| = O(n^{4/5})$.*

Proof. Let M denote the matching of the extended matching $M^{\mathcal{C}}$, and let M' denote the matching constructed in Lemma 7.1 inside $\square$. By Lemma 7.3, any point $b \in B_\square$ that is matched in M is matched to a point $a \in A_\square$ inside $\square$. Let $M_\square$ denote the subset of the matching edges of M that lie inside $\square$. Define $F(M_\square)$ (resp. $F(M')$) as the set of unmatched points of $B_\square$ in the matching $M_\square$ (resp. M'). Note that $B^{\mathcal{C}}_\square \subseteq F(M)$. Any simple path P in the symmetric difference $M_\square \oplus M'$ from a free point $b \in B^{\mathcal{C}}_\square$ is called a (standard) alternating path if P ends at a point $b \in F(M')$ and a (standard) augmenting path if it ends at an unmatched point $a \in A_\square$ with respect to $M_\square$. Let $\mathcal{P}_{\text{aug}}$ (resp. $\mathcal{P}_{\text{alt}}$) denote the set of augmenting paths (resp. alternating paths) with an endpoint in $B^{\mathcal{C}}_\square$ in the symmetric difference $M_\square \oplus M'$. Note that $|B^{\mathcal{C}}_\square| = |\mathcal{P}_{\text{aug}}| + |\mathcal{P}_{\text{alt}}|$. For $\mathcal{P}_{\text{alt}}$, since each alternating path in the symmetric difference has one free endpoint in $B^{\mathcal{C}}_\square$ and the other free endpoint in $F(M')$,

$|\mathcal{P}_{\text{alt}}| \leq |F(M')| = O(n^{4/5})$. Next, we show that $|\mathcal{P}_{\text{aug}}| = O(n^{4/5})$. Using the definition of net-cost of an augmenting path,

$$\sum_{P \in \mathcal{P}_{\text{aug}}} \phi(P) = \sum_{P \in \mathcal{P}_{\text{aug}}} \left(\sum_{(a,b) \in P \cap M'} \mathrm{c}(a,b) - \sum_{(a,b) \in P \cap M_\square} \mathrm{c}(a,b) \right)$$
$$\leq \sum_{P \in \mathcal{P}_{\text{aug}}} \left(\sum_{(a,b) \in P \cap M'} \mathrm{c}(a,b) \right) \leq w(M'). \tag{7.13}$$

For each path $P \in \mathcal{P}_{\text{aug}}$, let $b_P \in B_\square$ and $a_P \in A_\square$ denote the two end-points of P. Define $B_{\text{aug}} := \{b_P : P \in \mathcal{P}_{\text{aug}}\}$. Using Corollary 7.5 and Equation (7.13),

$$\sum_{b \in B_{\text{aug}}} y(b) = \sum_{P \in \mathcal{P}_{\text{aug}}} y(b_P) \leq \sum_{P \in \mathcal{P}_{\text{aug}}} \phi(P) \leq w(M'). \tag{7.14}$$

Define $\alpha := \ell_\square n^{-1/5}$. Each free point $b \in B_{\text{aug}}$ is called a *far* (resp. *close*) point if the distance of b to the divider of $\square$ is more than (resp. at most) α. Let $B_{\text{aug}}^{\text{far}}$ (resp. $B_{\text{aug}}^{\text{close}}$) denote the set of all far (resp. close) points of B_{aug}. By Lemma 6.1,

$$|B_{\text{aug}}^{\text{close}}| = O(n_\square n^{-1/5}) = O(n^{4/5}). \tag{7.15}$$

For each far point $b \in B_{\text{aug}}^{\text{far}}$, since b is mapped to the divider $\Gamma_\square$, $y(b) = \mathrm{c}(b, \Gamma_\square) \geq \alpha$; therefore,

$$\sum_{b \in B_{\text{aug}}} y(b) \geq \sum_{b \in B_{\text{aug}}^{\text{far}}} y(b) \geq \alpha \times |B_{\text{aug}}^{\text{far}}|. \tag{7.16}$$

Combining Equations (7.14) and (7.16),

$$|B_{\text{aug}}^{\text{far}}| \leq \frac{\sum_{b \in B_{\text{aug}}} y(b)}{\alpha} \leq \frac{w(M')}{\alpha} = O(n^{4/5}). \tag{7.17}$$

Combining with Equation (7.15),

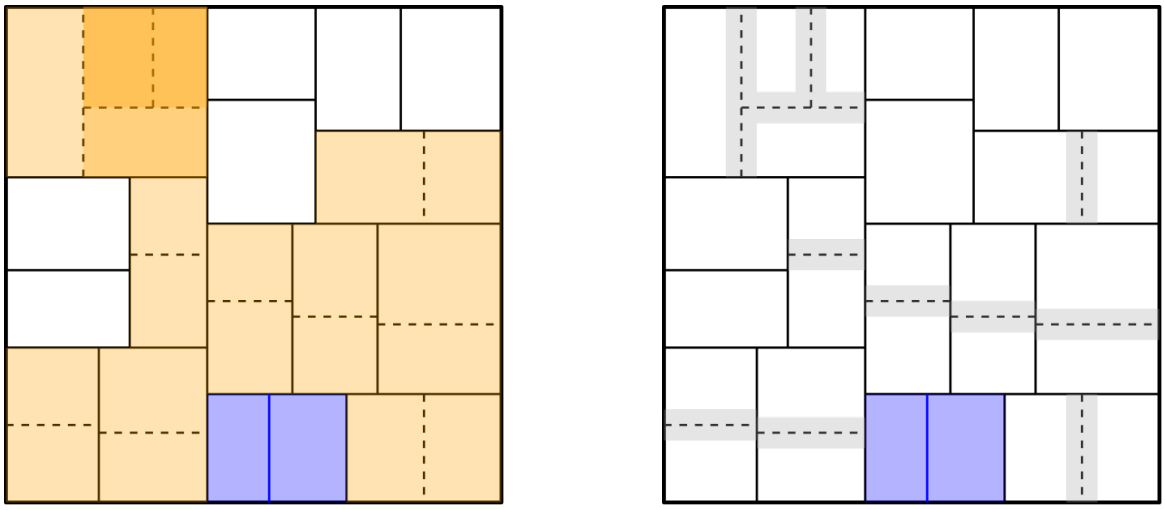
$$|B_{\square}^{\mathcal{C}}| \leq |B_{\text{aug}}| + |B_{\text{alt}}| \leq |B_{\text{aug}}^{\text{close}}| + |B_{\text{aug}}^{\text{far}}| + |B_{\text{alt}}| = O(n^{4/5}).$$

□

Therefore, the while-loop in the Merge procedure executes only $O(n^{4/5})$ times. Since each iteration takes $\tilde{O}(n_{\square}\Phi(n_{\square}))$ time, the total execution time of a single execution of the Merge procedure is $\tilde{O}(n^{4/5}n_{\square}\Phi(n_{\square}))$. Given that each point is in only $O(\log n\Delta)$ many cells of $\mathcal{H}$, the total time taken by the merge step across all cells of $\mathcal{H}$ is $\tilde{O}(n^{9/5}\Phi(n)\log n\Delta)$.

Similarly, if the execution time for the ExtendedHungarianSearch procedure within a single cell $\square$ is bounded by $O(n^{4/5}n_{\square}\Phi(n_{\square}))$, then the cumulative execution time of the ExtendedHungarianSearch procedure across all cells – and consequently, the total runtime of the entire algorithm – can be bounded by $\tilde{O}(n^{9/5}\Phi(n)\log n\Delta)$. In the remainder of our analysis, we establish a bound of $O(n^{4/5}n_{\square}\Phi(n_{\square}))$ for the time taken by the ExtendedHungarianSearch procedure within a single cell $\square$. Recall that the algorithm selects a cell $\square$ containing the minimum net-cost augmenting path from the priority queue PriorityQueue and executes the ExtendedHungarianSearch procedure on $\square$, requiring $\tilde{O}(n_{\square}\Phi(n_{\square}))$ time. To analyze the total execution time of the procedure, we show that any cell $\square$ can be selected by the algorithm at most $O(n^{4/5})$ times. In particular, we categorize the selection of $\square$ as a *low-net-cost* case if $\lambda \leq \ell_{\square}n^{-1/5}$ and a *high-net-cost* case otherwise. First, we show that in the high-net-cost case ($\lambda > \ell_{\square}n^{-1/5}$), the number of free points remaining in $\square$ is $O(n^{4/5})$, and as a result, the total number of high-net-cost selections of $\square$ is $O(n^{4/5})$.

Lemma 7.12. *Given a feasible extended matching $M^{\mathcal{C}}, y(\cdot)$ and any cell $\square \in \mathcal{C}$, if the net-cost of the minimum net-cost augmenting path inside $\square$ is greater than $\ell_{\square}n^{-1/5}$, then the number of free points of $M^{\mathcal{C}}$ is $O(n^{4/5})$.*



(a) (b)

Figure 7.4: (a) The set $\mathcal{C}_\square$ (the cells shaded in orange with dashed lines as their divider) for a cell $\square$ (shaded in blue), and (b) any boundary-matched point in the gray area might cause a low-net-cost execution of the ExtendedHungarianSearch procedure on $\square$.

Proof. Let M be the matching corresponding to the extended matching $M^{\mathcal{C}}$ and let M' be the matching defined in Lemma 7.1. Every free point of $M^{\mathcal{C}}$ participates in an augmenting or an alternating path in $M \oplus M'$. The number of alternating paths in the symmetric difference cannot exceed the number of free points of M', which is $O(n^{4/5})$. Furthermore, the combined net-cost of these augmenting paths is at most $w(M') = O(\ell_\square n^{3/5})$, and each one has a net-cost at least $\ell_\square n^{-1/5}$. Thus, there are $O(n^{4/5})$ augmenting paths in the symmetric difference. $\square$

Next, we bound the number of low-net-cost selections of $\square$. Define $\mathcal{C}_\square$ as the set of all cells $\square' \in \mathcal{H}$ that are processed by the Merge procedure while $\square \in \mathcal{C}$ and $\lambda \leq \ell_\square n^{-1/5}$ (Figure 7.4). Our algorithm always picks the smallest perimeter cells to merge, and as a result, we obtain the following lemma.

Lemma 7.13. *For any non-leaf cell $\square$ in $\mathcal{H}$ and any cell $\square' \in \mathcal{C}_\square$, $\frac{1}{3}\ell_{\square'} \leq \ell_\square \leq \frac{9}{4}\ell_{\square'}$.*

For any cell $\square' \in \mathcal{C}_\square$, the merge step erases the divider $\Gamma_{\square'}$ and makes the points that are

matched to $\Gamma_{\square'}$ free. Each of these new free points might cause $\square$ to be selected by the algorithm. Therefore, to bound the number of low-net-cost selections of $\square$, we show that the total number of points that are matched to the dividers of the cells in $\mathcal{C}_\square$ is at most $O(n^{4/5})$. For any cell $\square' \in \mathcal{C}_\square$, let $\mathcal{B}_{\square'}$ denote the set of points of $B^{\mathcal{C}}$ that are matched to the divider $\Gamma_{\square'}$ when our algorithm starts the merge step on $\square'$. Thus, we have to bound $\sum_{\square' \in \mathcal{C}_\square} |\mathcal{B}_{\square'}|$ by $O(n^{4/5})$. By invariant (I2), for each point $b \in \mathcal{B}_{\square'}$, $y(b) \leq \lambda \leq \ell_\square n^{-1/5}$. Furthermore, by Condition (7.5), $y(b) = \mathrm{c}(b, \square') = \mathrm{c}(b, \Gamma_{\square'})$. Therefore, the point b is within a distance $\ell_\square n^{-1/5}$ from the divider $\Gamma_{\square'}$. From Lemma 7.13, for each cell $\square' \in \mathcal{C}_\square$ and each point $b \in \mathcal{B}_{\square'}$, the point b is within a distance $\ell_\square n^{-1/5} < \ell_{\square'} \lambda$ from the divider $\Gamma_{\square'}$, and from the construction of $\mathcal{H}$ (Lemma 6.1), $|\mathcal{B}_{\square'}| = O(n_{\square'} n^{-1/5})$. Furthermore, from Lemma 7.13 and the construction of $\mathcal{H}$, each point $b \in B$ can lie inside a constant number of cells in $\mathcal{C}_\square$; hence, $\sum_{\square' \in \mathcal{C}_\square} |\mathcal{B}_{\square'}| = O(\sum_{\square' \in \mathcal{C}_\square} n_{\square'} n^{-1/5}) = O(n^{4/5})$. Therefore, the total number of low-net-cost selections of $\square$ by the extended Hungarian search procedure is $O(n^{4/5})$.

7.4 Fast Search Procedures for the Geometric k-SP Problem

Given an instance of the geometric k-SP, in this section, we show that the search for the minimum net-cost extended augmenting path using a Hungarian search style procedure (as in the Merge and ExtendedHungarianSearch procedures of our algorithm) can be done in $\tilde{O}(n\Phi(n))$ time, where $\Phi(n)$ is the query/update time of an existing DWBCP-DS.

Given a feasible matching $M, y(\cdot)$, the Hungarian search (which executes a Dijkstra's shortest path procedure) can be efficiently implemented on a complete bipartite graph using only $\tilde{O}(n)$ queries to a DWBCP-DS. This search procedure grows a tree $\mathcal{T}$ by finding the cheapest cut

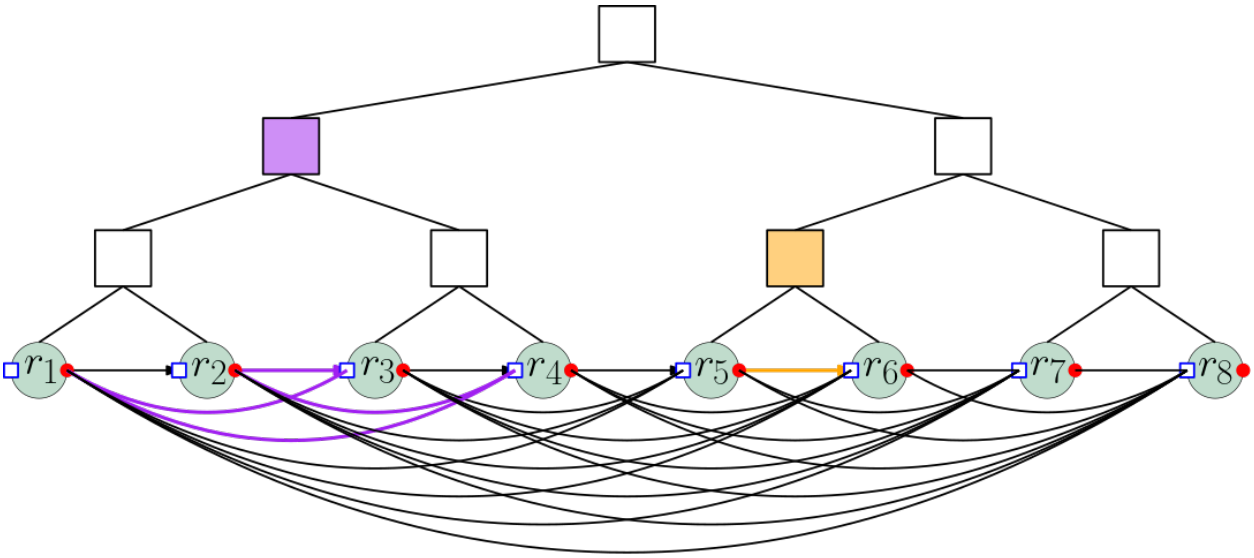


Figure 7.5: The balanced binary search tree T used in the construction of our DWBCP-DS. The purple (resp. orange) edges in the graph are stored at the purple (resp. orange) node of T.

edge under a weighted distance. See [4, 146, 152], where the Hungarian search procedure that uses $O(n)$ queries to a DWBCP-DShas been described.

In our case, the graph G_σ is not a complete graph. However, we can easily decompose the graph into a set of $O(n)$ complete bipartite graphs as follows. Build a balanced binary search tree T where the indices of requests $\{1, \ldots, n\}$ form the leaves. Let, for any node v, $L(v)$ denote the indices at the leaves of the left subtree of v and $R(v)$ denote the indices at the leaves of the right subtree of v. We partition the edges as follows: At each node v in T, we store all edges that go from the exit gates of requests with indices in $L(v)$ to the entry gates of the requests with indices in $R(V)$. See Figure 7.5 for an example. Thus, at each node, we simply store the complete bipartite graph between points in $L(v)$ and $R(v)$. It is easy to see that any request r_j participates in $O(\log n)$ cliques.

While executing a Hungarian Search, we simply build a DWBCP-DSat each node v of this tree. This data structure stores nodes of $L(v)$ that are currently in $\mathcal{T}$ and nodes of $R(v)$

that are not currently in $\mathcal{T}$. The weighted bichromatic closest pair for each of these cliques is stored in the global priority queue. The query to this data structure will simply return the pair at the top of this priority queue. Insertion (resp. deletion) of a vertex v will trigger insertion (resp. deletion) in the $O(\log n)$ cliques that v participates in. For each of these cliques, we update their representative pair in the global priority queue. Thus, all insertions, deletions, and queries can be done in $\tilde{O}(\Phi(n))$ time. We obtain a fast Hungarian search by using this data structure in place of the standard **DWBCP-DS**as used in [146].

Chapter 8

Conclusions

This thesis presented new algorithmic advances in optimal transport (OT), with a focus on the semi-discrete setting and bipartite matchings. Our goal has been to design efficient, scalable, and theoretically grounded methods for computing OT distances and transport plans in geometric settings.

In Chapter 3, we developed an efficient cost-scaling algorithm for computing an ε-close semi-discrete transport plan. The running time of our algorithm has a dependence of $\log(1/\varepsilon)$ on the error parameter ε. In Chapter 4, we designed a novel combinatorial framework for the semi-discrete optimal transport problem and introduced the concepts of augmenting paths and residual graphs to continuous spaces. We used this framework to develop an efficient algorithm to compute an ε-close semi-discrete OT. Both our algorithms have logarithmic dependence on $1/\varepsilon$ and an exponential dependence on the dimension d. Designing a combinatorial algorithm with a running time of $\text{poly}(n, d, 1/\varepsilon)$ is an important open question.

In Chapter 5, we characterized the optimal and additive approximate solutions to the robust OT problem in the semi-discrete setting. We extended our combinatorial framework and designed an efficient ε-close semi-discrete λ-robust OT problem. Our algorithm can shed light on designing algorithms for different robust variants of OT, such as α-optimal partial transport [37] and the robust partial p-Wasserstein distance [135].

In Chapter 6, we proposed a novel algorithm for computing an exact bipartite matching

problem between two point sets. Our algorithm achieves speedups by localizing the search for augmenting paths inside the cells of a partitioning of the space. We showed that, when the input points A and B are random subsets of a larger point set U, our algorithm computes a minimum-cost bipartite matching using sub-quadratic queries to a dynamic weighted bichromatic closest pair data structure. Notably, our algorithm is a weakly polynomial algorithm as its running time is dependent on the spread of the point set. Presenting a simple algorithm whose running time has no dependence on the spread would be an interesting future direction.

Finally, in Chapter 7, we extended our techniques to the partial matching problem and showed its application to the k-SP problem. We developed an efficient algorithm that computes a minimum-cost partial matching and showed that for the bipartite matching instances that are generated by the k-SP problem, the optimal solution can be computed by our algorithm using a sub-quadratic number of queries to a dynamic weighted bichromatic closest pair data structure.

The k-SP problem is a variant of the offline version of the well-known k-server problem where the initial locations of the servers are not determined. The reduction of the k-SP problem to the bipartite matching problem, therefore, proposes the k-server problem (online version) as a mild variant of the online metric matching problem. There has been extensive work on designing algorithms for the online k-server problem [32, 42, 95, 96, 101, 108, 134, 140]. By leveraging this reduction, the implementation of the work function algorithm, such as the scalable approach by Raghvendra and Sowle [134] can be simplified and improved. Furthermore, the best-known algorithm for online metric matching, the RM algorithm [118, 131, 132], can be adapted to solve the online k-server problem. Can we derive better upper bounds for the competitive ratio of the RM algorithm when applied to the k-server problem? Such results would mark progress toward proving the deterministic k-server conjecture.

Glossary

α-OPT plan: Given two distributions μ and ν, an α-OPT plan is an α-partial transport plan with a minimum cost. 4

α-partial transport plan: A transport plan τ that transports α mass, i.e., $\mathcal{M}(\tau) = \alpha$. 4

β-approximate transport plan: Any transport plan τ between μ and ν whose cost is at most β times the optimal transport cost. 5

δ-adjusted slack: The slackness defined on condition (3.6), rounded to an integer multiple of δ; more precisely, $s_\delta(\varrho, b) := \left\lfloor \frac{c(r_\varrho, b)+\delta-y(b)+y_\delta(r_\varrho)}{\delta} \right\rfloor \delta$. 50

δ-expanded Voronoi cell: For a point set B, a set of weights $y(\cdot)$ for B, and any point $b \in B$, define the weight function y_b^δ for B that assigns $y_b^\delta(b) = y(b) + \delta$ to b and $y_b^\delta(b') = y(b')$ for each point $b' \in B, b' \neq b$. The δ-expanded Voronoi cell of b, denoted by $\mathrm{Vor}_y^\delta(b)$, is the Voronoi cell b with respect to weights $y_b^\delta(\cdot)$. 31

δ-expanded restricted Voronoi cell: For a point set B, a set of weights $y(\cdot)$ for B, and any point $b \in B$, the δ-expanded restricted Voronoi cell of b, denoted by $\mathrm{ResVor}_y^\delta(b)$, is the intersection of the δ-expanded Voronoi cell of b with a ball of radius $y(b) + \delta$ centered at b. 92

δ-feasible: 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 for any pair $(a, b) \in A \times B$ with $\tau(a, b) > 0$, the point a lies inside the δ-expanded Voronoi cell $\mathrm{Vor}_y^\delta(b)$. 31

δ-optimal transport plan: Any δ-feasible transport plan $\tau, t(\cdot)$ between μ and ν where τ is a complete transport plan. 31

δ-restricted-feasible: Any (possibly partial) transport plan τ between μ and ν along with a weight function $y(\cdot)$ for the points in B is δ-restricted-feasible if for any pair $(a, b) \in A \times B$ with $\tau(a, b) > 0$, the point a lies inside the δ-expanded restricted Voronoi cell $\mathrm{ResVor}_y^\delta(b)$. 92

δ-weighted nearest neighbor: Given a point set P along with a set of weights $w(\cdot)$ for P, for any point $a \in \mathbb{R}^d$, a δ-weighted nearest neighbor of a in P is any point $p \in P$ such that $\mathrm{c}(a, p) - w(p) \leq \min_{p' \in P}(\mathrm{c}(a, p') - w(p')) + \delta$. 31

λ-ROT plan: Given two distributions μ and ν, a λ-ROT plan is a transport plan with a minimum λ-robust cost. 5

λ-ROT problem: Abbreviation of λ-robust OT problem. 4

λ-robust OT problem: Given two distributions μ and ν, the λ-robust OT problem aims to find a transport plan τ that minimizes the objective function $w(\tau) + \lambda(1 - \mathcal{M}(\tau))$. 4

λ-robust cost of a transport plan: the λ-robust cost of a transport plan τ is its cost in addition to the cost of burning the untransported mass, i.e., $w(\tau) + \lambda(1 - \mathcal{M}(\tau))$. 5

$\mathcal{C}$-extended t-matching: An extended matching $M^{\mathcal{C}}$ of size t. 151

$\mathcal{C}$-feasible extended matching: A $\mathcal{C}$-extended matching $M^{\mathcal{C}} = (M, B^{\mathcal{C}})$ and a set of non-negative weights $y(\cdot)$ for $A \cup B$ is $\mathcal{C}$-feasible if Equations (6.2)–(6.6) are satisfied. 128

ε-close transport plan: Any transport plan τ between μ and ν whose cost is within an ε additive error from the optimal transport cost. 5

t-matching: For point sets A and B of size n and a parameter $t < n$, a t-matching is a matching of A and B of size t. 6

Admissible edge: An edge (a, b) with a zero slack, i.e., $s(a, b) = 0$. 13, 100

Admissible extended augmenting path: For a $\mathcal{C}$-feasible extended matching $M^{\mathcal{C}}, y(\cdot)$, an extended augmenting path P is admissible if all edges of P are admissible, and if P ends at a point $b' \in B$, the slack of the end-point b' is $s(b') = 0$. 130

Admissible path/cycle: An alternating path/cycle or an augmenting path is admissible if all edges along the path are admissible. 14, 66, 100

Admissible triple: For a 2δ-feasible transport plan $\hat{\tau}, y(\cdot)$, a triple (b_1, r, b_2) is admissible if $\hat{\tau}(r, b_2) > 0$ and $\mathrm{c}_y(r, b_1) < \mathrm{c}_y(r, b_2)$. 65

Algebraic surface of constant degree: An algebraic surface of constant degree refers to a surface in $\mathbb{R}^d$ that is defined as the zero set of a polynomial equation whose degree is fixed. 44

Alternating cycle: An alternating cycle is a simple cycle whose edges alternate between those that transport mass (backward edges) and those that still can transport more mass (forward edges). 63

Alternating path: In the context of matchings, an alternating path with respect to a matching M is a simple path whose edges alternate between matching and non-matching edges. In the more general case of discrete OT problem, an alternating path is a simple path whose edges alternate between those that transport mass (backward edges) and those that still can transport more mass (forward edges). 11, 63, 100

Arrangement: Given a set of cells $\mathcal{V}$, the arrangement $\mathcal{A}(\mathcal{V})$ of $\mathcal{V}$ is a decomposition of the space $\mathbb{R}^d$ induced by $\mathcal{V}$, where each region $\varphi \in \mathcal{A}(\mathcal{V})$ is the maximum region in the intersection of a subset of cells $V \subseteq \mathcal{V}$ and outside all other cells $\mathcal{V} \setminus V$. 43

Augment process: In the context of matching, augmenting a matching M along an augmenting path is done by simply setting $M \leftarrow M \oplus P$. In the more general case of the discrete OT problem, if $\mathrm{bc}(P)$ denotes the bottleneck capacity of P, then the augment process increases (resp. decreases) the amount of mass transported along the forward (resp. backward) edges by $\mathrm{bc}(P)$. 11, 63, 100

Augment process (in extended matchings): For a $\mathcal{C}$-extended matching $M^{\mathcal{C}} = (M, B^{\mathcal{C}})$ and an extended augmenting path P, we augment the extended matching $M^{\mathcal{C}}$ along P by updating its matching $M \leftarrow M \oplus P$, and if P ends at a point $b' \in B$, we match b' to the boundary and update $B^{\mathcal{C}}$ to include b'. 127

Augmenting path: An augmenting path with respect to a transport plan or a matching is an alternating path whose endpoints are free. 11, 63, 100

Balanced point: For any transport plan τ, any point $b \in B$ is balanced if $\int_A \tau(a, b)\, da = \nu(b)$ and any point $a \in A$ is balanced if $\sum_{b \in B} \tau(a, b) = \mu(a)$. 92

Balanced restricted Voronoi cell: Given a set of weights $y(\cdot)$ for B, the restricted Voronoi cell $\mathrm{ResVor}_y(b)$ of a point $b \in B$ is balanced if the amount of continuous mass inside it is equal to the discrete mass at b, i.e., $\mu(\mathrm{ResVor}_y(b)) = \nu(b)$. 86

Bipartite matching: For two point sets A and B, a bipartite matching M is a set of vertex-disjoin edges between A and B. 6

Bottleneck capacity of an augmenting path: The maximum amount of mass that can be pushed through an augmenting path or a consolidating path, which is computed as the minimum of the excess of the two free endpoints of P and the amount of mass transported along its backward edges. 63, 100

Boundary-matched points: In a $\mathcal{C}$-extended matching, the points of B that are matched to the boundaries of $\mathcal{C}$ is called boundary-matched. 125

Cap of a weight function: For a valid weight function $y(\cdot)$ for B, the cap of $y(\cdot)$ is the maximum weight in $y(\cdot)$, i.e., $\text{Cap}(y) := \max_{b \in B} y(b)$. 87

Cell: Any axis-parallel rectangle in the hierarchical partitioning of space. 121

Complementary slackness property: In a primal-dual linear/integer program, the complementary slackness property states that for any condition in the primal formulation and its corresponding dual variable in the dual formulation, in the optimal solutions, either the condition in the primal formulation is tight or the dual variable is zero. 12, 88

Complete transport plan: A transport plan τ between two probability distributions that transports all the mass of the distributions, i.e., $\mathcal{M}(\tau) = 1$. 4

Consolidating path: For a semi-discrete transport plan τ and the partitioning $\mathcal{X}_\delta$ based on the expansions of the restricted Voronoi cells, any alternating path $P = \langle r_0, b_1, r_1, \ldots, b_k, r_k \rangle$ in the residual graph is called a *consolidating path* if r_0 is a point with $\bar{y}(r_0) = 0$ and r_k is a deficit point with $\bar{y}(r_k) > 0$, i.e., P is a path from a point outside of all 2δ-expanded cells to a violating deficit point. 101

Constant complexity region: In computational geometry, a constant complexity region refers to a geometric region whose boundary can be described using a constant number of simple components, regardless of the size of the overall input. 26

Cover of a weight function: For a valid weight function $y(\cdot)$ for B, the cover of $y(\cdot)$ is the total amount of continuous mass inside the restricted Voronoi cells of the points in B, i.e., $\text{Cover}(y) := \sum_{b \in B} \mu(\text{ResVor}_y(b))$. 87

Deficit point: For any transport plan τ, any point $a \in A$ is deficit if $\sum_{b \in B} \tau(a,b) < \mu(a)$. 92

Deficit restricted Voronoi cell: Given a set of weights $y(\cdot)$ for B, the restricted Voronoi cell $\text{ResVor}_y(b)$ of a point $b \in B$ is deficit if the amount of continuous mass inside it is greater than the discrete mass at b, i.e., $\mu(\text{ResVor}_y(b)) > \nu(b)$. 87

Derived weight: Given a weight function $y(\cdot)$ for B, the derived weight of any point $a \in A$ is $\bar{y}(a) := \max\{0, \max_{b \in B}\{y(b) - \text{c}(a,b)\}\}$. 93

Divider of a cell: For any cell $\square$ of the hierarchical partitioning $\mathcal{H}$, the divider $\Gamma_\square$ of $\square$ is the vertical or a horizontal line used to divide $\square$ into its two children. 122

Dual disc: For point sets A and B with non-negative dual weights $y(\cdot)$, the dual disc of each point $v \in A \cup B$ is a disc of radius $y(v)$ centered at v. 38

Dual weight: The weight assigned to a point in the dual formulation of the linear/integer program of the optimal transport/matching problem. 12

Excess mass of a point: For any transport plan τ and any point $v \in A \cup B$, the excess of v is the amount of mass at v that is not transported by τ. 60

Extended augmenting path: For a $\mathcal{C}$-extended matching $M^{\mathcal{C}} = (M, B^{\mathcal{C}})$, an extended augmenting path is an alternating path (with respect to the matching M) that starts at a free point $b \in B$ and ends at a either (i) a free point $a \in A$, or (ii) a matched or boundary-matched point $b' \in B$. 127

Feasible matching: A matching M and a set of non-negative weights $y(\cdot)$ for $A \cup B$ is feasible if Equations (1.11) and (1.12) are satisfied. 13

Forest transport plan: A discrete transport plan τ such that the edges transporting a positive mass in τ do not create an undirected cycle. 62

Free point: For any transport plan τ, any point $v \in A \cup B$ whose mass is not completely transported by τ is called a free point. In the context of matching, a free point is simply a point that not adjacent to any matching edges. 10, 60

i.i.d samples: i.i.d stands for independent and identically distributed, meaning that each sample does not inflence the selection of the other samples and all samples are drawn from the same distribution. 23

Implicit representation of a transport plan: For a semi-discrete transport plan τ between μ and ν, a partitioning $\mathcal{X}_\delta$ of the support A of μ, and the set of representative points A_δ of the regions in $\mathcal{X}_\delta$, the implicit representation of τ is a transport plan $\hat{\tau}$ over $A_\delta \times B$ that assigns, for each pair $(\varphi, b) \in \mathcal{X}_\delta \times B$, $\hat{\tau}(r_\varphi, b) = \tau(\varphi, b)$. 98

Mass of a transport plan τ: The amount of mass transported by τ. 4

Matching: A matching between two point sets A and B is a subset of vertex-disjoint edges of A and B. 10

Matching edge: For a matching M between A and B, any edge $(a, b) \in A \times B$ is a matching edge if $(a, b) \in M$. 10

Minimum-cost $\mathcal{C}$-extended t-matching: An extended matching $M^{\mathcal{C}}$ of size t with the minimum cost. 125

Minimum-cost perfect $\mathcal{C}$-extended matching: An extended matching $M^{\mathcal{C}}$ of size n with the minimum cost. 125

Net-cost (in extended matchings): For a $\mathcal{C}$-extended matching $M^{\mathcal{C}} = (M, B^{\mathcal{C}})$ and an extended augmenting path P, the net-cost of P is the change in the cost of the extended matching $M^{\mathcal{C}}$ when $M^{\mathcal{C}}$ is augmented along P. More precisely, if P ends at a free point $a \in A$, then its net-cost is $\phi(P) := \sum_{(a,b)\in P\setminus M} \mathrm{c}(a,b) - \sum_{(a,b)\in P\cap M} \mathrm{c}(a,b)$, and if P ends at a point $b' \in B$, then the net-cost of P is $\phi(P) := \mathrm{c}(b', \mathcal{C}) + \sum_{(a,b)\in P\setminus M} \mathrm{c}(a,b) - \sum_{(a,b)\in P\cap M} \mathrm{c}(a,b)$. 153

Net-cost of an augmenting path: For any matching M and any augmenting path P, the net-cost of P is the increase in the cost of the matching when we augment M along P. 11

Non-matching edge: For a matching M between A and B, any edge $(a, b) \in A \times B$ is a non-matching edge if $(a, b) \in M$. 10

Optimal transport plan: Any complete transport plan between two distributions with a minimum cost among all complete transport plans between those distributions. 4

OT plan: An optimal transport plan. 4

Perfect matching: A perfect matching is a bipartite matching between point sets A and B that matches all points of A and B. 6

Representative point: An arbitrary point designated to represent the mass inside a region. 43

Residual capacity of a point: The amount of mass of the point that is not transported by the transport plan. 100

Residual graph: For a feasible matching $M, y(\cdot)$ between A and B, the residual graph $\mathcal{G}$ is a directed bipartite graph defined over the point set $A \cup B$ in addition to a source vertex

s. For any pair of points $(a, b) \in A \times B$, if (a, b) is a matching (resp. non-matching) edge in M, we add a directed edge from a to b (resp. b to a) with a weight $s(a, b)$ to the residual graph $\mathcal{G}$. Furthermore, we add a zero-weight edge from the source vertex s to all free points $b \in B_F$. 15

Restricted Voronoi cell: Given a point set P with weights $y(\cdot)$, the restricted Voronoi cell of each point $p \in P$ is the intersection of its Voronoi cell with a disc of radius $y(p)$ centered at p. 9

Semi-discrete α-OPT problem: Abbreviation of semi-discrete α-optimal partial transport problem. 4

Semi-discrete α-optimal partial transport problem: Given a continuous distribution μ and a discrete distribution ν, the semi-discrete α-optimal partial transport problem aims to find an α-partial transport plan with a minimum cost. 4

Semi-discrete OT problem: Given a continuous distribution μ and a discrete distribution ν, the semi-discrete OT problem aims to find an optimal transport plan between μ and ν. 4

Semi-discrete transport plan: For a continuous distribution μ defined over a bounded support A and a discrete distribution ν defined over point set B, a semi-discrete transport plan τ is a distribution $\tau\colon A \times B \to \mathbb{R}_{\geq 0}$ whose first and second marginals are dominated by μ and ν. 4

Sequence partitioning problem: For a sequence of requests $\sigma = \langle r_1, \ldots, r_n \rangle$, the k-sequence partitioning problem aims to partition the requests in σ into k subsequences such that the total distance between each consecutive pair of points in these subsequences is minimized. 24

Size of a matching: The number of matching edges in a matching. 10

Slack of a point: For an extended feasible matching $M^{\mathcal{C}} = (M, B^{\mathcal{C}}), y(\cdot)$, the slack of any point $b \in B$ is defined as $s(b) := \mathrm{c}(b, \mathcal{C}) - y(b)$: How far the feasibility constraint on the point b is from becoming an equality. 130

Slack of an edge: $s(a, b) := \mathrm{c}(a, b) - y(b) + y(a)$: How far the feasibility constraint on the edge (a, b) is from becoming an equality. 13

Spread: Given a set of points X, the spread of X is the ratio of the largest to smallest positive pairwise distances between the points in X, i.e., if $C_{\max}$ denotes the maximum pairwise distance of points in X and $C_{\min}$ denotes the smallest distance between all pairs of points that are not co-located, then the spread of X is $\Delta := \frac{C_{\max}}{C_{\min}}$. 34

Strong duality property: In the primal-dual linear programming formulation, the strong duality property states that the optimal solutions to the primal and dual formulations exists and the optimal values of their objective functions are equal to each other. 88

Surplus point: For any transport plan τ, any point $b \in B$ is surplus if $\int_A \tau(a, b)\, da < \nu(b)$. 92

Surplus restricted Voronoi cell: Given a set of weights $y(\cdot)$ for B, the restricted Voronoi cell $\mathrm{ResVor}_y(b)$ of a point $b \in B$ is surplus if the amount of continuous mass inside it is less than the discrete mass at b, i.e., $\mu(\mathrm{ResVor}_y(b)) < \nu(b)$. 87

Update process: For a transport plan τ and an alternating path P, we update τ along P by a mass of β by increasing (resp. decreasing) the amount of mass transported along the forward (resp. backward) edges by β. 100

Valid weight function: A weight function $y : B \to \mathbb{R}_{\geq 0}$ is called valid if all points $b \in B$ are either balanced or surplus and the surplus points have the maximum weight, i.e., for all surplus points $b \in B$, $y(b) = \max_{b' \in B} y(b')$.. 87

Violating point: Given a transport plan τ and weights $y(\cdot)$, any representative point r_φ in the set of representative points A_δ is violating if τ does not transport all the mass of φ and r_φ has a positive derived weights, i.e., $\bar{y}(r_\varphi) > 0$. In other words, r_φ lies inside the restricted Voronoi cell of a point $b \in B$ but its mass is not fully transported. 98

Weighted bisector: For two points b_1 and b_2 with weights w_1 and w_2, the weighted bisector of b_1 and b_2 is the locus of points with the same weighted distance to b_1 and b_2, i.e., $\{x \in \mathbb{R}^d \mid \mathrm{c}(x, b_1) - w_1 = \mathrm{c}(x, b_2) - w_2\}$. 46

Weighted nearest neighbor: Given a point set P along with a set of weights $w(\cdot)$ for P, for any point $a \in \mathbb{R}^d$, the weighted nearest neighbor of a in P is the point $p \in P$ with the smallest weighted distance to a, i.e., $p := \arg\min_{p' \in P} \mathrm{c}(a, p') - w(p')$. 31

List of Notations

A_F^M: The free points of A with respect to a matching M. 10

$A_\square$: The subset of points of A that lie inside $\square$. 121

A_δ: The set of representative points of all regions in the arrangement. 43, 59, 98

B_F^M: The free points of B with respect to a matching M. 10

$B_\square$: The subset of points of B that lie inside $\square$. 121

M: A matching. 10

Δ: The spread of point set. 34

$\bar{y}(a)$: The derived weight of a. 92

$\mathrm{Cap}(y)$: The cap of the weight function $y(\cdot)$. 87

$\square$: Any cell of the hierarchical partitioning. 121

$\square^*$: The root cell of the hierarchical partitioning. 121

$\mathrm{Cover}(y)$: The cover of the valid weight function $y(\cdot)$. 87

$C_{\max}$: The diameter of the space, i.e., the maximum pairwise distance between all points of A and B. 3

$\mathcal{X}_\delta$: A decomposition of the support A of the continuous distribution μ. 46, 59, 97

$\mathrm{c}(a, b)$: The ground distance between a and b. 3

$\Gamma_\square$: The divider of the cell $\square$. 122

$\ell_{\square}$: The larger of the length of width of the rectangle $\square$. 121

$\hat{\mu}_{\delta}$: The discrete distribution over the set of representative points A_{δ}. 43, 59

$\mathcal{M}(\tau)$: The mass of τ. 4

bc(P): The bottleneck capacity of an augmenting path P. 63, 100

ex(b): The excess of the point b. 60

rc(v): The residual capacity of a point v. 100

$\mathcal{A}(\mathcal{V})$: The arrangement of a set of cells $\mathcal{V}$. 43

$\mathcal{H}$: The hierarchical partitioning. 121

Q: The query time of an ORACLE that returns the amount of mass of a continuous distribution μ inside a constant complexity region. 26

$\phi(P)$: The net-cost of P. 11

$\pi_{w_1,w2}(b_1, b_2)$: The weighted bisector of b_1 and b_2 with weights w_1 and w_2. 46

r_{φ}: The representative point of a region φ. 43, 98

$\mathcal{G}$: The residual graph. 15

ResVor$_y(b)$: The restricted Voronoi cell of b with weights $y(\cdot)$ for B. 9

ResVor$^{\delta}_y(b)$: The δ-expanded restricted Voronoi cell of b with respect to weights $y(\cdot)$. 92

$w^{\mathbf{ROT}}_{\lambda}(\tau)$: The λ-robust cost of τ. 5

Vor$^{\delta}_b$: The δ-expanded Voronoi cell of b with respect a set of weights $y(\cdot)$ that is clear from the context. 31

$\mathrm{Vor}_y^\delta(b)$**:** The δ-expanded Voronoi cell of b with respect to weights $y(\cdot)$. 31

$\|a-b\|_p^q$**:** The ℓ_p^q norm of the points a and b. 3

$n_\square$**:** The number of points of $A \cup B$ that lie inside $\square$. 121

$s(a,b)$**:** The slack of (a,b): $s(a,b) := \mathrm{c}(a,b) - y(b) + y(a)$. 13

$s_\delta(\varrho,b)$**:** The δ-adjusted slack of the pair (ϱ,b). 50

DWBCP-DS: A dynamic weighted bichromatic closest pair data structure. 15

Bibliography

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

[2] Pankaj K. Agarwal and Micha Sharir. Efficient algorithms for geometric optimization. *ACM Computing Surveys*, 30(4):412–458, 1998.

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

[4] Pankaj K. Agarwal, Alon Efrat, and Micha Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. In *Proceedings of the 11th Annual Symposium on Computational Geometry*, pages 39–50, 1995.

[5] Pankaj K. Agarwal, Hsien-Chih Chang, Sharath Raghvendra, and Allen Xiao. Deterministic, near-linear ε-approximation algorithm for geometric bipartite matching. In *Proceedings of the 54th Annual ACM Symposium on Theory of Computing*, pages 1052–1065, 2022.

[6] Pankaj K. Agarwal, Sharath Raghvendra, Pouyan Shirzadian, and Rachita Sowle. An improved ε-approximation algorithm for geometric bipartite matching. In *18th Scandinavian Symposium and Workshops on Algorithm Theory*, pages 6–1, 2022.

[7] Pankaj K. Agarwal, Sharath Raghvendra, Pouyan Shirzadian, and Rachita Sowle. A higher precision algorithm for computing the 1-Wasserstein distance. In *Proceedings of the International Conference on Learning Representations*, 2023.

[8] Pankaj K Agarwal, Sharath Raghvendra, Pouyan Shirzadian, and Keegan Yao. A combinatorial algorithm for the semi-discrete optimal transport problem. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, pages 25857–25887, 2024.

[9] 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 *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 4514–4529, 2024.

[10] Pankaj K. Agarwal, Sharath Raghvendra, Pouyan Shirzadian, and Keegan Yao. Efficient approximation algorithm for computing Wasserstein barycenter under Euclidean metric. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 4809–4826. SIAM, 2025.

[11] Pankaj K. Agarwal, Sharath Raghvendra, Pouyan Shirzadian, and Keegan Yao. Efficient algorithms for robust and partial semi-discrete optimal transport. 2025.

[12] Miklós Ajtai, János Komlós, and Gábor Tusnády. On optimal matchings. *Combinatorica*, 4(4):259–264, 1984.

[13] Jason Altschuler, Jonathan Weed, and Philippe Rigollet. Near-linear time approximation algorithms for optimal transport via Sinkhorn iteration. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1961–1971, 2017.

[14] Jason Altschuler, Francis Bach, Alessandro Rudi, and Jonathan Niles-Weed. Massively scalable Sinkhorn distances via the Nyström method. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 4427–4437, 2019.

[15] 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.

[16] Luca Ambrogioni, Umut Güçlü, Yagmur Güçlütürk, Max Hinne, Eric Maris, and Marcel AJ van Gerven. Wasserstein variational inference. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 2478–2487, 2018.

[17] 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.

[18] Luigi Ambrosio, Nicola Gigli, and Giuseppe Savaré. *Gradient flows: in metric spaces and in the space of probability measures*. Springer Science & Business Media, 2008.

[19] 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 *Proceedings of the International Conference on Learning Representations*, 2019.

[20] Alexandr Andoni, Piotr Indyk, and Robert Krauthgamer. Earth mover distance over high-dimensional spaces. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, volume 8, pages 343–352, 2008.

[21] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proceedings of the International Conference on Machine Learning*, pages 214–223, 2017.

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

[23] Arturs Backurs, Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Scalable nearest neighbor search for optimal transport. *Proceedings of the International Conference on Machine Learning*, pages 497–506, 2020.

[24] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in real algebraic geometry*. Springer, 2006.

[25] 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.

[26] 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.

[27] Espen Bernton, Pierre E. Jacob, Mathieu Gerber, and Christian P. Robert. On parameter estimation with the Wasserstein distance. *Information and Inference: A Journal of the IMA*, 8(4):657–676, 2019.

[28] Jérémie Bigot, Elsa Cazelles, and Nicolas Papadakis. Central limit theorems for entropy-regularized optimal transport on finite spaces and statistical applications. *Electronic Journal of Statistics*, 13:5120–5150, 2019.

[29] Jose Blanchet, Arun Jambulapati, Carson Kent, and Aaron Sidford. Towards optimal running times for optimal transport. *Operations Research Letters*, 52:107054, 2024.

[30] Jacek Bochnak, Michel Coste, and Marie-Françoise Roy. *Real Algebraic Geometry*, volume 36. Springer, 2013.

[31] Ali Borji. Pros and cons of GAN evaluation measures. *Computer Vision and Image Understanding*, 179:41–65, 2019.

[32] Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Mądry. K-server via multiscale entropic regularization. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing*, pages 3–16, 2018.

[33] Luis A. Caffarelli and Robert J. McCann. Free boundaries in optimal transport and Monge-Ampere obstacle problems. *Annals of Mathematics*, pages 673–730, 2010.

[34] Jiezhang Cao, Langyuan Mo, Yifan Zhang, Kui Jia, Chunhua Shen, and Mingkui Tan. Multi-marginal Wasserstein GAN. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 1776–1786, 2019.

[35] Sergio Caracciolo, Carlo Lucibello, Giorgio Parisi, and Gabriele Sicuro. Scaling hypothesis for the Euclidean bipartite matching problem. *Physical Review E*, 90(1): 012118, 2014.

[36] Wanxing Chang, Ye Shi, and Jingya Wang. Csot: Curriculum and structure-aware optimal transport for learning with noisy labels. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pages 8528–8541, 2023.

[37] Laetitia Chapel, Mokhtar Z. Alaya, and Gilles Gasso. Partial optimal transport with applications on positive-unlabeled learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, pages 2903–2913, 2020.

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

[39] 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 *Proceedings of the 63rd IEEE Annual Symposium Foundations of Computer Science*, pages 612–623. IEEE, 2022.

[40] 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 *Proceedings of the International Conference on Machine Learning*, pages 1071–1080, 2019.

[41] Jaemoo Choi, Jaewoong Choi, and Myungjoo Kang. Generative modeling through the semi-dual formulation of unbalanced optimal transport. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pages 42433–42455, 2023.

[42] Marek Chrobak and Lawrence L. Larmore. An optimal on-line algorithm for k servers on trees. *SIAM Journal of Computing*, 20(1):144–148, 1991.

[43] Ching-Yao Chuang, R. Devon Hjelm, Xin Wang, Vibhav Vineet, Neel Joshi, Antonio Torralba, Stefanie Jegelka, and Yale Song. Robust contrastive learning against noisy views. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16670–16681, 2022.

[44] Sebastian Claici, Edward Chien, and Justin Solomon. Stochastic Wasserstein barycenters. In *Proceedings of the International Conference on Machine Learning*, pages 999–1008, 2018.

[45] Kenneth L. Clarkson, Herbert Edelsbrunner, Leonidas J. Guibas, Micha Sharir, and Emo Welzl. Combinatorial complexity bounds for arrangements of curves and spheres. *Discrete and Computational Geometry*, 5(2):99–160, 1990.

[46] Nicolas Courty, Rémi Flamary, and Devis Tuia. Domain adaptation with regularized

optimal transport. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 274–289. Springer, 2014.

[47] Nicolas Courty, Rémi Flamary, Devis Tuia, and Alain Rakotomamonjy. Optimal transport for domain adaptation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(9):1853–1865, 2016.

[48] Nicolas Courty, Rémi Flamary, Amaury Habrard, and Alain Rakotomamonjy. Joint distribution optimal transportation for domain adaptation. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 3733–3742, 2017.

[49] Gianluca Crippa, Chloé Jimenez, and Aldo Pratelli. Optimum and equilibrium in a transport problem with queue penalization effect. *Advances in Calculus of Variations*, 2(3):207–246, 2009.

[50] Marco Cuturi. Sinkhorn distances: lightspeed computation of optimal transport. In *Proceedings of the 27th International Conference on Neural Information Processing Systems-Volume 2*, pages 2292–2300, 2013.

[51] Marco Cuturi and Arnaud Doucet. Fast computation of Wasserstein barycenters. In *Proceedings of the International Conference on Machine Learning*, pages 685–693, 2014.

[52] Fernando De Goes, David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. An optimal transport approach to robust reconstruction and simplification of 2d shapes. *Computer Graphics Forum*, 30(5):1593–1602, 2011.

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

[54] Nabarun Deb and Bodhisattva Sen. Multivariate rank-based distribution-free nonparametric testing using measure transportation. *Journal of the American Statistical Association*, 118(541):192–207, 2023.

[55] Nabarun Deb, Bhaswar B. Bhattacharya, and Bodhisattva Sen. Efficiency lower bounds for distribution-free Hotelling-type two-sample tests based on optimal transport. *ArXiv preprint arXiv:2104.01986*, 2021.

[56] Eustasio Del Barrio and Jean-Michel Loubes. Central limit theorems for empirical transportation cost in general dimension. *The Annals of Probability*, 47(2):926 – 951, 2019.

[57] Ishan Deshpande, Ziyu Zhang, and Alexander G. Schwing. Generative modeling using the sliced Wasserstein distance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3483–3491, 2018.

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

[59] Richard M. Dudley. Real analysis and probability. *American history*, 1861(1900), 1945.

[60] Richard M. Dudley. The speed of mean Glivenko-Cantelli convergence. *The Annals of Mathematical Statistics*, 40(1):40–50, 1969.

[61] Pavel Dvurechensky, Alexander Gasnikov, and Alexey Kroshnin. Computational optimal transport: Complexity by accelerated gradient descent is better than by Sinkhorn's algorithm. In *Proceedings of the International Conference on Machine Learning*, pages 1367–1376, 2018.

[62] Nabil El Malki, Robin Cugny, Olivier Teste, and Franck Ravat. DECWA: Density-based clustering using Wasserstein distance. In *Proceedings of the 29th ACM Inter-*

national Conference on Information and Knowledge Management, pages 2005–2008, 2020.

[63] Peyman Mohajerin Esfahani and Daniel Kuhn. Data-driven distributionally robust optimization using the Wasserstein metric: Performance guarantees and tractable reformulations. *Mathematical Programming*, 171(1):115–166, 2018.

[64] Lawrence C. Evans. Partial differential equations and Monge-Kantorovich mass transfer. *Current Developments in Mathematics*, 1997(1):65–126, 1997.

[65] Alessio Figalli. The optimal partial transport problem. *Archive for Rational Mechanics and Analysis*, 195(2):533–560, 2010.

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

[67] Nicolas Fournier and Arnaud Guillin. On the rate of convergence in Wasserstein distance of the empirical measure. *Probability Theory and Related Fields*, 162(3):707–738, 2015.

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

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

[70] Harold N. Gabow and Robert E. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal of Computing*, 18(5):1013–1036, 1989.

[71] Harold N. Gabow and Robert E. Tarjan. Faster scaling algorithms for general graph-matching problems. *Journal ACM*, 38(4):815–853, 1991.

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

[73] 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.

[74] 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.

[75] Akshaykumar G Gattani, Sharath Raghvendra, and Pouyan Shirzadian. A robust exact algorithm for the Euclidean bipartite matching problem. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pages 51706–51718, 2023.

[76] Darius Geiß, Rolf Klein, Rainer Penninger, and Günter Rote. Optimally solving a transportation problem using Voronoi diagrams. *Computational Geometry*, 46(8):1009–1016, 2013.

[77] Aude Genevay, Marco Cuturi, Gabriel Peyré, and Francis Bach. Stochastic optimization for large-scale optimal transport. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 3440–3448, 2016.

[78] Aude Genevay, Gabriel Peyre, and Marco Cuturi. Learning generative models with

Sinkhorn divergences. *International Conference on Artificial Intelligence and Statistics*, page 1608–1617, 2018.

[79] Javier González-Delgado, Alberto González-Sanz, Juan Cortés, and Pierre Neuvial. Two-sample goodness-of-fit tests on the flat torus based on Wasserstein distance and their relevance to structural biology. *Electronic Journal of Statistics*, 17(1):1547–1586, 2023.

[80] Wenshuo Guo, Nhat Ho, and Michael Jordan. Fast algorithms for computational optimal transport and Wasserstein barycenter. *In International Conference on Artificial Intelligence and Statistics*, pages 2088–2097, 2020.

[81] Xin Guo, Johnny Hong, Tianyi Lin, and Nan Yang. Relaxed Wasserstein with applications to GANs. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3325–3329. IEEE, 2021.

[82] Rishi Gupta, Piotr Indyk, and Eric Price. Sparse recovery for earth mover distance. In *2010 48th Annual Allerton Conference on Communication, Control, and Computing*, pages 1742–1744. IEEE, 2010.

[83] Marc Hallin, Gilles Mordant, and Johan Segers. Multivariate goodness-of-fit tests based on Wasserstein distance. *Electronic Journal of Statistics*, 15:1328–1371, 2021.

[84] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *The Collected Works of Wassily Hoeffding*, pages 409–426, 1994.

[85] 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.

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

[87] Masaaki Imaizumi, Hirofumi Ota, and Takuo Hamaguchi. Hypothesis test and confidence analysis with Wasserstein distance on general dimension. *Neural Computation*, 34(6):1448–1487, 2022.

[88] Hicham Janati, Marco Cuturi, and Alexandre Gramfort. Wasserstein regularization for sparse multi-task regression. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1407–1416, 2019.

[89] Leonid V Kantorovich. On the transfer of masses. *Doklady Akademii Nauk*, 37, 1942.

[90] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. *Discrete & Computational Geometry*, 64:838–904, 2020.

[91] Andrey Boris Khesin, Aleksandar Nikolov, and Dmitry Paramonov. Preconditioning for the geometric transportation problem. In *35th International Symposium on Computational Geometry*, pages 15–1, 2019.

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

[93] 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.

[94] Soheil Kolouri, Navid Naderializadeh, Gustavo K Rohde, and Heiko Hoffmann. Wasserstein embedding for graph learning. In *Proceedings of the International Conference on Learning Representations*, 2021.

[95] Elias Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009.

[96] Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42(5):971–983, 1995.

[97] Harold W Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[98] Nathaniel Lahn and Sharath Raghvendra. A faster algorithm for minimum-cost bipartite matching in minor-free graphs. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 569–588, 2019.

[99] Khang Le, Huy Nguyen, Quang M. Nguyen, Tung Pham, Hung Bui, and Nhat Ho. On robust optimal transport: computational complexity and barycenter computation. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, pages 21947–21959, 2021.

[100] 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.

[101] James R. Lee. Fusible HSTs and the randomized k-server conjecture. In *Proceedings of the 59th IEEE Annual Symposium on Foundations of Computer Science*, pages 438–449, 2018.

[102] Bruno Lévy. Partial optimal transport for a constant-volume Lagrangian mesh with free boundaries. *Journal of Computational Physics*, 451:110838, 2022.

[103] 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.

[104] Richard J. Lipton and Robert E. Tarjan. Applications of a planar separator theorem. *SIAM Journal on Computing*, 9(3):615–627, 1980.

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

[106] Giulia Luise, Alessandro Rudi, Massimiliano Pontil, and Carlo Ciliberto. Differential properties of Sinkhorn approximation for learning with Wasserstein distance. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 5864–5874, 2018.

[107] Yiming Ma, Hang Liu, Davide La Vecchia, and Matthieu Lerasle. Inference via robust optimal transportation: Theory and methods. *International Statistical Review*, 2025.

[108] Mark S. Manasse, Lyle A. McGeoch, and Daniel D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990.

[109] Hermina Petric Maretic, Mireille El Gheche, Giovanni Chierchia, and Pascal Frossard. Got: an optimal transport framework for graph comparison. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 13899–13910, 2019.

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

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

[112] Quentin Mérigot, Jocelyn Meyron, and Boris Thibert. An algorithm for optimal trans-

port between a simplex soup and a point cloud. *SIAM Journal of Imaging Sciences*, 11(2):1363–1389, 2018.

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

[114] Jocelyn Meyron, Quentin Mérigot, and Boris Thibert. Light in power: a general and parameter-free algorithm for caustic design. *ACM Transactions of Graphics*, 37(6): 1–13, 2018.

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

[116] Gaspard Monge. Mémoire sur la théorie des déblais et des remblais. *Mem. Math. Phys. Acad. Royale Sci.*, pages 666–704, 1781.

[117] Debarghya Mukherjee, Aritra Guha, Justin M Solomon, Yuekai Sun, and Mikhail Yurochkin. Outlier-robust optimal transport. In *International Conference on Machine Learning*, pages 7850–7860, 2021.

[118] Krati Nayyar and Sharath Raghvendra. An input sensitive online algorithm for the metric bipartite matching problem. In *Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science*, pages 505–515, 2017.

[119] Sloan Nietert, Ziv Goldfeld, and Rachel Cummings. Outlier-robust optimal transport: Duality, structure, and statistical analysis. In *International Conference on Artificial Intelligence and Statistics*, pages 11691–11719, 2022.

[120] Sloan Nietert, Ziv Goldfeld, and Soroosh Shafiee. Outlier-robust wasserstein dro. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pages 62792–62820, 2023.

[121] 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.

[122] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mido Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jégou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research Journal*, pages 1–31, 2024.

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

[124] Victor M Panaretos and Yoav Zemel. Statistical aspects of Wasserstein distances. *Annual Review of Statistics and Its Application*, 6:405–431, 2019.

[125] Abhijeet Phatak, Sharath Raghvendra, Chittaranjan Tripathy, and Kaiyi Zhang. Computing all optimal partial transports. In *Proceedings of the International Conference on Learning Representations*, 2022.

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

[127] Kent Quanrud. Approximating optimal transport with linear programs. In *2nd Symposium on Simplicity in Algorithms*, pages 6–1, 2019.

[128] Stephan Rabanser, Stephan Günnemann, and Zachary C Lipton. Failing loudly: an empirical study of methods for detecting dataset shift. In *Proceedings of the 33rd*

International Conference on Neural Information Processing Systems, pages 1396–1408, 2019.

[129] Julien Rabin, Julie Delon, and Yann Gousseau. Circular earth mover's distance for the comparison of local features. In *Proceedings of the 19th International Conference on Pattern Recognition*, pages 1–4. IEEE, 2008.

[130] Svetlozar T Rachev. The Monge-Kantorovich mass transference problem and its stochastic applications. *Theory of Probability and Its Applications*, 29(4):647–676, 1985.

[131] Sharath Raghvendra. A robust and optimal online algorithm for minimum metric bipartite matching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 18–1, 2016.

[132] Sharath Raghvendra. Optimal analysis of an online algorithm for the bipartite matching problem on a line. In *34th International Symposium on Computational Geometry*, pages 67:1–67:14, 2018.

[133] 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.

[134] Sharath Raghvendra and Rachita Sowle. A scalable work function algorithm for the k-server problem. In *18th Scandinavian Symposium and Workshops on Algorithm Theory*, pages 30–1, 2022.

[135] Sharath Raghvendra, Pouyan Shirzadian, and Kaiyi Zhang. A new robust partial p-Wasserstein-based metric for comparing distributions. In *Proceedings of the 41st International Conference on Machine Learning*, pages 41867–41885, 2024.

[136] Sharath Raghvendra, Pouyan Shirzadian, and Rachita Sowle. Geometric bipartite matching based exact algorithms for server problems. In *41st International Symposium on Computational Geometry*, pages 72–1, 2025.

[137] Aaditya Ramdas, Nicolás García Trillos, and Marco Cuturi. On Wasserstein two-sample testing and related families of nonparametric tests. *Entropy*, 19(2):47, 2017.

[138] Lyle Ramshaw and Robert E Tarjan. A weight-scaling algorithm for min-cost imperfect matchings in bipartite graphs. In *Proceedings of the 53rd Annual Symposium on Foundations of Computer Science*, pages 581–590. IEEE, 2012.

[139] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.

[140] Tomislav Rudec, Alfonzo Baumgartner, and Robert Manger. A fast work function algorithm for solving the k-server problem. *Central European Journal of Operations Research*, 21:187–205, 2013.

[141] Tim Salimans, Han Zhang, Alec Radford, and Dimitris Metaxas. Improving GANs using optimal transport. *Proceedings of the International Conference on Learning Representations*, 2018.

[142] F. Santambrogio. *Optimal Transport for Applied Mathematicians: Calculus of Variations, PDEs, and Modeling*. Springer, 1st edition, 2015.

[143] Yuliy Schwartzburg, Romain Testuz, Andrea Tagliasacchi, and Mark Pauly. High-contrast computational caustic design. *ACM Transactions of Graphics*, 33(4):1–11, 2014.

[144] Ravi Seshadri and Karthik K Srinivasan. Algorithm for determining path of maximum reliability on a network subject to random arc connectivity failures. *Transportation Research Record*, 2467(1):80–90, 2014.

[145] R. Sharathkumar. A sub-quadratic algorithm for bipartite matching of planar points with bounded integer coordinates. In *Proceedings of the 29th Annual Symposium on Computational Geometry*, pages 9–16, 2013.

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

[147] Jonah Sherman. Generalized preconditioning and undirected minimum-cost flow. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 772–780, 2017.

[148] Peter Williston Shor. *Random planar matching and bin packing*. PhD thesis, Massachusetts Institute of Technology, 1985.

[149] Daniel D. Sleator and Robert E. Tarjan. A data structure for dynamic trees. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, pages 114–122, 1981.

[150] Max Sommerfeld, Jörn Schrieber, Yoav Zemel, and Axel Munk. Optimal transport: Fast probabilistic approximation with exact solvers. *Journal of Machine Learning Research*, 20(105):1–23, 2019.

[151] Rachita Sowle. *Combinatorial Algorithms for Server Allocation Problem*. PhD thesis, Virginia Tech, 2024.

[152] Pravin M Vaidya. Geometry helps in matching. *SIAM Journal of Computing*, 18(6): 1201–1225, 1989.

[153] Marc Van Kreveld, Frank Staals, Amir Vaxman, and Jordi Vermeulen. Approximating the earth mover's distance between sets of geometric objects. *ArXiv preprint arXiv:2104.08136*, 2021.

[154] Tomas Vaskevicius and Lénaïc Chizat. Computational guarantees for doubly entropic Wasserstein barycenters. In *Proceedings of the 37th Conference on Neural Information Processing Systems*, 2023.

[155] Cédric Vincent-Cuaz, Rémi Flamary, Marco Corneli, Titouan Vayer, and Nicolas Courty. Semi-relaxed Gromov-Wasserstein divergence and applications on graphs. In *Proceedings of the International Conference on Learning Representations*, 2021.

[156] Xu-Jia Wang. On the design of a reflector antenna. *Inverse Problems*, 12(3):351, 1996.

[157] Jonathan Weed and Francis Bach. Sharp asymptotic and finite-sample rates of convergence of empirical measures in Wasserstein distance. *Bernoulli*, 25(4A):2620 – 2648, 2019.

[158] Emo Welzl. Partition trees for triangle counting and other range searching problems. In *Proceedings of the 4th Annual Symposium Computational Geometry*, pages 23–33, 1988.

[159] Jingjing Xu, Hao Zhou, Chun Gan, Zaixiang Zheng, and Lei Li. Vocabulary learning via optimal transport for neural machine translation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7361–7373, 2021.

[160] Mikhail Yurochkin, Sebastian Claici, Edward Chien, Farzaneh Mirzazadeh, and Justin Solomon. Hierarchical optimal transport for document representation. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 1601–1611, 2019.

[161] Yubo Zhuang, Xiaohui Chen, and Yun Yang. Wasserstein K-means for clustering probability distributions. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, pages 11382–11395, 2022.

[162] Goran Zuzic. A simple boosting framework for transshipment. In *Proceedings of the 31st Annual European Symposium on Algorithms*, volume 274, pages 104–1, 2023.

Appendices

Appendix A

Semi-Discrete OT

A.1 Missing Details and Proofs of Chapter 3

In this section, we present the missing details and the proofs of the claims made in Section 3.

Next, we show that if there exists a transport plan τ from μ to ν, a set of dual weights $y(\cdot)$ for points in B, and a set of dual weights $y'(\cdot)$ for representative points of the regions in $\mathcal{X}_\delta$ that satisfy δ-feasibility conditions (3.6) and (3.7) (in which $y_\delta(\cdot)$ is replaced with $y'(\cdot)$), then reassigning the dual weights based on Equation (3.5) does not violate conditions (3.6) and (3.7), i.e., the transport plan τ and dual weights $y(\cdot)$ for points in B are δ-feasible.

Lemma A.1. *For any scale δ, if there exists a transport plan τ from μ to ν, a set of dual weights $y(\cdot)$ for points in B, and a set of dual weights $y'(\cdot)$ for representative points of regions in $\mathcal{X}_\delta$ satisfying δ-feasibility conditions (3.6) and (3.7), then $\tau, y(\cdot)$ are δ-feasible.*

Proof. To prove this lemma, we show that conditions (3.6) and (3.7) hold when plugging dual weights $y(\cdot)$ for points in B and dual weights $y_\delta(\cdot)$ derived by Equation (3.5) for representative points of $\mathcal{X}_\delta$. For any region $\varrho \in \mathcal{X}_\delta$, let b_ϱ denote the weighted nearest neighbor of r_ϱ in B with respect to weights $y(\cdot)$. For any pair $(\varrho, b) \in \mathcal{X}_\delta \times B$,

$$y_\delta(r_\varrho) = y(b_\varrho) - \mathrm{c}(r_\varrho, b_\varrho) - \delta \geq y(b) - \mathrm{c}(r_\varrho, b) - \delta;$$

therefore, the feasibility condition (3.6) holds for (ϱ, b). Next, we show that the feasibility condition (3.7) also holds for all pairs (ϱ, b) with $\tau(\varrho, b) > 0$. By condition (3.6) on $\tau, y(\cdot), y'(\cdot)$, for any point $b' \in B$, $y'(r_\varrho) \geq y(b') - \mathrm{c}(r_\varrho, b') - \delta$. Therefore,

$$y(r_\varrho) \geq \max_{b' \in B}(y(b') - \mathrm{c}(r_\varrho, b') - \delta) = y(b_\varrho) - \mathrm{c}(r_\varrho, b_\varrho) - \delta = y_\delta(r_\varrho).$$

As a result, for the point $b \in B$ with $\tau(\varrho, b) > 0$, by condition (3.7) on $\tau, y(\cdot), y'(\cdot)$, we have

$$y(b) - y_\delta(r_\varrho) \geq y(b) - y'(r_\varrho) \geq \mathrm{c}(r_\varrho, b),$$

and the δ-feasibility condition (3.7) holds after replacing $y'(\cdot)$ with $y_\delta(\cdot)$. □

Lemma 3.4. *Let $\tau_{2\delta}, y(\cdot)$ be any 2δ-feasible transport plan between μ and ν, where the dual weights of points in B are integer multiples of 2δ. Then, for any region $\varrho \in \mathcal{X}_\delta$ and any point $b \in B$, if $\tau_{2\delta}(\varrho, b) > 0$, then $s_\delta(\varrho, b) \leq 4\delta$.*

Proof. Let ϱ^* denote the region in $\mathcal{X}_{2\delta}$ containing ϱ (by construction, it can be easily confirmed that the set of valid weight vectors $\mathbb{W}_{2\delta}$ is a subset of $\mathbb{W}_\delta$ and hence, each region in $\mathcal{X}_\delta$ completely lies inside a region in $\mathcal{X}_{2\delta}$). Define b_ϱ to be the weighted nearest neighbor of r_ϱ (and consequently r_{ϱ^*}) with respect to weights $y(\cdot)$. By Equation (3.5), $y_{2\delta}(r_{\varrho^*}) = y(b_\varrho) - \mathrm{c}(r_{\varrho^*}, b_\varrho) - 2\delta$ and $y_\delta(r_\varrho) = y(b_\varrho) - \mathrm{c}(r_\varrho, b_\varrho) - \delta$. Hence,

$$\begin{aligned} s_\delta(\varrho, b) &= \left\lfloor \frac{\mathrm{c}(r_\varrho, b) + \delta - y(b) + y_\delta(r_\varrho)}{\delta} \right\rfloor \delta \\ &= \left\lfloor \frac{\mathrm{c}(r_\varrho, b) + \delta - y(b) + (y(b_\varrho) - \mathrm{c}(r_\varrho, b_\varrho) - \delta)}{\delta} \right\rfloor \delta \\ &= \left\lfloor \frac{\mathrm{c}_y(r_\varrho, b) - \mathrm{c}_y(r_\varrho, b_\varrho)}{\delta} \right\rfloor \delta \leq \left\lfloor \frac{\mathrm{c}_y(r_{\varrho^*}, b) - \mathrm{c}_y(r_{\varrho^*}, b_\varrho)}{\delta} \right\rfloor \delta + 2\delta, \end{aligned} \tag{A.1}$$

where the last inequality is resulted from Lemma A.2 below. Finally, from the 2δ-feasibility

condition (3.7) on $\tau_{2\delta}, y(\cdot)$,

$$\mathrm{c}(r_{\varrho^*}, b) \leq y(b) - y_{2\delta}(r_{\varrho^*}) = y(b) - (y(b_\varrho) - \mathrm{c}(r_{\varrho^*}, b_\varrho) - 2\delta).$$

Hence,

$$\mathrm{c}_y(r_{\varrho^*}, b) - \mathrm{c}_y(r_{\varrho^*}, b_\varrho) \leq 2\delta. \tag{A.2}$$

Plugging Equations (A.2) into Equation (A.1),

$$s_\delta(\varrho, b) \leq \left\lfloor \frac{\mathrm{c}_y(r_{\varrho^*}, b) - \mathrm{c}_y(r_{\varrho^*}, b_\varrho)}{\delta} \right\rfloor \delta + 2\delta \leq 4\delta,$$

as claimed. □

Lemma A.2. *For any region $\varrho^* \in \mathcal{X}_{2\delta}$, any pair of points $a_1, a_2 \in \varrho^*$, and any pair of points $b_1, b_2 \in B$, $\left\lfloor \frac{\mathrm{c}(a_1,b_1)-\mathrm{c}(a_1,b_2)}{\delta} \right\rfloor \delta \leq \left\lfloor \frac{\mathrm{c}(a_2,b_1)-\mathrm{c}(a_2,b_2)}{\delta} \right\rfloor \delta + 2\delta$.*

Proof. To prove this lemma, we first construct a valid weight vector $w \in \mathbb{W}_{2\delta}$ such that in the Voronoi diagram $\mathrm{VD}_w(B)$, the region ϱ^* lies inside the Voronoi cell of b_1, which gives us $\mathrm{c}_w(a_1, b_1) \leq \mathrm{c}_w(a_1, b_2)$. Then, we increase the weight of b_2 in w by 2δ and obtain another valid weight vector $w_+ \in \mathbb{W}_{2\delta}$ such that ϱ^* now lies inside the Voronoi cell of b_2 and conclude $\mathrm{c}_w(a_2, b_2) \leq \mathrm{c}_w(a_2, b_1) + 2\delta$. Combining the two bounds, we get $\mathrm{c}(a_1, b_1) - \mathrm{c}(a_1, b_2) \leq \mathrm{c}(a_2, b_1) - \mathrm{c}(a_2, b_2) + 2\delta$, leading to the lemma statement. We describe the details below.

Consider a valid weight vector $w \in \mathbb{W}_{2\delta}$ that assigns $w(b_1) = \left\lceil \frac{\mathrm{c}(r_{\varrho^*}, b_1)}{2\delta} \right\rceil 2\delta$, $w(b_2) = \left\lceil \frac{\mathrm{c}(r_{\varrho^*}, b_2)}{2\delta} \right\rceil 2\delta$, and $w(b') = 0$ for each $b' \neq b_1, b_2$ in B. Without loss of generality, assume $\mathrm{c}_w(r_{\varrho^*}, b_1) < \mathrm{c}_w(r_{\varrho^*}, b_2)$[1]. By construction,

$$-2\delta < \mathrm{c}_w(r_{\varrho^*}, b_1) < \mathrm{c}_w(r_{\varrho^*}, b_2) \leq 0 \leq \min_{b' \in B, b' \neq b_1, b_2} \mathrm{c}_w(r_{\varrho^*}, b).$$

[1] If $\mathrm{c}_w(r_{\varrho^*}, b_1) \geq \mathrm{c}_w(r_{\varrho^*}, b_2)$, one can simply decrease $w(b_2)$ by 2δ and follow a similar argument.

Hence, the point r_{ϱ^*} and consequently the region ϱ^* lie inside the Voronoi cell of b_1 in $\mathrm{VD}_w(B)$. Therefore,

$$\begin{aligned} \mathrm{c}(a_1,b_1) - w(b_1) = \mathrm{c}_w(a_1,b_1) \leq \mathrm{c}_w(a_1,b_2) = \mathrm{c}(a_1,b_2) - w(b_2). \\ \mathrm{c}(a_1,b_1) - \mathrm{c}(a_1,b_2) \leq w(b_1) - w(b_2). \end{aligned} \tag{A.3}$$

Next, consider the weight vector $w_+ \in \mathbb{W}_{2\delta}$ that assigns $w_+(b_2) = w(b_2) + 2\delta$ and $w_+(b') = w(b')$ for all points $b' \neq b_2$ in B. In this case,

$$\mathrm{c}_{w_+}(r_{\varrho^*}, b_2) \leq -2\delta < \mathrm{c}_{w_+}(r_{\varrho^*}, b_1) = \mathrm{c}_w(r_{\varrho^*}, b_1) \leq 0 \leq \min_{b' \in B, b' \neq b_1, b_2} \mathrm{c}_w(r_{\varrho^*}, b).$$

Therefore, the point r_{ϱ^*} and consequently the region ϱ^* lie inside the Voronoi cell of b_2 in $\mathrm{VD}_{w_+}(B)$. Therefore,

$$\begin{aligned} \mathrm{c}(a_2,b_2) - (w(b_2) + 2\delta) = \mathrm{c}_{w_+}(a_2,b_2) \leq \mathrm{c}_{w_+}(a_2,b_1) = \mathrm{c}(a_2,b_1) - w(b_1), \\ w(b_1) - w(b_2) \leq \mathrm{c}(a_2,b_1) - \mathrm{c}(a_2,b_2) + 2\delta. \end{aligned} \tag{A.4}$$

Combining Equations (A.3) and (A.4),

$$\begin{aligned} \mathrm{c}(a_1,b_1) - \mathrm{c}(a_1,b_2) \leq w(b_1) - w(b_2) \leq \mathrm{c}(a_2,b_1) - \mathrm{c}(a_2,b_2) + 2\delta, \\ \left\lfloor \frac{\mathrm{c}(a_1,b_1) - \mathrm{c}(a_1,b_2)}{\delta} \right\rfloor \delta \leq \left\lfloor \frac{\mathrm{c}(a_2,b_1) - \mathrm{c}(a_2,b_2)}{\delta} \right\rfloor \delta + 2\delta. \end{aligned}$$

□

Given two transport plans σ_1 and σ_2 from $\hat{\mu}_\delta$ to ν, we define a directed graph $\mathcal{G}(\sigma_1, \sigma_2)$ on the vertex set $A_\delta \cup B$ as follows. For any pair $(r, b) \in A_\delta \times B$, if $\sigma_1(r,b) > \sigma_2(r,b)$, then we add a forward edge from r to b with a capacity $\sigma_1(r,b) - \sigma_2(r,b)$; otherwise, if $\sigma_1(r,b) < \sigma_2(r,b)$,

then we add a backward edge from b to r with a capacity $\sigma_2(r,b) - \sigma_1(r,b)$.

Lemma A.3. *Given any two transport plans σ_1 and σ_2 from $\hat{\mu}_\delta$ to ν, for any edge $(r,b) \in A_\delta \times B$ in $\mathcal{G}(\sigma_1,\sigma_2)$, there exists a directed cycle C in $\mathcal{G}(\sigma_1,\sigma_2)$ that contains the edge (r,b).*

Proof. To prove this lemma, we conduct a DFS-style search from the point b in $\mathcal{G}(\sigma_1,\sigma_2)$ to compute a directed path P from b to r. This proves the lemma since concatenating the edge (r,b) to P results in a directed cycle on $\mathcal{G}(\sigma_1,\sigma_2)$ containing (r,b). Our proof relies on the following observation: Since both σ_1 and σ_2 are transport plans from $\hat{\mu}_\delta$ to ν, by the construction of $\mathcal{G}(\sigma_1,\sigma_2)$, for any point $u \in A_\delta \cup B$, the total capacity of incoming edges to u is equal to the total capacity of outgoing edges from u.

We conduct a DFS-style procedure that grows a path $P = \langle r = p_0, b = p_1, p_2, \ldots, p_k\rangle$ as follows. Initially, we set $P = \langle r = p_0, b = p_1\rangle$. At each step, for the last point p_k of the path P, let $N(p_k)$ denote the set of all outgoing edges from p_k. Note that since there exists an incoming edge (p_{k-1}, p_k) in $\mathcal{G}(\sigma_1,\sigma_2)$, by the observation stated above, $N(p_k)$ is not empty. Consider any point $p \in N(p_k)$.

- If $p = r$, then $P \circ (p_k, r)$ is a cycle containing (r,b), as desired.
- Otherwise, if p already exists in the path P as p_i for some $i \geq 1$, then we have found a cycle $C = \langle p_i, p_{i+1}, \ldots, p_k, p_i\rangle$. We "cancel" this cycle as follows. Define the capacity of the cycle $c(C)$ as the minimum capacity of all edges on C. We then decrease the capacity of all edges on C by $c(C)$ and for those that now have a zero capacity, we simply remove them from $\mathcal{G}(\sigma_1,\sigma_2)$. We set $P = \langle p_0 = r, p_1 = b, \ldots, p_i\rangle$ and continue our search.
- Otherwise, we add p as p_{k+1} to the path P and continue the search.

Note that since all edges in $\mathcal{G}(\sigma_1, \sigma_2)$ have a positive finite capacity at all times, each time we cancel a cycle reduces the total capacity of the edges of $\mathcal{G}(\sigma_1, \sigma_2)$. Furthermore, the length of the path P will never be more than $2n$, as there are only n points in the set B and $\mathcal{G}(\sigma_1, \sigma_2)$ is a bipartite graph. Hence, our DFS-style procedure will terminate by returning a cycle containing (r, b). □

A.2 Missing Details and Proofs of Chapter 4

A.2.1 Missing Proofs of Section 4.1

In this section, we present the missing proofs and details of our combinatorial framework described in Section 4.1.

Residual graph. Below, 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 4.2. 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 Vor_b^0 (resp. Vor_b^δ, $\mathrm{Vor}_b^{2\delta}$) 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 Vor_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 $\mathrm{Vor}_{b_1}^i$ and $\mathrm{Vor}_{b_2}^j$, for some $i, j \in \{0, \delta, 2\delta\}$. We show that $\mathrm{Vor}_{b_1}^i$ and $\mathrm{Vor}_{b_2}^j$ 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 $\mathrm{c}_y(x, b) = \mathrm{c}_y(x, b')$. Note that under the squared Euclidean distances, the weighted bisector of two points is a straight line. Let v denote an intersection point of $\mathrm{Vor}_{b_1}^i$ and $\mathrm{Vor}_{b_2}^j$, and suppose v lies on the segment of $\mathrm{Vor}_{b_1}^i$ corresponding to the weighted bisector of b_1 and a point $b_3 \in B$ (See Figure A.1). Note that if $b_3 = b_2$, then $\mathrm{Vor}_{b_1}^i$ and $\mathrm{Vor}_{b_2}^j$ share a segment containing v, which means that

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

and also

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

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,

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

By the construction of the weight function $y_{b_1}^i(\cdot)$ in Equation (4.1), for any point $b \in B \setminus$

$\{b_1, b_2\}$, $y^i_{b_1}(b) = y^j_{b_2}(b) = y(b)$; thus, using Equation (A.5),

$$\mathrm{c}_{y^j_{b_2}}(v, b_3) = \mathrm{c}_{y^i_{b_1}}(v, b_3) = \min_{b \in B \setminus \{b_1, b_2\}} \mathrm{c}_{y^i_{b_1}}(v, b) = \min_{b \in B \setminus \{b_1, b_2\}} \mathrm{c}_{y^j_{b_2}}(v, b). \tag{A.6}$$

Using Equation (A.6), the point v also lies in the segment of $\mathrm{Vor}^j_{b_2}$ that corresponds to the weighted bisector of b_2 and b_3 with respect to weights $y^j_{b_2}$. Thus,

$$\mathrm{c}_{y^i_{b_1}}(v, b_1) = \mathrm{c}_{y^i_{b_1}}(v, b_3) = \mathrm{c}_{y^j_{b_2}}(v, b_3) = \mathrm{c}_{y^j_{b_2}}(v, b_2).$$

In other words, the point v satisfies

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

i.e., the point v lies on the weighted bisector of b_1 and b_2 (the blue dashed line in Figure A.1). Since the weighted bisector is a straight line, it intersects the convex polygon $\mathrm{Vor}^i_{b_1}$ in at most 2 points; hence, the two Voronoi cells $\mathrm{Vor}^i_{b_1}$ and $\mathrm{Vor}^j_{b_2}$ intersect each other in at most 2 points. Since there 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 4.2. *For any $\delta > 0$ and a 2δ-feasible transport plan $\hat{\tau}, y(\cdot)$, the residual graph $\mathcal{G}_\delta$ has $O(n^2)$ nodes and $O(n^3)$ edges.*

Next, we extend our bounds for the number of vertices and edges of the residual graph in Lemma 4.2 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

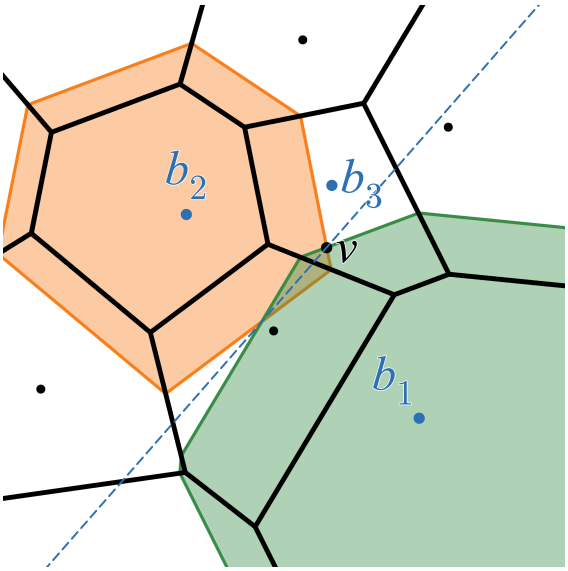


Figure A.1: For two cells $\mathrm{Vor}^i_{b_1}$ (green cell) and $\mathrm{Vor}^j_{b_2}$ (orange cell), any intersection point v lies on their weighted bisector.

$O(n^d)$ 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 Vor^i_b 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 $(\mathrm{Vor}^{i_1}_{b_1}, \mathrm{Vor}^{i_2}_{b_2}, \ldots, \mathrm{Vor}^{i_d}_{b_d})$ 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 $O(n^d)$[2]. Therefore, we conclude that the number of intersection points is $O(n^d)$, as desired.

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

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

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

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

  $$\mathrm{c}_y(v, b_1) - i_1 = \mathrm{c}_{y_{b_1}^{i_1}}(v, b_1) = \mathrm{c}_{y_{b_1}^{i_1}}(v, b_j) = \mathrm{c}_y(v, b_j),$$

  and also

  $$\mathrm{c}_y(v, b_j) - i_j = \mathrm{c}_{y_{b_j}^{i_j}}(v, b_j) = \mathrm{c}_{y_{b_j}^{i_j}}(v, b_1) = \mathrm{c}_y(v, b_1).$$

  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 (A.7),

$$\mathrm{c}_y(v, b_{d+1}) = \mathrm{c}_{y_{b_1}^{i_1}}(v, b_{d+1}) = \min_{b \in B} \mathrm{c}_{y_{b_1}^{i_1}}(v, b) = \min_{b \in B} \mathrm{c}_y(v, b), \tag{A.8}$$

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

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

From Equation (A.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 $(\mathrm{Vor}_{b_1}^{i_1}, \mathrm{Vor}_{b_2}^{i_2}, \ldots, \mathrm{Vor}_{b_d}^{i_d})$ intersect each other in at most 2 points.

Lemma A.4. *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.*

A.2.2 Missing Details of Section 4.2

In this section, we provide the proofs of the properties of the three procedures.

Lemma 4.6. *Suppose invariants (I1) and (I2) hold at the start of the* SEARCHANDAUGMENT *procedure. Then, after the execution of the* SEARCHANDAUGMENT *procedure,*

(S1) the transport plan $\hat{\tau}_\delta, y(\cdot)$ remains 2δ-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 4.5, augmenting $\hat{\tau}_\delta$ 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

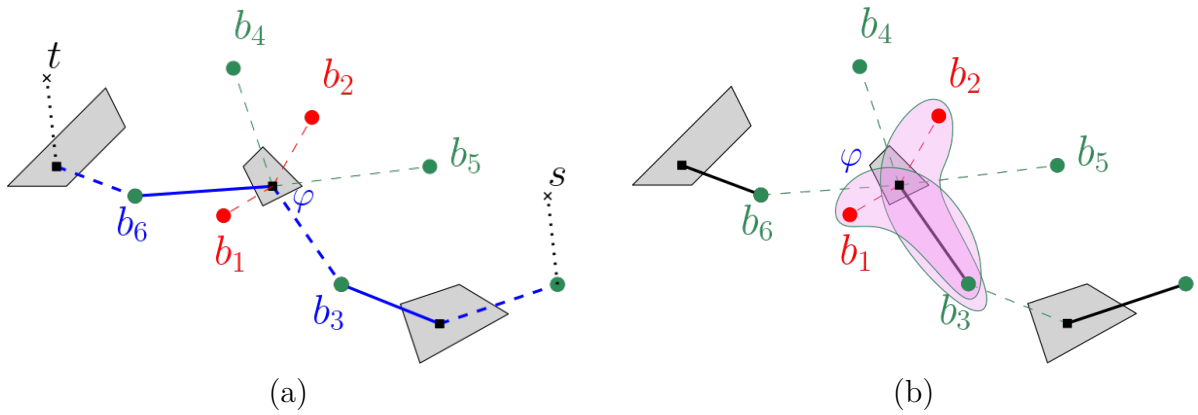


Figure A.2: (a) An augmenting path found by the SearchAndAugment procedure (blue path), and (b) two admissible triples (highlighted in pink) formed after augmentation.

we formally state and prove in Lemma A.5 below. Using that, we first prove (S3) and then prove (S2).

In Figure A.2(a), 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 A.2(b)), 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 A.5, 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 A.5, 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 A.5, 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.

□

Lemma A.5. *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 $\mathrm{c}_y(r, b) > \mathrm{c}_y(r, b')$. 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 $\mathrm{c}_y(r, b) > \mathrm{c}_y(r, b')$ 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). □

Lemma 4.7. *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 2δ-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 $\mathrm{c}_{y'}(a,b) - 2\delta \leq \mathrm{c}_{y'}(a,b'_a)$. To prove property (W1), we show that $\mathrm{c}_y(a,b) - 2\delta \leq \mathrm{c}_y(a,b_a)$.

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

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

  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, $\mathrm{c}_y(a,b) - 2\delta \leq \mathrm{c}_y(a,b_a)$.

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

  - If $b_a \in \mathcal{K}$, then $\mathrm{c}_{y'}(r_{\varphi_a},b) \leq \mathrm{c}_{y'}(r_{\varphi_a},b_a)$ (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,

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

    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 Vor^δ_b and $\mathrm{Vor}^\delta_{b_a}$. Thus,

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

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

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

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 $\varphi \in \mathrm{Vor}_b^\delta$, since we defined $\mathrm{c}_y(r_\varphi, t) = \min_{b' \in B} \mathrm{c}_y(r_\varphi, b')$, 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). □

Lemma 4.8. *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 2δ-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 acyclic. 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 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 $\mathrm{Vor}_b^{2\delta}$, 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 $C^1, \dots, C^k$ denote the sequence of admissible cycles found by the procedure, and let $\hat{\tau}_\delta^0, \dots, \hat{\tau}_\delta^k$ 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 A.5, 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. □

A.2.3 Missing Details of Section 4.3

In this section, we analyze the efficiency of the three procedures SearchAndAugment, IncreaseWeights, and Acyclify.

Lemma 4.9. *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 A.5, for any newly formed admissible triples (b', r, b), the point b' is already marked as visited. By property (S2) in Lemma 4.6, 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 $O(n^{d+1})$. □

Lemma 4.10. *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 after the execution of the first step 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 A.5, 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 4.8, 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 $O(n^{d+1})$. □

Number of Iterations

Lemma 4.11. *For each scale δ, the second step of our algorithm runs $O(n)$ iterations.*

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

$$\begin{aligned}\phi(P) &= \mathrm{c}(r_1, b_0) + \sum_{i=1}^{k} \big[\mathrm{c}(r_i, b_i) - \mathrm{c}(r_i, b_{i-1})\big] \\ &= \mathrm{c}_{y_\delta}(r_1, b_0) + \sum_{i=1}^{k} \big[\mathrm{c}_{y_\delta}(r_i, b_i) - \mathrm{c}_{y_\delta}(r_i, b_{i-1})\big] + y_\delta(b_k).\end{aligned}$$

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, $\mathrm{c}_{y_\delta}(r_i, b_i) - \mathrm{c}_{y_\delta}(r_i, b_{i-1}) \geq -2\delta$. Since the length of P is at most $2n - 1$,

$$\phi(P) = \mathrm{c}_{y_\delta}(r_1, b_0) + \sum_{i=1}^{k} \big[\mathrm{c}_{y_\delta}(r_i, b_i) - \mathrm{c}_{y_\delta}(r_i, b_{i-1})\big] + y_\delta(b_k) \geq \mathrm{c}_{y_\delta}(r_1, b_0) + y_\delta(b_k) - 2n\delta. \quad \text{(A.10)}$$

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

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

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, $\mathrm{c}_{y_{2\delta}}(r_i, b_i) - \mathrm{c}_{y_{2\delta}}(r_i, b_{i-1}) \leq 4\delta$. Since the length of P is at most $2n - 1$,

$$\phi(P) = \mathrm{c}_{y_{2\delta}}(r_1, b_0) + \sum_{i=1}^{k} \left[\mathrm{c}_{y_{2\delta}}(r_i, b_i) - \mathrm{c}_{y_{2\delta}}(r_i, b_{i-1})\right] + y_{2\delta}(b_k) \leq \mathrm{c}_{y_{2\delta}}(r_1, b_0) + y_{2\delta}(b_k) + 4n\delta. \tag{A.11}$$

By property (W3) in Lemma 4.8, 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 $\mathrm{c}_{y_{2\delta}}(r_1, b_0) = \mathrm{c}_{y_\delta}(r_1, b_0)$. Combining with Equations (A.10) and (A.11),

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

Equivalently,

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

By property (W2) in Lemma 4.8, 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. □

A.3 Missing Details and Proofs of Chapter 5

In this section, we present the missing details and proofs of the lemmas in Section 5.3. We begin by presenting missing proofs of our combinatorial framework in Section A.3.1. We then provide the details of the implementation of the procedures in Section A.3.2–A.3.4.

A.3.1 Missing proofs of the combinatorial framework

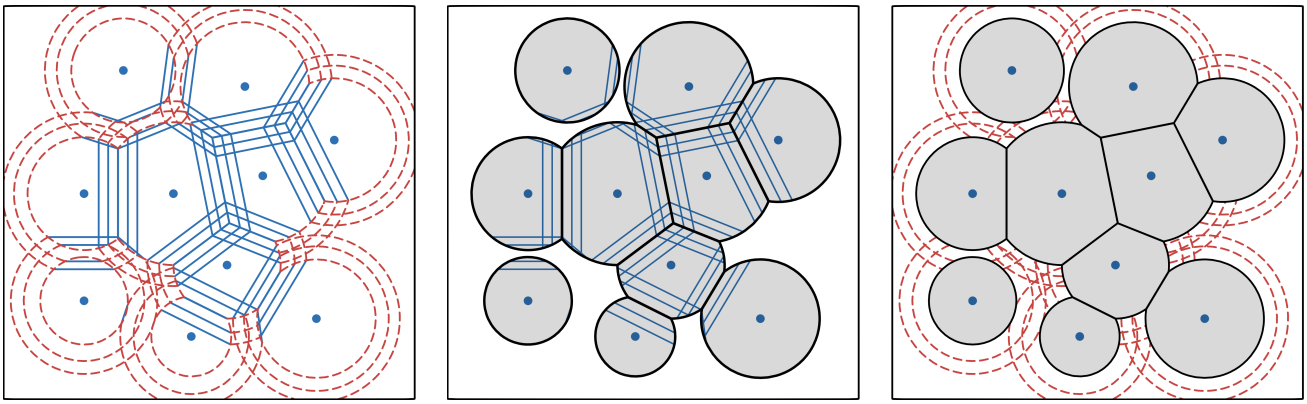
Size of residual graph. In this part, we show that the residual graph has $O(n^d)$ points and $O(n^{d+1})$ edges for d-dimensional distributions, which will be used in analyzing the running time of different steps of our algorithm.

Fix any set of weights $y(\cdot)$ for B. For any point $b \in B$, for simplicity in presentation, we omit the weights $y(\cdot)$ from notation and let $\text{ResVor}^0(b) := \text{ResVor}^\delta_y(b), \text{ResVor}^1(b) := \text{ResVor}^\delta_y(b)$, and $\text{ResVor}^2(b) := \text{ResVor}^{2\delta}_y(b)$ to denote the restricted Voronoi cell, δ-expanded, and 2δ-expanded Voronoi cells of b, respectively. For any cell $\text{ResVor}^i(b)$ for $b \in B$ and $i \in \{0,1,2\}$, let $\text{ResVor}^i_V(b)$ (resp. $\text{ResVor}^i_D(b)$) denote the part of the boundary of $\text{ResVor}^i(b)$ that is formed by the Voronoi cell $V^{i\delta}_y(b)$ (resp. disk with radius $y(b)+i\delta$ centered at b). We refer to this subset of boundaries as *Voronoi-based* (resp. *disk-based*) boundaries. See Figure A.3(a).

The following lemma provides an important property of these boundaries, which helps us in bounding the size of the residual graph.

Lemma A.6. *Suppose $\mathcal{R} := \bigcup_{b\in B} \text{ResVor}_y(b)$ denotes the union (of the interior) of all restricted Voronoi cells. For any point $b \in B$ and any $i \in \{1,2\}$, all interior points of the Voronoi-based boundaries $\text{ResVor}^i_V(b)$ lies inside $\mathcal{R}$ and all interior points of the disk-based boundaries $\text{ResVor}^i_D(b)$ lies outside $\mathcal{R}$.*

Proof. Consider any interior point $p \in \text{ResVor}^i_V(b)$. Since this point is determined by the



(a) (b) (c)

Figure A.3: (a) the Voronoi-based boundaries (blue solid lines) and disk-based boundaries (red dashed lines), (b) the subset of boundaries of the expansions that lie inside the restricted Voronoi cells, and (c) the subset of boundaries of the expansions that lies outside of all restricted Voronoi cells.

$i\delta$-expanded Voronoi cell of b, then $\mathrm{c}(p,b) < y(b) + i\delta$. Suppose p lies on a weighted bisector between b and another point $b' \in B$. In this case,

$$\mathrm{c}(p,b') - y(b') = \mathrm{c}(p,b) - (y(b) + i\delta) > 0.$$

Since b' is a weighted nearest neighbor of p and has a positive weighted distance to p, the point p lies inside the restricted Voronoi cell of b'. See Figure A.3(middle).

Similarly, consider any interior point $p \in \mathrm{ResVor}^i_D(b)$. By construction, $\mathrm{c}(p,b) = y(b) + i\delta$. Consider a weight function $y^i_b(\cdot)$ that assigns $y^i_b(b) := y(b) + i\delta$ and $y^i_b(b') := y(b')$ for all $b' \neq b$. By the definition of restricted Voronoi cells, the weighted nearest neighbor of p with weights $y^i_b(\cdot)$ is b, and any other point $b' \in B$ would have a non-positive weighted distance to p. Since the weighted distance of each point $b' \neq b$ to p is the same with respect to $y(\cdot)$ and $y^i_b(\cdot)$, the point p lies outside of the restricted Voronoi cell of b'. See Figure A.3(right). □

From Lemma A.6, any intersection point in the arrangement is formed using either d Voronoi-

based boundaries or d disk-based boundaries. As shown in Section 4.2.3, there are $O(n^d)$ intersection points between the Voronoi-based boundaries. Furthermore, any subset of d disks intersect in at most 2 points. Therefore, the total number of such intersection points is $O(\binom{n}{d}) = O(n^d)$. Hence, the number of intersection points, and consequently the number of regions in the arrangement is $O(n^d)$. Since the residual graph is a bipartite graph between set A_δ (with size $O(n^d)$) and set B (with size n), the number of its edges would be $O(n^{d+1})$, leading to the following lemma.

Lemma 5.4. *For d-dimensional distributions, for any $d \geq 2$, the residual graph has $O(n^d)$ nodes and $O(n^{d+1})$ edges.*

A.3.2 ACYCLIFY Procedure

Given a δ-restricted-feasible transport plan $\tau_\delta, y(\cdot)$ obtained after the execution of REDUCEWEIGHTS or INCREASEWEIGHTS procedure, the ACYCLIFY procedure updates the transport plan such that, for the implicit representation $\hat{\tau}_\delta$ of τ_δ, the transport plan is a forest, i.e., the graph containing all edges $(r, b) \in A_\delta \times B$ does not contain any edges. Furthermore, the ACYCLIFY procedure makes sure that there are no admissible alternating cycles in the residual graph. This procedure is identical to the ACYCLIFY procedure presented in Section 4.2.3. The procedure first uses a dynamic tree structure Sleator and Tarjan [149] to make the transport plan a forest (using phase 1). The procedure then uses partial DFS to follow the admissible triples and remove any admissible alternating cycles (using phase 2). The procedure finally makes the transport plan a forest again (using phase 1). We summarize the two main phases of the procedure next.

Phase 1: Acyclifying the Transport Plan. To make the transport plan a forest, the procedure initializes an empty transport plan $\hat{\tau}$ and iteratively adds the edges transporting

mass in $\hat{\tau}_\delta$ to $\hat{\tau}$. The procedure also maintains a dynamic tree data structure and for any new edge, if the edge creates a cycle of transportation, the Acyclify procedure cancels the cycle right away. More precisely, after adding an edge (r_1, b_1) to the transport plan, if a cycle $(b_1, r_1, b_2, \ldots, b_k, r_k)$ is formed such that $\hat{\tau}(r_i, b_i) > 0$ and $\hat{\tau}(b_i, r_{i-1}) > 0$ for all $i \in [1, k]$ (assuming $r_0 = r_k$), then the procedure increases (resp. decreases) the amount of mass transported on the even-indexed (resp. odd-indexed) edges of this cycle until at least one of the odd-indexed edges gets removed from the transport plan. Using the dynamic tree structure by Sleator and Tarjan [149], each of the above-mentioned operations takes $O(\log n)$ amortized time; therefore, using Lemma 5.4, since the number of edges of the residual graph is $O(n^{d+1})$, this phase takes a total of $O(n^{d+1} \log n)$ time.

Phase 2: Acyclifying the Admissible Triples. This step uses a partial DFS on the reverse of the residual graph $\overleftarrow{\mathcal{G}_\delta}$ to remove any admissible cycles from the residual graph. For any admissible cycle $C = \langle b_1, r_1, \ldots, b_i, r_i, b_{i+1} = b_1 \rangle$ in $\overleftarrow{\mathcal{G}_\delta}$, define the *bottleneck capacity* of C as $\min_{j \in [1,i]} \hat{\tau}(r_j, b_j)$. The procedure *cancels* C by increasing (resp. decreasing) the amount of mass transported along the forward (resp. backward) edges of C by the bottleneck capacity. The update will remove at least one of the transporting edges, hence removing the admissible cycle from the residual graph.

Mark all points of B and all backward edges as unvisited. Define $U := B$ as the set of unvisited points. For any unvisited point $b \in B$, initiate a partial DFS in the reversed residual graph $\overleftarrow{\mathcal{G}_\delta}$ by setting $Q = \langle b = b_1 \rangle$ and search as follows until Q becomes empty.

1. If $Q = \langle b_1, r_1, \ldots, b_i \rangle$,
   (a) If there exists an unvisited backward edge (b_i, r_φ), add r_φ as r_{i+1} to Q.
   (b) Otherwise, mark b_i as visited and remove b_i from Q and U.

2. If $Q = \langle b_1, r_1, \ldots, b_i, r_i \rangle$, let $b := \arg\min_{b' \in U} \mathrm{c}_y(r_i, b')$.

   (a) If $\mathrm{c}_y(r_i, b) < \mathrm{c}_y(r_i, b_i)$, i.e., (b, r_i, b_i) is admissible, if b lies on the search path Q as b_j, then the cycle $C = \langle b_j, r_j, \ldots, b_i, r_i, b_{i+1} = b = b_j \rangle$ is admissible. Cancel the cycle C and set $Q = \langle b_1, r_1, \ldots, b_j \rangle$. Otherwise, add b as b_{i+1} to Q.

   (b) Otherwise, (b, r_i, b_i) is not admissible. Remove r_i from Q and mark the backward edge (b_i, r_i) in $\overleftarrow{\mathcal{G}_\delta}$ as visited.

Lemma 5.7. *Given a δ-restricted-feasible transport plan $\tau_\delta, y(\cdot)$, after the execution of the* ACYCLIFY *procedure,*

(AC1) the transport plan $\hat{\tau}_\delta, y(\cdot)$ remains δ-restricted-feasible,

(AC2) the transport plan $\hat{\tau}_\delta$ is a forest and there are no admissible cycles in the residual graph, and

(AC3) if $\tau_\delta, y(\cdot)$ satisfy condition (F3) in Lemma 5.2, then condition (F3) remains satisfies.

Proof. We begin by proving (AC1). The ACYCLIFY procedure first makes the transportation graph acyclic. Since the edges carrying mass in the new transport plan is a subset of the original transportation network prior to applying ACYCLIFY , for each edge $(r, b) \in A_\delta \times B$ that transports mass in the new transport plan, $r \in \mathrm{ResVor}_y^{2\delta}(b)$, which implies that the transport plan produced after this phase remains δ-restricted-feasible. In the second phase of the procedure, at any time, all triples in the search path are admissible. Thus, updating the transport plan along any one of such cycles does not violate the δ-restricted-feasibility conditions.

Furthermore, since the amount of mass transported to each region remains the same at all steps of the procedure, any point $r \in A_\delta$ that is a deficit in the new transport plan is also a

deficit with respect to the transport plan prior to this phase of Acyclify procedure. Since the procedure does not change the weights of the points, if (F3) holds prior to the execution of the Acyclify procedure, any deficit region would have a zero weight, and condition (F3) continues to be true, i.e., (AC3) holds.

To argue that property (AC2) also holds, we aim to demonstrate:

(AC4) any point $b \in B$ (and any backward edge (r, b)) that is marked as visited does not participate in any admissible cycle in the same execution of the Acyclify procedure, and

(AC5) the portion of the residual graph consisting of visited points and their adjacent regions does not include any admissible cycles. Moreover, there are no admissible paths from an unvisited point to any visited point.

Assuming (AC4) is true, we conclude that no visited point $b \in B$ becomes part of an admissible cycle. Since the Acyclify algorithm terminates only when all elements of B have been marked as visited, this implies that no admissible cycles exist in the residual graph at termination. Additionally, since the procedure runs phase 1 once more after the execution of phase 2, the transportation network becomes acyclic, leading to (AC2).

Furthermore, as discussed next, property (AC4) naturally follows from (AC5): For any visited point $b \in B$, all points $b' \in B$ from which there is an admissible path to b must also have already been visited (since the algorithm performs the search in the reversed residual graph). If (AC5) holds, then no admissible cycle can be formed solely from visited nodes. Similarly, a backward edge (r, b) is marked as visited only if, for every admissible triple (b', r, b), the point b' has already been marked as visited. If b' does not participate in any admissible cycles, then neither can the triple (b', r, b). Accordingly, we prove (AC5) below, which proves (AC4) and consequently, proves (AC2).

We use induction to show that (AC5) holds throughout the second phase. Let $C^i, i \in [1, k]$ denote the sequence of admissible cycles computed 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 at the beginning of phase 2, and $\hat{\tau}_\delta^i$ is derived by updating $\hat{\tau}_\delta^{i-1}$ along C^i. Initially, at the start of the second phase, property (AC5) clearly holds.

A point $b \in B$ is marked as visited when a search from b does not lead to the formation of an admissible cycle. This means that for all admissible triples (b', r, b), the point b' and backward edge (r, b) must already be marked as visited. Consequently, all points that can reach b via an admissible path (all points that are reachable from b by our backward DFS) are also visited, and b has no admissible paths to any visited points in the residual graph. Therefore, if (AC5) holds prior to marking b as visited, it will remain valid afterward. We next consider the case where a cycle C^i is eliminated.

Any admissible cycle computed by the procedure is made up only of unvisited points of B. After canceling such a cycle, any newly created admissible triple (b, r, b') will consist of a visited point b and an unvisited point b'. This means that cycle cancellation can only produce new admissible paths from visited to unvisited points. Provided that (AC5) is held before canceling C^i, canceling C^i does not introduce any admissible cycles among visited points. Additionally, since cancellation does not generate any admissible triples directed from unvisited to visited points, there will be no admissible paths from any unvisited point to any visited point. Therefore, property (AC5) is preserved after canceling any cycle C^i. □

A.3.3 SEARCHANDAUGMENT Procedure

The SEARCHANDAUGMENT procedure initiates a partial DFS from deficit points of A_δ in the reverse of the residual graph to find admissible augmenting paths and admissible alternating

paths to an inactive point. The procedure then updates the transport plan along those paths. The SearchAndAugment procedure in this paper is similar to the SearchAndAugment procedure introduced in Section 4.2.1, and the only difference is that when the search path reaches an inactive point of B, i.e., a point $b \in B$ with $y(b) = \lambda$, the procedure updates the transport plan along the alternating path and restarts the search.

Let $\mathcal{D} \subset A_\delta$ denote the subset of deficit points of A_δ with respect to τ_δ. Mark all points of B and all backward edges as unvisited. Define $U := B$ as the set of unvisited points of B. For any point $r \in \mathcal{D}$, initiate a partial DFS $Q = \langle r = r_1 \rangle$ in the reverse residual graph $\overleftarrow{\mathcal{G}_\delta}$ as the search path that the procedure grows. Execute the following steps.

1. If $Q = \langle r = r_1, b_1, \ldots, r_i, b_i \rangle$,
   (a) If b_i is surplus, then the reverse path $P = \langle b_i, r_i, \ldots, r_1 \rangle$ is an admissible augmenting path in $\mathcal{G}_\delta$. Augment $\hat{\tau}_\delta$ along P. If r is still a deficit point, set $Q = \langle r = r_1 \rangle$. Otherwise, stop the current search.
   (b) Otherwise, if b_i is an inactive point (i.e., $y(b_i) = \lambda$), then the reverse path $P = \langle b_i, r_i, \ldots, r_1 \rangle$ is an admissible alternating path to an inactive point in $\mathcal{G}_\delta$. Update $\hat{\tau}_\delta$ along P. If r is still a deficit point, set $Q = \langle r = r_1 \rangle$. Otherwise, stop the current search.
   (c) Otherwise, if there is an unvisited backward edge (b_i, r') in $\overleftarrow{\mathcal{G}_\delta}$, then add r' as r_{i+1} to Q. Otherwise, mark b_i as visited and remove b_i from U and Q.
2. If $Q = \langle r = r_1, b_1, \ldots, b_i, r_{i+1} \rangle$, let $b := \arg\min_{b' \in U} d_y(r_{i+1}, b')$.
   (a) If $d_y(r_{i+1}, b) < d_y(r_{i+1}, b_i)$, i.e., (b, r_{i+1}, b_i) is admissible, add b as b_{i+1} to Q.
   (b) Otherwise, remove r_{i+1} from Q and mark the backward edge (b_i, r_{i+1}) as visited.

The procedure terminates when all partial DFS executions from all points in $\mathcal{D}$ terminate.

Lemma 5.8. *Given a δ-restricted-feasible transport plan $\tau_\delta, y(\cdot)$ satisfying condition (F3) in Lemma 5.2, after the execution of the* SEARCHANDAUGMENT *procedure,*

(SA1) the transport plan $\hat{\tau}_\delta, y(\cdot)$ remains δ-restricted-feasible and (F3) remains satisfies, and

(SA2) there are no admissible cycles, admissible augmenting paths, and admissible alternating paths from inactive points to deficit regions in the residual graph.

Proof. Suppose τ_δ (resp. τ'_δ) denotes the transport plan maintained by the algorithm after (resp. before) the execution of the SEARCHANDAUGMENT procedure.

For any edge $(r, b) \in A_\delta \times B$ with $\hat{\tau}_\delta(r, b) > 0$, if $\hat{\tau}'_\delta(r, b) > 0$, then by the δ-restricted-feasibility of $\hat{\tau}'_\delta, y(\cdot)$, the point r lies inside $\mathrm{ResVor}_y^{2\delta}(b)$. Otherwise, the edge (r, b) was a forward edge on an augmenting path P (resp. alternating path P' from an inactive point) so that after updating the transport plan along P (resp. P'), it has become a backward edge. In this case, there is a forward edge from b to r in the residual graph, which means that $r \in \mathrm{ResVor}_y^{2\delta}(b)$, i.e., $\hat{\tau}_\delta, y(\cdot)$ is also δ-restricted-feasible. Furthermore, since the SEARCHANDAUGMENT procedure does not change the weights, (F3) continues to be true, and (SA1) holds.

Suppose $P^1, \ldots, P^k$ denotes the sequence of admissible augmenting paths and alternating paths to inactive points computed by the SEARCHANDAUGMENT procedure, and for any $i \in [0, k]$, let $\hat{\tau}_\delta^i$ and $\mathcal{G}_\delta^i$ denote the transport plan and residual graph maintained after updating the transport plan along P^i, respectively. Define V^i (resp. U^i) to be the set of visited (resp. unvisited) points of B with respect to $\hat{\tau}_\delta^i$.

We use an induction on i to show that there are no admissible cycles in the residual graph. Since $\hat{\tau}_\delta^0$ is obtained from the transport plan computed by the ACYCLIFY procedure, there are no admissible cycles in the residual graph $\mathcal{G}_\delta^0$. For any $i \in [1, k]$, assuming there are no

admissible cycles in $\mathcal{G}_\delta^0, \ldots, \mathcal{G}_\delta^{i-1}$, we show that the same holds for $\mathcal{G}^i$. For any admissible triple (b, r, b') in $\mathcal{G}^i$ formed after updating $\hat{\tau}_\delta^{i-1}$ along P^i, we show below that the triple (b, r, b') does not form admissible cycles and conclude that there are no admissible cycles in $\mathcal{G}_\delta^i$.

For any visited point $b \in V^{i-1} \setminus V^{i-2}$, the search from b did not lead to the computation of an augmenting path or an alternating path to an inactive point; therefore, any point b' that is reachable from b by an admissible path in the reverse of residual graph $\overleftarrow{\mathcal{G}^{i-1}}$ (and therefore, is reachable by our partial DFS procedure from b) would have also been marked as visited. In other words, 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 U^{i-1} to any visited point in V^{i-1}.

Below, we show that for any newly formed admissible triple (b, r, b'), $b \in V^{i-1}$ and $b' \in U^{i-1}$. Assuming this, there are no admissible triples from any visited point to any unvisited point (in the reversed residual graph), and therefore, the newly formed admissible triples (which are from an unvisited point to a visited point in the reversed residual graph) do not form any admissible cycles, as claimed. Consider any triple (b, r, b') that is admissible in $\mathcal{G}_\delta^i$ but not in $\mathcal{G}_\delta^{i-1}$. Recall that the SearchAndAugment procedure does not change the weights of points, and therefore, since (b, r, b') is not admissible in $\mathcal{G}_\delta^{i-1}$, (r, b') is not a backward edge in $\mathcal{G}_\delta^{i-1}$, i.e., $(r, b') \in P^i$ as a forward edge. From step 2 of the SearchAndAugment procedure, b' is the weighted nearest unvisited neighbor of r; therefore, since $\mathrm{c}_y(r, b) > \mathrm{c}_y(r, b')$ (from the admissibility of (b, 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'), $b \in V^{i-1}$ and $b' \in U^{i-1}$. To summarize, any newly formed admissible triple is from a visited point to an unvisited point (in the residual graph), while there are no admissible triples from an unvisited point to a visited points; hence, the newly formed

admissible triples do not form admissible cycles, and using the inductive hypothesis, the residual graph remains free of any admissible cycles.

To prove property (SA3), we next next the following properties:

(SA4) any point $b \in B$ and any backward edge that is marked as visited does not form any admissible augmenting paths or alternating paths to inactive points in the same execution of the SEARCHANDAUGMENT procedure.

For any visited point $b \in V^i$, as discussed above, all vertices that are reachable from b by an admissible path in the reversed residual graph are also visited. Since all inactive points $b_s \in B$ (resp. points $b_z \in B$ with zero-weight points $r \in A_\delta$ in $\text{ResVor}_y^{2\delta}(b)$) are unvisited, b_s (resp. b_z) is not reachable from any visited point $b \in B$ by our search algorithm and therefore, the visited points do not participate in an admissible augmenting path or alternating path to inactive points. 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 or alternating path to inactive points, the visited backward edge (r, b) also does not form an admissible augmenting path or alternating path to inactive points.

For any deficit point $r \in \mathcal{D}$ with respect to $\hat{\tau}_\delta, y(\cdot)$, the search from r has been terminated since Q was empty (otherwise, r would have been balanced); hence, all forward edges incident on r are to visited points of B, which from (SA4) do not form admissible augmenting paths or alternating paths to inactive points; therefore, r will not form any admissible augmenting paths or alternating paths to inactive points during the same execution of the SEARCHANDCONSOLIDATE procedure, leading to (SA2). □

A.3.4 IncreaseWeights Procedure

The IncreaseWeights procedure is also similar to the IncreaseWeights procedure introduced in Section 4.2.2, with the slight difference that it inactivates the points whose weights reach λ after increasing their weights. In particular, the IncreaseWeights procedure computes a set $\mathcal{K}$ of points that are reachable from the surplus points by admissible alternating paths and increases the weights of those points by δ. It then marks all points of B whose weights have reached λ as inactive and recomputes A_δ, $\mathcal{G}_\delta$, and $\hat{\tau}_\delta$. For completeness, we include the description of the IncreaseWeights procedure below.

For each point $r \in A_\delta$, let $\mathcal{N}(r) \subseteq B$ denote the set of points $b \in B$ with $\hat{\tau}_\delta(r, b) > 0$. Let $\mathcal{S} \subset B$ denote the set of active surplus points of B. Mark all points $b \in B$ and all forward edges (b, r) in the residual graph as unvisited and set $\mathcal{K} = \emptyset$. Let $U = B$ denote the set of unvisited points of B. Initiate a partial DFS from any point $b \in \mathcal{S}$ in the residual graph $\mathcal{G}_\delta$ by setting $Q := \langle b = b_0 \rangle$ as the search path that the algorithm grows. Execute the following steps.

1. If $Q = \langle b = b_0, r_1, \ldots, b_i \rangle$,
   (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 and remove b_i from U and Q. Add b_i to $\mathcal{K}$.
2. If $Q = \langle b = b_0, r_1, \ldots, b_i, r_i \rangle$, let $b^* := \arg\max_{b' \in U \cap \mathcal{N}(r_i)} \mathrm{c}_y(r_i, b')$.
   (a) If $\mathrm{c}_y(r_i, b^*) > \mathrm{c}_y(r_i, b_i)$, i.e., (b_i, r_i, b^*) is admissible, add b^* as b_{i+1} to Q.
   (b) Otherwise, remove r_i from Q and mark (b_i, r_i) as visited.

After all DFS procedures from all points in $\mathcal{S}$ terminate, for each point $b \in \mathcal{K}$, set $y(b) \leftarrow y(b) + \delta$. We next describe how to recompute the residual graph and the compressed transport

plan with respect to the updated weights.

Lemma 5.9. *Given a δ-restricted-feasible transport plan $\tau_\delta, y(\cdot)$ satisfying condition (F3) in Lemma 5.2, after the execution of the* IncreaseWeights *procedure,*

(IW1) the transport plan $\hat{\tau}_\delta, y(\cdot)$ remain δ-restricted-feasible and condition (F3) remains satisfies,

(IW2) the weight of each active surplus point $b \in B$ increases by δ, and

(IW3) the weights of all points $b \in B$ with deficit regions inside $\mathrm{ResVor}_y^{2\delta}(b)$ and all inactive points remain unchanged.

Proof. Let $y(\cdot), \mathcal{X}_\delta, A_\delta$ (resp. $y'(\cdot), \mathcal{X}'_\delta, A'_\delta$) denote the set of weights of B, decomposition of A, and the set of representative points before (resp. after) the execution of the IncreaseWeights procedure.

To prove property (IW1), we show that for any pair of points $(a, b) \in A \times B$ such that $\tau_\delta(a, b) > 0$, we have $a \in \mathrm{ResVor}_{y'}^{2\delta}(b)$. Let b_a (resp. b'_a) denote the weighted nearest neighbor of a with respect to weights $y(\cdot)$ (resp. $y'(\cdot)$). From the δ-restricted-feasibility of $\tau_\delta, y(\cdot)$, we have $\mathrm{c}_y(a, b) \le \mathrm{c}_y(a, b'_a) + 2\delta$. Let r_a denote the representative point of the region containing a.

- If $b \in \mathcal{K}$, then b_a also is included in $\mathcal{K}$, since either $b = b_a$ or the triple (b_a, r_a, b) would be admissible. Hence, b_a remains the weighted nearest neighbor of a with respect to weights b'_a, and

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

- Otherwise, $b \notin \mathcal{K}$. In this case,

– if $b'_a \notin \mathcal{K}$, then

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

– otherwise, we would have $\mathrm{c}_y(r_a,b'_a) \geq \mathrm{c}_y(r_a,b)$, since otherwise (b'_a,r_a,b) would be admissible and since $b'_a \in \mathcal{K}$, then $b \in \mathcal{K}$ as well. In this case, since b'_a became the weighted nearest neighbor of r_a, we should have $r_a \in \mathrm{ResVor}_y^{\delta}(b'_a)$, and since $\mathrm{c}_y(r_a,b'_a) \geq \mathrm{c}_y(r_a,b)$, we should also have that $r_a \in \mathrm{ResVor}_y^{\delta}(b)$ and consequently $a \in \mathrm{ResVor}_y^{\delta}(b)$. Then,

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

Therefore, for any pair (a,b) with $\tau_\delta(a,b) > 0$, we have $a \in \mathrm{ResVor}_y^{2\delta}(b)$, and $\tau_\delta, y'(\cdot)$ would be δ-restricted-feasible, i.e., (IW1) holds.

Note that the IncreaseWeights procedure initiates a partial DFS from each active surplus point, which terminates when the search path becomes empty; hence, each active surplus point $b \in B$ will be added to the set $\mathcal{K}$ in step 1(a) of the procedure, and its weight will be increased by δ, proving (IW2).

Finally, note that from Lemma 5.9, there are no admissible augmenting paths or alternating paths from active surplus points to inactive points in the residual graph. Hence, the partial DFS procedure from the active surplus points cannot reach any inactive point or any point $b \in B$ with deficit regions inside $\mathrm{ResVor}_y^{2\delta}(b)$; therefore, all such points would not be in the set $\mathcal{K}$ and their weights remain unchanged, leading to (IW3).

□

A.3.5 Missing Proofs of Efficiency

In this section, we provide the efficiency analysis of each procedure used in our algorithm. Our analysis is similar to the analysis presented in Section 6.3.2; however, for completeness, we provide the analysis in detail for d-dimensional space.

Efficiency of the SEARCHANDCONSOLIDATE and SEARCHANDAUGMENT Procedures

Each execution of the SEARCHANDCONSOLIDATE (resp. SEARCHANDAUGMENT) procedure runs partial DFS procedures on the residual graph to find a set of admissible augmenting and consolidating (resp. alternating and 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 add it to the search path again in the same execution. Upon finding an admissible path P, the procedure updates the transport plan along P in $O(|P|)$ time. Let $\{P_1, \ldots, P_k\}$ denote the set of all paths found by the SEARCHANDCONSOLIDATE (resp. SEARCHANDAUGMENT) procedure. Since the residual graph has $O(n^{d+1}$ edges (from Lemma 5.4), the running time of the procedure would be $O(n^{d+1} + \sum_{i=1}^{k} |P_i|)$. Next, we bound the total length of all paths found by the procedures.

Lemma A.7. *The total length of augmenting and consolidating paths computed during the execution of the* SEARCHANDCONSOLIDATE *procedure is* $O(n^{d+1})$ *in* d *dimensions.*

Proof. Let $\hat{\tau}_\delta^0$ denote the transport plan maintained by the algorithm at the beginning of the execution of the SEARCHANDCONSOLIDATE procedure. To prove this lemma, we categorize the augmenting and consolidating paths found by the procedure based on the source of their bottleneck capacity, namely (1) set $\mathcal{P}_v$ consisting of augmenting and consolidating paths whose bottleneck capacity is determined based on the residual capacity of its endpoints, and (2) set $\mathcal{P}_e$ consisting of paths whose bottleneck capacity is determined based on mass

transportation over its backward edges. We begin by showing that $|\mathcal{P}_v| = O(n^d)$ and then show $|\mathcal{P}_e| = O(n^d)$. Since each path has a length of at most $2n$ (since the number of points in B is n), we then conclude that the total length of all paths computed by the procedure is $O(n^3)$.

Let P be a consolidating path in $\mathcal{P}_v$. If the bottleneck capacity of P is determined by its free endpoint $r \in A_\delta$, then the mass of r will be fully transported after updating the transport plan along P; similarly, for any augmenting path $P \in \mathcal{P}_v$, if the bottleneck capacity of P is determined by its free endpoint $b \in B$ (resp. $r \in A_\delta$), then b (resp. r) will be balanced after augmentation. Therefore, $|\mathcal{P}_v| = O(n^2)$ since $|A_\delta \cup B| = O(n^2)$. Next, let P be a 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.

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 $\mathrm{c}_y(r, b) > \mathrm{c}_y(r, b')$. Since the SearchAndConsolidate 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 updating $\hat{\tau}_\delta$ along P results in transporting mass from b' to r. On the other hand, by step 2(b) of the SearchAndConsolidate 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 $\mathrm{c}_y(r, b) > \mathrm{c}_y(r, b')$ 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).

By property (SC2) in Lemma 5.5, the point b' cannot form an admissible augmenting path during the same execution of the SearchAndConsolidate procedure. Hence, the edge (r, b) determining the bottleneck capacity of P was a backward edge of the initial transport plan $\hat{\tau}^0_\delta$ and updating the transport plan along each path $P \in \mathcal{P}_e$ removes one of the transporting

edges of the transport plan $\hat{\tau}_\delta^0$. Since the transport $\hat{\tau}_\delta^0$ is obtained from the ACYCLIFY procedure, using property (AC2) in Lemma 5.7, the transport plan is a forest and the number of backward edges is $O(n^d)$; hence, $|\mathcal{P}_e| = O(n^d)$, as claimed. The total number of augmenting paths found by the procedure, therefore, is $O(n^d)$, and since each augmenting path has a length of at most $2n$, their total length is $O(n^{d+1})$. □

Using an identical argument, one can show that the total length of all alternating and augmenting paths found by the SEARCHANDAUGMENT procedure is $O(n^{d+1})$. Therefore, each execution of the SEARCHANDCONSOLIDATE and SEARCHANDAUGMENT procedures takes $O(n^{d+1})$ time.

Efficiency of the REDUCEWEIGHTS and INCREASEWEIGHTS Procedures

The REDUCEWEIGHTS and INCREASEWEIGHTS procedures run a DFS that visits each edge of the residual graph at most once and has a total running time of $O(n^{d+1})$ d dimensions. Furthermore, in the arrangement used to construct partitioning $\mathcal{Y}$, each point $b \in B$ has at most 6 restricted Voronoi cells (three cells that are used in the construction of $\mathcal{X}_\delta$ and three that are used in the construction of $\mathcal{X}'_\delta$). Similar to our proof for the size of the graph, one can show that the total number of vertices in the arrangement used to construct $\mathcal{Y}$ is $O(n^d)$, and the number of regions in $\mathcal{Y}$ is at most $O(n^d)$. The construction of the transport plan $\hat{\tau}$, therefore, can be done in $O(n^d(\mathrm{Q} + n))$ time since

(1) the mass of all regions in $\mathcal{Y}$ can be determined using the ORACLE in $O(n^d\mathrm{Q})$ time, 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 $O(n^{d+1})$ time, since there are $O(n^{d+1})$ pairs of points in $A_\delta \times B$, and therefore, the total complexity of $\hat{\tau}$ is $O(n^{d+1})$.

Finally, storing a sorted list of neighbors for each region $r \in A_\delta$ takes $O(n^{d+1}) \log n$ time in total. Hence, the executions of the ReduceWeights and IncreaseWeights procedures take $O(n^d(\mathrm{Q} + n \log n))$ time.

Efficiency of the Acyclify Procedure

In the first step, the Acyclify procedure uses a dynamic tree structure to remove all cycles of transportation from the transport plan $\hat{\tau}_\delta$. Using the dynamic tree structure by Sleator and Tarjan [149], since the total number of edges of the graph is $O(n^{d+1})$ in d dimensions, the running time of this step would be $O(n^{d+1} \log n)$. 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 add it again to the search path in the same execution. Furthermore, upon finding an admissible cycle C, it cancels the cycle in $O(|C|)$ time. Let $\{C_1, \ldots, C_k\}$ denote the set of all cycles found in the execution of the Acyclify procedure. Given that the size of the residual graph is at most $O(n^{d+1})$, the second step of the Acyclify procedure takes a total of $O(n^{d+1} + \sum_{i=1}^{k} |C_i|)$ time. The following lemma bounds the total length of all cycles found by the Acyclify procedure.

Lemma A.8. *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 after the first step of the Acyclify procedure, i.e., $\hat{\tau}_\delta^0$ is a forest. To prove this lemma, we show that the Acyclify procedure finds $O(n^d)$ 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 $O(n^{d+1})$.

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 A.7, 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 5.7, 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 $\hat{\tau}_\delta^0$, where $\hat{\tau}_\delta^0$ is a forest and the number of its transporting edges is $O(n^d)$. Therefore, the total number of cycles found by the ACYCLIFY procedure is $O(n^d)$, and their total length is $O(n^{d+1})$, as claimed. □

From Lemma A.8, the total execution time of the ACYCLIFY procedure is $O(n^{d+1})$ in d dimensions.

Appendix B

Bipartite Matching

Lemma 1.1. *Suppose M is a perfect matching and $y(\cdot)$ is a set of dual weights for $A \cup B$ where $M, y(\cdot)$ is feasible. Then, M is a minimum-cost perfect matching.*

Proof. Using Equation (1.12), we rewrite the cost of the matching M as follows.

$$w(M) := \sum_{(a,b)\in M} \mathrm{c}(a,b) = \sum_{(a,b)\in M} y(b) - y(a) = \sum_{b\in B} y(b) - \sum_{a\in A} y(a). \tag{B.1}$$

Suppose M^* is any minimum-cost perfect matching between A and B. Using Equation (1.11), we rewrite the cost of M^* as follows.

$$w(M^*) := \sum_{(a^*,b^*)\in M^*} \mathrm{c}(a^*,b^*) \geq \sum_{(a^*,b^*)\in M^*} y(b^*) - y(a^*) = \sum_{b^*\in B} y(b^*) - \sum_{a^*\in A} y(a^*). \tag{B.2}$$

Combining Equations (B.1) and (B.2), we have

$$w(M) = \sum_{b\in B} y(b) - \sum_{a\in A} y(a) \leq w(M^*).$$

Since M^* is a minimum-cost perfect matching and M is some perfect matching between A and B whose cost is no more than the cost of M^*, we conclude that M is also a minimum-cost perfect matching between A and B, as claimed. □

Lemma 1.2. *For any feasible matching $M, y(\cdot)$ and any augmenting path P between a free*

point $a \in A$ and a free point $b \in B$, $\phi(P) = y(b) - y(a) + \sum_{(a',b') \in P} s(a',b')$.

Proof. Recall that the slack of any edge (a,b) is defined as $s(a,b) = \mathrm{c}(a,b) - y(b) + y(a)$, and from Equations (1.12), the slack of any matching edge is zero. We rewrite the net-cost of an augmenting path P from a free point $b \in B$ to a free point $a \in A$ as follows.

$$\begin{aligned}
\phi(P) &:= \sum_{(a',b') \in P \setminus M} \mathrm{c}(a',b') + \sum_{(a',b') \in P \cap M} -\mathrm{c}(a',b') \\
&= y(b) - y(a) + \sum_{(a',b') \in P \setminus M} \big(\mathrm{c}(a',b') - y(b') + y(a')\big) + \sum_{(a',b') \in P \cap M} \big(- \mathrm{c}(a',b') + y(b') - y(a')\big) \qquad \text{(B.3)} \\
&= y(b) - y(a) + \sum_{(a',b') \in P \setminus M} s(a',b') + \sum_{(a',b') \in P \cap M} 0 \\
&= y(b) - y(a) + \sum_{(a',b') \in P \setminus M} s(a',b') + \sum_{(a',b') \in P \cap M} s(a',b') \\
&= y(b) - y(a) + \sum_{(a',b') \in P} s(a',b').
\end{aligned}$$

Equation (B.3) is resulted since the endpoints $b \in B$ and $a \in A$ of the augmenting path P are incident on exactly one non-matching edge and all other points along the path P are incident on exactly one matching edge and one non-matching edge (and hence, their dual weight is added once and subtracted once in the summation). □

B.1 Missing Details and Proofs of Chapter 6

B.1.1 Constructing the hierarchical partitioning

Recall that, as described in Section 6.1.1, our algorithm constructs a hierarchical partitioning by recursively partitioning each non-leaf cell of $\mathcal{H}$ into two smaller cells as its children. We

describe the partitioning procedure with respect to an arbitrary cell $\square$ of $\mathcal{H}$. Let ℓ_x and ℓ_y denote the width and length of $\square$, and let $x_{\min}$ (resp. $x_{\max}$) denote the x coordinate of the left (resp. right) side of $\square$. Without loss of generality, assume $\ell_x \geq \ell_y$. Recall that $\lambda = 9n^{-1/5}$. Define a priority queue $\mathcal{Q}$ storing two events for each point $u \in A_\square \cup B_\square$, namely an entry event $e_u^\downarrow$ at $u_x - \ell_\square\lambda$ and an exit event $e_u^\uparrow$ at $u_x + \ell_\square\lambda$; here, u_x denotes the x coordinate of point u. See the events $b^\downarrow$ and $b^\uparrow$ in Figure B.1.

Consider a vertical sweep line that moves from left to right, searching for a value x^* in the middle third of the x side of $\square$ minimizing $|\Lambda(x^*)|$; recall that

$$\Lambda(x) := \{u \in A_\square \cup B_\square : |u_x - x| \leq \ell_\square\lambda\}.$$

Let x' denote the state of the sweep line. Intuitively, as the sweep-line moves from left to right (i.e., x' increases), any point $u \in A_\square \cup B_\square$ enters the set $\Lambda(x')$ at its entry event and exists $\Lambda(x')$ at its exit event. Thus, we can keep track of the number of points close to the sweep line by incrementing (resp. decrementing) the count at each entry (resp. exit) event and return the value x^* realizing the minimum count during the sweep-line procedure.

More formally, we maintain a value γ representing the number of points of $A_\square \cup B_\square$ at a distance at most $\ell_\square\lambda$ to the sweep line. We also store, for a set of $O(n_\square)$ values $x'' \in [x_{\min}, x_{\max}]$, a function $\eta : [x_{\min}, x_{\max}] \to \mathbb{Z}_{\geq 0}$ representing $|\Lambda(x'')|$. Initially, for the minimum event e in $\mathcal{Q}$, set $\gamma \leftarrow 1$ and $\eta(e) \leftarrow \gamma$. Iteratively, our sweep-line algorithm extracts the minimum event e from $\mathcal{Q}$. If e is an entry event, set $\gamma \leftarrow \gamma + 1$, and otherwise, e is an exit event and set $\gamma \leftarrow \gamma - 1$. Set $\eta(e) \leftarrow \gamma$. After processing all events in $\mathcal{Q}$, define the entry x' of $\eta(\cdot)$ in the middle third of x side of $\square$ with the minimum value, i.e.,

$$x^* := \arg \min_{x \in [x_{\min} + \frac{\ell_\square}{3}, x_{\min} + \frac{2\ell_\square}{3}] \cap S(\eta)} \eta(x),$$

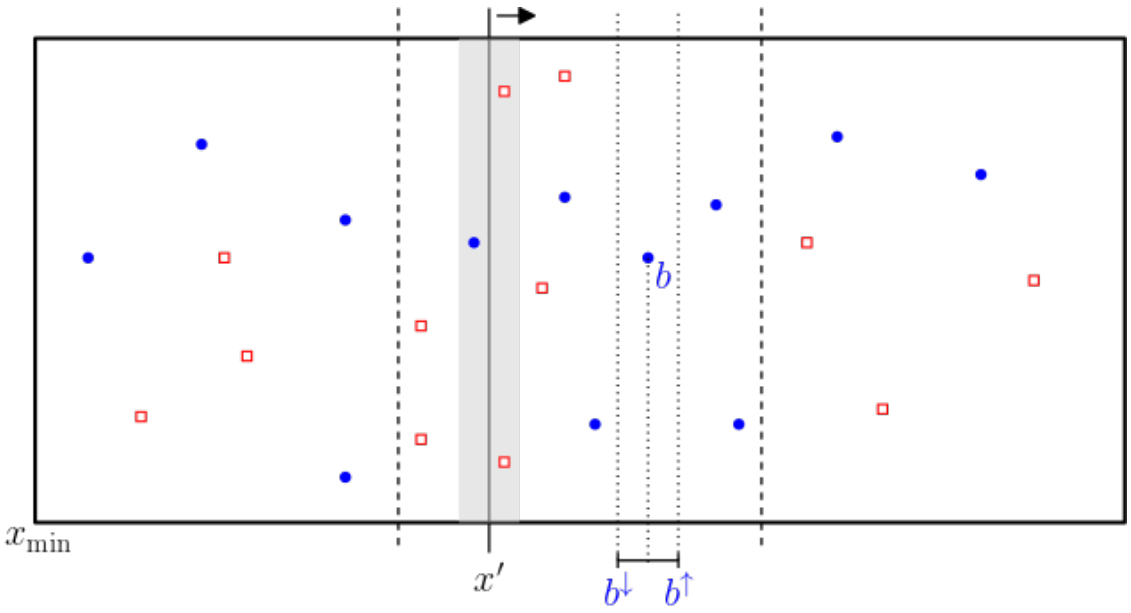


Figure B.1: The solid vertical line shows the line our sweep-line procedure maintains. The two values $b^{\downarrow}$ and $b^{\uparrow}$ show the entry and exit events created for the blue point b.

where $S(\eta)$ denotes the support of η. Partition $\square$ using the vertical line $x = x^*$ into two smaller cells $\square_1$ and $\square_2$, and add the non-empty ones as the children of $\square$ to $\mathcal{H}$.

B.1.2 Analysis for General d and q

For a point set U of $2n$ points in d dimensions and any $q \geq 1$, let A denote a random subset of n points of U and let $B = U \setminus A$. To compute a minimum-cost perfect matching between A and B under ℓ_2^q distances, we construct our hierarchical partitioning $\mathcal{H}$ with a parameter $\lambda = 9n^{-\frac{1}{d+2}}$ and execute our algorithm from Section 6.2. In the following, we extend our analysis from Section 6.3.2 to any dimension $d \geq 2$ and any $q \geq 1$ and show the following result.

Lemma B.1. *Suppose U is a set of $2n$ points inside the unit d-dimensional hypercube, $d \geq 2$, and A is a subset chosen uniformly at random from all subsets of size n. Let $B = U \setminus A$. Then, for any parameters $q \geq 1$, the expected running time of our algorithm for computing*

the minimum-cost matching on the complete bipartite graph on A and B under ℓ_2^q costs is

$$\begin{cases} \tilde{O}(n^{2-\frac{q}{(q+1)d}}\Phi(n)\log\Delta), & q \le \frac{d}{2}, \\ \tilde{O}(n^{2-\frac{1}{d+2}}\Phi(n)\log\Delta), & q > \frac{d}{2}. \end{cases}$$

We begin by showing that there exists a low-cost high-cardinality partial matching.

Lemma B.2. *For any cell $\square$ of $\mathcal{H}$, there exists a matching M' that, in expectation, matches all except $O\left(n_\square^{1-\frac{1}{d+2}}\right)$ points of $B_\square$ and has a cost $O\left((2\ell_\square)^q n_\square^{1-\frac{q+1}{d+2}}\right)$.*

Proof. Define $r := \frac{2q}{d} > 1$ and $t := \lceil \log_r \log_2 n \rceil$. Let $\beta := q(d+2)r^t - d(q+1)$. Define a sequence of grids $\langle \mathbb{G}_1, \ldots, \mathbb{G}_t \rangle$, where each grid $\mathbb{G}_i$ has a side-length $O\left(\ell_\square n^{-\alpha_i}\right)$ for

$$\alpha_i := \frac{1}{d} - \frac{r^{t+1}}{\beta}\left(1 - \frac{1}{r^i}\right).$$

We construct a matching M as follows. Let $A_\square^0 := A_\square$ and $B_\square^0 := B_\square$. Starting from $i = 1$, let M_i be a matching from $B_\square^{i-1}$ to $A_\square^{i-1}$ that matches as many points as possible inside each hypercube of $\mathbb{G}_i$. Let $A_\square^i$ and $B_\square^i$ denote the set of free points of $A_\square^{i-1}$ and $B_\square^{i-1}$, respectively. Set $i \leftarrow i+1$ and continue this procedure until the last grid $\mathbb{G}_t$ is processed. Define $M := \sum_{i=1}^t M_i$. In the following, we show that the total number of free points with respect to M is $\tilde{O}\left(n_\square^{1-\frac{1}{d+2}}\right)$ and the cost of M is $\mathbb{E}[w(M)] = O\left((2\ell_\square)^q n_\square^{1-\frac{q+1}{d+2}}\right)$.

The matching M matches as many points as possible inside each square of the grid $\mathbb{G}_t$. Therefore, the total number of unmatched points is equal to the excess of $\mathbb{G}_t$, which by Lemma B.3 below is

$$\mathbb{E}[\mathrm{exc}(\mathbb{G}_t)] = \tilde{O}\left(n_\square^{\frac{d}{2}\alpha_t + \frac{1}{2}}\right) = \tilde{O}\left(n_\square^{1-\frac{1}{d+2}}\right), \tag{B.4}$$

where the second equality holds since

$$\frac{d}{2}\alpha_t + \frac{1}{2} = \frac{d}{2}\left(\frac{1}{d} - \frac{r^{t+1}}{\beta}(1-\frac{1}{r^t})\right) + \frac{1}{2} = \left(\frac{1}{2} - \frac{qr^t - q}{q(d+2)r^t - d(q+1)}\right) + \frac{1}{2}$$
$$= 1 - \frac{1}{d+2} + \frac{q - \frac{d(q+1)}{d+2}}{q(d+2)r^t - d(q+1)} \leq 1 - \frac{1}{d+2} + \frac{1}{2\log n}.$$

We next analyze the expected cost of M. For $i = 1$, since all matching edges in M_1 has a cost of at most $O((\ell_\square n_\square^{-\alpha_1})^q)$, the cost of M_1 would be

$$w(M_1) = O\left(n_\square \times (\ell_\square n_\square^{-\alpha_1})^q\right) = O\left((2\ell_\square)^q n_\square^{1-\frac{q+1}{d+2}}\right), \tag{B.5}$$

where the second equality holds since

$$1 - q\alpha_1 = 1 - \frac{q}{d} + \frac{q(r-1)r^t}{\beta} = 1 - \frac{q}{d} + \frac{\frac{2q^2-qd}{d}r^t}{q(d+2)r^t - d(q+1)}$$
$$= 1 - \frac{q}{d} + \frac{2q-d}{d(d+2)} + \frac{\frac{(2q-d)(q+1)}{d+2}}{q(d+2)r^t - d(q+1)} \leq 1 - \frac{q+1}{d+2} + \frac{q}{d\log n}.$$

Finally, for each $1 < i \leq t$, the matching M_i consists of matching edges with a cost of $O\left((\ell_\square n_\square^{-\alpha_i})^q\right)$. By Lemma B.3, the expected number of matching edges in M_i is at most $\tilde{O}\left(n_\square^{\frac{d}{2}\alpha_{i-1}+\frac{1}{2}}\right)$; therefore,

$$\mathbb{E}[w(M_i)] = \tilde{O}\left(n_\square^{\frac{d}{2}\alpha_{i-1}+\frac{1}{2}} \times (\ell_\square n_\square^{-\alpha_i})^q\right) = \tilde{O}\left((2\ell_\square)^q n_\square^{1-\frac{q+1}{d+2}}\right), \tag{B.6}$$

where the second equality is resulted from as follows.

$$\frac{d}{2}\alpha_{i-1} + \frac{1}{2} - q\alpha_i = \left(\frac{1}{2} - \frac{qr^t(1-\frac{1}{r^{i-1}})}{\beta}\right) + \frac{1}{2} - \left(\frac{q}{d} - \frac{qr^{t+1}(1-\frac{1}{r^i})}{\beta}\right)$$
$$= 1 - \frac{q}{d} + \frac{q(r-1)r^t}{\beta} \leq 1 - \frac{q+1}{d+2} + \frac{q}{d\log n}.$$

Combining Equations (B.5) and (B.6), the expected cost of M is

$$\mathbb{E}[w(M)] = \sum_{i=1}^{t} \mathbb{E}[w(M_i)] = O\left((2\ell_\square)^q n_\square^{1-\frac{q+1}{d+2}}\right).$$

□

Lemma B.3. *For any random partitioning of a set U of $2n$ points inside the d-dimensional unit hypercube into two sets A and B of n points each, a hypercube $\square$ of $\mathcal{H}$, and a grid $\mathbb{G}$ inside $\square$ with cell side length $O\left(\ell_\square n_\square^{-\alpha}\right)$, $\mathbb{E}[\text{exc}(\mathbb{G})] = \tilde{O}\left(n_\square^{\frac{d}{2}\alpha+\frac{1}{2}}\right)$.*

Proof. For $\alpha \geq \frac{1}{d}$, the lemma statement holds trivially since $n_\square^{\frac{d}{2}\alpha+\frac{1}{2}} \geq n_\square$. Therefore, we assume $\alpha \leq \frac{1}{d}$. Using the Hoeffding's inequality [84], for any hypercube ξ of G,

$$\Pr\left[|B_\square^\xi| - |A_\square^\xi| \geq c_1\sqrt{n_\square log n}\right] \leq n^{-c_2}$$

for some constants $c_1, c_2 > 1$. Since $\text{exc}(\xi) \leq n_\square$,

$$\mathbb{E}[\text{exc}(\xi)] = O(\sqrt{n_\square \log n}).$$

Summing over all squares of G,

$$\mathbb{E}[\text{exc}(G)] = \sum_{\xi\in G} \mathbb{E}[\text{exc}(\xi)] = O\left(\sum_{\xi\in G}\sqrt{n_\square \log n}\right) = O\left(\sqrt{n \log n \times |G|}\right) = \tilde{O}\left(n^{\frac{d}{2}\alpha+\frac{1}{2}}\right).$$

□

Recall that for any cell $\square$ in $\mathcal{H}$, the points of B that are matched to the divider of $\square$ when our algorithm erases $\Gamma_\square$ is denoted by $B_\square^{\mathcal{C}}$, and that while erasing the divider $\Gamma_\square$, we showed that our algorithm executes the ExtendedHungarianSearch procedure $|B_\square^{\mathcal{C}}|$ times.

We use Lemma B.2 to show that the number of points in $B^{\mathcal{C}}_{\square}$ is $\tilde{O}\left(n^{1-\frac{1}{d+2}}\right)$. Let M' denote the matching constructed in Lemma B.2, and let M denote the matching of the extended matching $M^{\mathcal{C}}$ maintained by our algorithm. Let $M_{\square}$ denote the matching edges of M that lie inside $\square$. Let $\mathcal{P}_{\text{aug}}$ (resp. $\mathcal{P}_{\text{alt}}$) denote the set of augmenting paths (resp. alternating paths) with an endpoint in $|B^{\mathcal{C}}_{\square}|$ in the symmetric difference $M_{\square} \oplus M'$. As discussed above, $|\mathcal{P}_{\text{alt}}| \leq |F(M')| = O\left(n^{1-\frac{1}{d+2}}\right)$. We partition the free endpoints of $\mathcal{P}_{\text{aug}}$ into the set $B^{\text{close}}_{\text{aug}}$ that are at a Euclidean distance closer than $\lambda'_{\square} = 2\ell_{\square} n^{-\frac{1}{d+2}}$ to the divider $\Gamma_{\square}$ of $\square$ and the set $B^{\text{far}}_{\text{aug}}$ that are at a Euclidean distance further than $\lambda'_{\square}$ from $\Gamma_{\square}$. From Lemma 6.1, $|B^{\text{close}}_{\text{aug}}| = \tilde{O}(n^{1-\frac{1}{d+2}})$. Finally, by Equation (6.17),

$$\sum_{b \in B^{\text{far}}_{\text{aug}}} y(b) \leq \sum_{b \in B_{\text{aug}}} y(b) \leq w(M').$$

Since the dual weight of each free point in $B^{\text{far}}_{\text{aug}}$ is at least $(\lambda'_{\square})^q$, we get a bound of $\tilde{O}(n^{1-\frac{1}{d+2}})$ on the number of such points.

Therefore, the total number of executions of the ExtendedHungarianSearch procedure on each cell $\square$ is $\tilde{O}(n^{1-\frac{1}{d+2}})$. Each execution takes $\tilde{O}(n_{\square}\Phi(n))$ time; as a result, the total execution time of our algorithm on $\square$ would be $\tilde{O}(n^{1-\frac{1}{d+2}} n_{\square}\Phi(n))$. Since each point participates in $O(\log \Delta)$ cells in $\mathcal{H}$, the total execution time of our algorithm across all cells would be $\tilde{O}(n^{2-\frac{1}{d+2}}\Phi(n)\log \Delta)$, proving Lemma B.1.

B.2 Missing Details and Proofs of Chapter 7

In this section, we provide the missing proofs of lemmas and claims made in Section 7.2.

Lemma B.4. *For any partitioning $\mathcal{C}$ of the root square $\square^*$ of $\mathcal{H}$, any feasible extended matching $M^{\mathcal{C}} = (M, B^{\mathcal{C}}), y(\cdot)$ on $\mathcal{G}_{\sigma}$, and any admissible alternating or augmenting path P,*

the matching obtained after updating $M^{\mathcal{C}}$ along P remains feasible.

Proof. For any edge $(a, b) \in P$, due to the admissibility of the edge, $y(b) - y(a) = \mathrm{c}(a, b)$. If $(a, b) \notin M$ prior to augmentation, it is a matching edge after the augmentation and condition (7.3) holds for (a, b). Otherwise, $(a, b) \in M$ prior to augmentation, it is a non-matching edge after the augmentation, and condition (7.2) holds for (a, b). Note that all dual weights remain unchanged and consequently, conditions (7.4)–(7.6) remain satisfied. □

B.2.1 Missing Details and Proofs of the ExtendedHungarianSearch Procedure

Lemma 7.9. *After the execution of the extended Hungarian search procedure for a cell $\square$, the extended matching $M^{\mathcal{C}}, y(\cdot)$ remains feasible, $y(v) \leq \lambda$ for all points $v \in A_{\square} \cup B_{\square}$, and $y(b_f) = \lambda$ for all free points $b_f \in B_{\square}$. Furthermore, the path P computed by the procedure is a minimum net-cost augmenting path inside $\square$. After augmenting along P, the updated key for $\square$ is the smallest net-cost of all augmenting paths inside $\square$ and is at least λ.*

Let $y(\cdot)$ (resp. $y'(\cdot)$) denote the dual weights of the points after (resp. before) the update duals step of the extended Hungarian search procedure. To prove this lemma, we first show that after the update duals step, the extended matching $M^{\mathcal{C}}, y(\cdot)$ is feasible, $y(v) \leq \lambda$ for all $v \in A_{\square} \cup B_{\square}$, and $y(b_f) = \lambda$ for all free points $b_f \in B_{\square}$. We then show that the augmenting path P computed by the procedure is admissible to prove that P is a minimum net-cost augmenting path inside $\square$. We then use Lemma B.4 to show that after augmentation, the extended matching $M^{\mathcal{C}}, y(\cdot)$ remains feasible. Finally, we show that the updated key of $\square$ is the smallest net-cost of augmenting paths inside $\square$ and is at least λ.

Feasibility of points.

1. For any point $b \in B$:

   - if $b \in B \setminus B_\square$ is outside of $\square$, then $y(b) = y'(b)$; therefore, Conditions (7.4) and (7.5) remains satisfied.
   - Otherwise, $b \in B_\square$ and $\kappa \leq \kappa_b + s(b)$. Therefore, Condition (7.4) holds for b since

$$y(b) = y'(b) + \kappa - \kappa_b \leq y'(b) + s(b) = \mathrm{c}(b, \square). \tag{B.7}$$

2. For any free point $a \in A_F$:

   - if $a \in A \setminus A_\square$ is a free point outside of $\square$, then $y(a) = y'(a) = 0$ and Condition (7.6) remains true.
   - if $a \in A_\square$ is a free point inside $\square$, then $\kappa_a \geq \kappa$ and the procedure does not update the dual weight of a, i.e., the dual weight of a remains 0, satisfying Condition (7.6).

Feasibility of edges. For any edge $(a, b) \in E$, let $s(a, b)$ denote the slack of (a, b) before updating the dual weights. For any edge $(a, b) \in E$:

1. if $a \in A \setminus A_\square$ and $b \in B \setminus B_\square$, then $y(a) = y'(a)$ and $y(b) = y'(b)$; furthermore, the extended Hungarian search procedure does not add (resp. remove) the edge (a, b) to (resp. from) the matching and therefore, the feasibility conditions (7.2) and (7.3) remains satisfied for (a, b).

2. if $a \in A_\square$ and $b \in B \setminus B_\square$, then $y(b) = y'(b)$ and $y(a) \geq y'(a)$, since the extended Hungarian search procedure does not decrease the dual weight of any point inside $\square$; hence, $y(b) - y(a) \leq y'(b) - y'(a) \leq \mathrm{c}(a, b)$ and conditions (7.2) is satisfied. (Note that by Lemma 7.3, the edge (a, b) cannot be a matching edge).

3. if $a \in A \setminus A_\square$ and $b \in B_\square$, then $y(b) \leq \mathrm{c}(b, \mathcal{C}') \leq \mathrm{c}(a, b)$; hence, $y(b) - y(a) \leq y(b) \leq \mathrm{c}(a, b)$ and conditions (7.2) is satisfied. (Note that by Lemma 7.3, the edge (a, b) cannot be a matching edge).

4. if $a \in A \setminus A_\square$ and $b \in B \setminus B_\square$:

   - If $(a, b) \in M$ is a matching edge, then $\kappa_b = \kappa_a$ since the only edge directed to b in the residual graph is the zero-slack edge (a, b). Thus, the feasibility condition (7.3) holds since

   $$y(b) - y(a) = (y'(b) + \kappa - \kappa_b) - (y'(a) + \kappa - \kappa_a) = y'(b) - y'(a) = \mathrm{c}(a, b).$$

   - Otherwise, for any non-matching edge (a, b), $\kappa_a \leq \kappa_b + s(b, a)$, and the feasibility condition (7.2) holds since

   $$y(b) - y(a) = (y'(b) + \kappa - \kappa_b) - (y'(a) + \kappa - \kappa_a) \leq y'(b) - y'(a) + s(b, a) = \mathrm{c}(a, b). \tag{B.8}$$

Maximum dual weight. Let $y_{\max} := \max_{b \in B_\square} y'(b)$. By invariant (I2) prior to the extended Hungarian search procedure, for all free points $b \in B_\square$, $y'(b) = y_{\max}$. By the construction of the residual graph, $\kappa_b = y'(b) = y_{\max}$ for all free points $b \in B_\square$ and $\kappa_b \geq y_{\max}$ for all points $b \in B_\square$. Therefore, for any free point $b \in B_\square$,

$$y(b) = y'(b) + \kappa - \kappa_b = \kappa = \lambda,$$

where the last equality holds since κ is the net-cost of the minimum net-cost path inside $\square$, which is λ. Furthermore, for any point $b' \in B_\square$,

$$y(b) = y'(b) + \kappa - \kappa_b \leq y'(b) + \kappa - y_{\max} \leq \kappa = \lambda.$$

Note that for any free point $a \in A_\square$, $y(a) = 0$ and for any matched point $a \in A_\square$, if a is matched to a point $b \in B_\square$, then by Condition (7.3), $y(a) = y(b) - \mathrm{c}(a, b) \leq y(b) \leq \lambda$. Hence, we conclude $y(v) \leq \lambda$ for all points $v \in A_\square \cup B_\square$ and $y(b_f) = \lambda$ for all free points $b_f \in B_\square$ after the dual updates.

Net-cost of P. To prove that P is a minimum net-cost augmenting path, we show that P is an admissible augmenting path. Consequently, if $b \in B_\square$ is the free endpoint of P, by Corollary 7.5, $\phi(P) = y(b) = \lambda$. Note that for all other augmenting paths P' from a free point $b' \in B_\square$, from Lemmas 7.4, $\phi(P') \geq y(b') = \lambda$, and therefore, P would be a minimum net-cost augmenting path inside $\square$.

For each edge $(u, v) \in P$, $\kappa_v = \kappa_u + s(u, v)$, since (u, v) is an edge of the shortest path tree of the residual graph. Plugging into Equation (B.8), for each non-matching edge (b, a), $y(b) - y(a) = \mathrm{c}(a, b)$ and the edge (b, a) is admissible. Furthermore, if $\kappa = \kappa_b + s(b)$ for a point $b \in B$, then by Equation (B.7), $y(b) = \mathrm{c}(b, \square)$ and the point b would be a zero-slack point. Therefore, the path P computed by the algorithm would be an admissible path from a free point $b \in B_\square$ to either a free point $a \in A_\square$ or a zero-slack matched point $b' \in B_\square$, i.e., P is an admissible augmenting path.

From Lemma B.4, the extended matching $M^{\mathcal{C}}, y(\cdot)$ obtained after augmenting the matching $M^{\mathcal{C}}$ along P remains feasible.

Updated key. Note that the extended matching $M^{\mathcal{C}}, y(\cdot)$ after the augmentation step is feasible, $y(v) \leq \lambda$ for all vertices $v \in A_{\square} \cup B_{\square}$, and $y(b_f) = \lambda$ for all free points $b_f \in B_{\square}$. Let κ_v, for each $v \in A_{\square} \cup B_{\square}$, denote the distances computed in the update key step of the extended Hungarian search procedure. By Lemma 7.4, for any augmenting path P from a free point $b \in B_{\square}$ to a free point $a \in A_{\square}$, the net-cost of P is

$$\phi(P) = y(b) + \sum_{(a'',b'') \in P} s(a'', b'') = \kappa_a,$$

where the second equality holds by the construction of the residual graph. Similarly, for any augmenting path P from a free point $b \in B_{\square}$ to a matched point $b' \in B_{\square}$, the net-cost of P is

$$\phi(P) = y(b) + s(b') + \sum_{(a'',b'') \in P} s(a'', b'') = \kappa_{b'} + s(b').$$

Therefore, the updated key of $\square$, i.e., $\kappa_{\square} = \min\{\min_{a \in A_{\square}^F} \kappa_a, \min_{b \in B_{\square}} \kappa_b + s(b)\}$ correctly computes the net-cost of the minimum net-cost augmenting path inside $\square$. Finally, note that for any augmenting path P from a free point $b \in B_{\square}$, by Lemma 7.4, $\phi(P) \geq y(b) = \lambda$ and the key of $\square$ would be at least λ.

B.2.2 Missing Details and Proofs of the Merge Procedure

Lemma 7.10. *After the execution of the* Merge *procedure on a cell $\square$, the updated extended matching $M^{\mathcal{C}}, y(\cdot)$ is feasible, the dual weights $y(v)$ for every $v \in A_{\square} \cup B_{\square}$ is at most λ, and $y(b_f) = \lambda$ for all free points $b_f \in B_{\square}$. Furthermore, the updated key for $\square$ is the smallest net-cost of all augmenting paths within $\square$ and is at least λ.*

Let $\hat{M}^{\mathcal{C}} = (M, \hat{B}^{\mathcal{C}}), \hat{y}(\cdot)$ denote the feasible extended matching before the execution of the merge procedure. Let $B_{\square}^{\mathcal{C}}$ denote the subset of points in $B^{\mathcal{C}}$ that are matched to the divider

$\Gamma_\square$ of $\square$, and let $M^{\mathcal{C}} = (M, B^{\mathcal{C}} = \hat{B}^{\mathcal{C}} \setminus B^{\mathcal{C}}_\square)$ be the extended matching after adding the boundary-matched points in $B^{\mathcal{C}}_\square$ to the free points. By Lemma 7.11, the matching $M^{\mathcal{C}}, \hat{y}(\cdot)$ is feasible. Next, we show that, given a feasible extended matching $M^{\mathcal{C}}, y'(\cdot)$, after one iteration of the while-loop in the merge step, the matching remains feasible and the dual weights of points in $\square$ are at most λ.

Let $y(\cdot)$ (resp. $y'(\cdot)$) denote the dual weights of the points after (resp. before) the execution of one iteration of the merge step. We first show that $M^{\mathcal{C}}, y(\cdot)$ is a feasible extended matching where $y(v) \leq \lambda$ for all $v \in A_\square \cup B_\square$. We then show that the path P is admissible and use Lemma B.4 to show that after augmentation, the extended matching $M^{\mathcal{C}}, y(\cdot)$ remains feasible.

Feasibility of points.

1. For any point $b \in B$:
   - if $b \in B \setminus B_\square$ is outside of $\square$, then $y(b) = y'(b)$; therefore, Conditions (7.4) and (7.5) remains satisfied.
   - Otherwise, $b \in B_\square$ and $\kappa \leq \kappa_b + s(b)$. Therefore, Condition (7.4) holds for b since

$$y(b) = y'(b) + \kappa - \kappa_b \leq y'(b) + s(b) = \mathrm{c}(b, \square). \tag{B.9}$$

2. For any free point $a \in A_F$:
   - if $a \in A \setminus A_\square$ is a free point outside of $\square$, then $y(a) = y'(a) = 0$ and Condition (7.6) remains true.
   - if $a \in A_\square$ is a free point inside $\square$, then $\kappa_a \geq \kappa$ and the procedure does not update the dual weight of a, i.e., the dual weight of a remains 0, satisfying Condition (7.6).

Feasibility of edges. For any edge $(a, b) \in E$, let $s(a, b)$ denote the slack of (a, b) with respect to $y'(\cdot)$. For any edge $(a, b) \in E$:

1. if $a \in A \setminus A_\square$ and $b \in B \setminus B_\square$, then $y(a) = y'(a)$ and $y(b) = y'(b)$; furthermore, the merge procedure does not add (resp. remove) the edge (a, b) to (resp. from) the matching and therefore, the feasibility conditions (7.2) and (7.3) remains satisfied for (a, b).

2. if $a \in A_\square$ and $b \in B \setminus B_\square$, then $y(b) = y'(b)$ and $y(a) \geq y'(a)$, since the merge procedure does not decrease the dual weight of any point inside $\square$; hence, $y(b) - y(a) \leq y'(b) - y'(a) \leq \mathrm{c}(a, b)$ and conditions (7.2) is satisfied. (Note that by Lemma 7.3, the edge (a, b) cannot be a matching edge).

3. if $a \in A \setminus A_\square$ and $b \in B_\square$, then $y(b) \leq \mathrm{c}(b, \mathcal{C}') \leq \mathrm{c}(a, b)$; hence, $y(b) - y(a) \leq y(b) \leq \mathrm{c}(a, b)$ and conditions (7.2) is satisfied. (Note that by Lemma 7.3, the edge (a, b) cannot be a matching edge).

4. if $a \in A \setminus A_\square$ and $b \in B \setminus B_\square$:

   - If $(a, b) \in M$ is a matching edge, then $\kappa_b = \kappa_a$ since the only edge directed to b in the residual graph is the zero-slack edge (a, b). Thus, Condition (7.3) holds since

   $$y(b) - y(a) = (y'(b) + \kappa - \kappa_b) - (y'(a) + \kappa - \kappa_a) = y'(b) - y'(a) = \mathrm{c}(a, b).$$

   - Otherwise, for any non-matching edge (a, b), $\kappa_a \leq \kappa_b + s(b, a)$, and Condition (7.2) holds since

   $$y(b) - y(a) = (y'(b) + \kappa - \kappa_b) - (y'(a) + \kappa - \kappa_a) \leq y'(b) - y'(a) + s(b, a) = \mathrm{c}(a, b). \tag{B.10}$$

Maximum dual weight. Next, we show that $y(v) \leq \lambda$ for all $v \in A_\square \cup B_\square$. Note that for any point $b \in B_\square$, by the definition of κ, we have $\kappa \leq \kappa_b + \lambda - y'(b)$. Therefore, if $\kappa_b < \kappa$, then $y(b) = y'(b) - \kappa_b + \kappa \leq \lambda$. Otherwise, $\kappa_b \geq \kappa$ and $y(b) = y'(b) \leq \lambda$. Furthermore, for all free points $a \in A_\square$, by Condition (7.6), $y(a) = 0$ and for all matched points $a \in A_\square$, if a is matched to a point $b \in B_\square$, then $y(a) = y(b) - \mathrm{c}(a, b) \leq y(b) \leq \lambda$.

Net-cost of P. Next, we show that the path P computed in an iteration of the merge step is either (i) an admissible augmenting path, or (ii) an admissible alternating path from a free point $b \in B_\square$ to a matched point $b' \in B_\square$ with $y(b') = \lambda$. Then, from Lemma B.4, the extended matching obtained after updating $M^{\mathcal{C}}, y(\cdot)$ along P is feasible.

For each non-matching edge $(b, a) \in P$, since (b, a) is in the Dijkstra's shortest path tree, $\kappa_a = \kappa_b + s(b, a)$. Plugging into Equation (B.10), for each non-matching edge (b, a), $y(b) - y(a) = \mathrm{c}(a, b)$ and the edge (b, a) is admissible (i.e., all edges of P are admissible). Furthermore, if P is an augmenting path from a free point $b \in B$ to a matched point $b' \in B$, then $\kappa = \kappa_{b'} + s(b')$. Plugging into Equation (B.9), $y(b') = y'(b') + s(b') = \mathrm{c}(b', \mathcal{C})$. Therefore, P is an admissible alternating or augmenting path and the matching obtained by updating $M^{\mathcal{C}}$ along P remains feasible (Lemma B.4).

Termination. Let P be a path from a free point $b \in B_\square$ to a point $u \in A^F_\square \cup B_\square$.

- If $u \in A^F_\square$, then P is an admissible augmenting path and P is in case (i).
- Otherwise, if $u \in B_\square$ and $\kappa = \kappa_u + s(u)$, then P is an admissible augmenting path and P is in case (ii).
- Otherwise, $u \in B_\square$ and $\kappa = \kappa_u + \lambda - y(u)$. In this case, P is an admissible alternating path from b to a matched point $b' \in B_\square$ with $y'(b') = \lambda$.

In either case, the number of free points $b \in B_\square$ with $y(b) < \lambda$ reduces by one, and the merge step terminates.

Dual weight of free points. Note that the while-loop terminates when there are no free points $b_f \in B_\square$ with $y(b_f) < \lambda$, whereas as discussed above, all points will have a dual weight at most λ, i.e., the dual weight of each free point $b_f \in B_\square$ after the termination of the while-loop is λ.

Updated key. Finally, we show that the updated key of $\square$ denotes the net-cost of the minimum net-cost augmenting path inside $\square$. Note that the extended matching $M^{\mathcal{C}}, y(\cdot)$ after the termination of the while-loop is feasible, $y(v) \leq \lambda$ for all vertices $v \in A_\square \cup B_\square$, and $y(b_f) = \lambda$ for all free points $b_f \in B_\square$. Let κ_v, for each $v \in A_\square \cup B_\square$, denote the distances computed in the update key step of the extended Hungarian search procedure. By Lemma 7.4, for any augmenting path P from a free point $b \in B_\square$ to a free point $a \in A_\square$, the net-cost of P is

$$\phi(P) = y(b) + \sum_{(a'',b'') \in P} s(a'', b'') = \kappa_a,$$

where the second equality holds by the construction of the residual graph. Similarly, for any augmenting path P from a free point $b \in B_\square$ to a matched point $b' \in B_\square$, the net-cost of P is

$$\phi(P) = y(b) + s(b') + \sum_{(a'',b'') \in P} s(a'', b'') = \kappa_{b'} + s(b').$$

Therefore, the updated key of $\square$, i.e., $\kappa_\square = \min\{\min_{a \in A^F_\square} \kappa_a, \min_{b \in B_\square} \kappa_b + s(b)\}$ correctly computes the net-cost of the minimum net-cost augmenting path inside $\square$. Finally, note that for any augmenting path P from a free point $b \in B_\square$, by Lemma 7.4, $\phi(P) \geq y(b) = \lambda$ and the key of $\square$ would be at least λ.

B.2.3 Missing Proofs and Details of the Efficiency Analysis

Runtime Analysis of the MERGE Procedure

For any non-leaf cell $\square$ with $\square'$ and $\square''$ as children, the merge step at $\square$ first increases the dual weights of the free points inside $\square'$ and $\square''$ in $\tilde{O}(n_{\square}\Phi(n_{\square}))$ time (Lemma 7.7). For the feasible extended matching $M^{\mathcal{C}} = (M, B^{\mathcal{C}}), y(\cdot)$ this initial step, let $B^{\mathcal{C}}_{\square}$ denote the subset of the boundary-matched points in $B^{\mathcal{C}}$ that are matched to $\Gamma_{\square}$. As discussed in Section B.2.2, each iteration of the while-loop in the merge step reduces the number of free points with a dual weight less than λ by one; therefore, the total number of executions of the while-loop in the merge step is at most $|B^{\mathcal{C}}_{\square}|$.

Lemma B.5. *For any cell $\square$, the number of iterations of the merge step on $\square$ is $O(|B^{\mathcal{C}}_{\square}|)$.*

Each iteration requires the computation of the distance κ_v for each point $v \in A_{\square} \cup B_{\square}$, which takes $\tilde{O}(n^2_{\square})$ time. As shown in Section 7.4, the efficiency of this computation can be improved to $\tilde{O}(n_{\square}\Phi(n))$ using a dynamic weighted nearest neighbor data structure with a query/update time of $\Phi(n)$. Furthermore, by Lemma 7.11, $|B^{\mathcal{C}}_{\square}| = O(n^{4/5})$. Computing the updated key of $\square$ also requires the computation of the distance of each point from the source in the residual graph, which also takes $\tilde{O}(n_{\square}\Phi(n))$ time. Therefore, the total execution time of the merge step would be $\tilde{O}(n^{4/5}n_{\square}\Phi(n))$.

Runtime Analysis of the EXTENDEDHUNGARIANSEARCH Procedure

Recall that the extended Hungarian search procedure iteratively picks the cell with the minimum key from PRIORITYQUEUE to be processed. For each leaf cell $\square$ in $\mathcal{H}$, since $\square$ contains the points corresponding to a single request and contains only one point of B, the procedure picks $\square$ at most once, at which it matches the point $b \in B_{\square}$ to the boundaries of

$\square$. Therefore, the total time of the extended Hungarian search procedure on all leaf cells of $\mathcal{H}$ is $O(n)$.

Next, we show that for any non-leaf cell $\square$ of $\mathcal{H}$, our algorithm executes the search procedure on $\square$ at most $O(n^{4/5})$ times. Since each execution of the procedure takes $\tilde{O}(n_{\square}\Phi(n))$ time, the total running time of the extended Hungarian search procedure on all cells of $\mathcal{H}$ would be $O(n^{4/5}\sum_{\square\in\mathcal{H}} n_{\square}) = O(n^{9/5}\log(n\Delta))$, as claimed.

For any execution of the search procedure on $\square$, since $\square$ has the minimum key in PriorityQueue, the value λ represents the net-cost of the minimum net-cost augmenting path inside $\square$. Recall that for any non-leaf cell $\square$, we categorized the iterations of the search procedure on $\square$ as low-net-cost if the value of λ in that iteration is at most $\ell_{\square}n^{-1/5}$ and high-net-cost otherwise. To bound the high-net-cost executions, we show that as soon as the value λ exceeds $\ell_{\square}n^{-1/5}$, the number of free points inside $\square$ cannot be more than $O(n^{4/5})$, and therefore, the number of high-net-cost executions of the extended Hungarian search procedure on $\square$ is $O(n^{4/5})$.

Lemma 7.12. *Given a feasible extended matching $M^{\mathcal{C}}, y(\cdot)$ and any cell $\square \in \mathcal{C}$, if the net-cost of the minimum net-cost augmenting path inside $\square$ is greater than $\ell_{\square}n^{-1/5}$, then the number of free points of $M^{\mathcal{C}}$ is $O(n^{4/5})$.*

Proof. Let M' be the matching inside $\square$ as constructed in Lemma 7.1, and let M denote the matching of the extended matching $M^{\mathcal{C}} = (M, B^{\mathcal{C}})$. Define $B^{\mathcal{C}}_{\square}$ as the set of free points of $B_{\square}$ with respect to $M^{\mathcal{C}}, y(\cdot)$. Let $M_{\square}$ denote the subset of matching edges of M that lie inside $\square$, and let $\mathcal{P}_{\text{aug}}$ (resp. $\mathcal{P}_{\text{alt}}$) denote the set of (standard) augmenting (resp. alternating) paths in the symmetric difference $M_{\square} \oplus M'$ with one endpoint in the set $B^{\mathcal{C}}_{\square}$. Note that each path in $\mathcal{P}_{\text{alt}}$ has one endpoint that is free in M' and therefore, $|\mathcal{P}_{\text{alt}}| \leq |F(M')| = O(n^{4/5})$; here, $F(M')$ denote the set of free points of M'. Next, we show that $|\mathcal{P}_{\text{aug}}| = O(n^{4/5})$.

From the definition of the net-cost of an augmenting path,

$$\sum_{P\in\mathcal{P}_{\text{aug}}}\phi(P)=\sum_{P\in\mathcal{P}_{\text{aug}}}\left(\sum_{(a,b)\in P\cap M'}\mathrm{c}(a,b)-\sum_{(a,b)\in P\cap M_\square}\mathrm{c}(a,b)\right)$$
$$\leq\sum_{P\in\mathcal{P}_{\text{aug}}}\left(\sum_{(a,b)\in P\cap M}\mathrm{c}(a,b)\right)\leq w(M'). \tag{B.11}$$

For each path $P\in\mathcal{P}_{\text{aug}}$, let $b_P\in B_\square$ and $a_P\in A_\square$ denote the two end-points of P. Define $B_{\text{aug}}:=\{b_P:P\in\mathcal{P}_{\text{aug}}\}$ as the set of free endpoints of the paths in $\mathcal{P}_{\text{aug}}$. Using Lemma 7.4 and Equation (B.11),

$$\sum_{b\in B_{\text{aug}}}y(b)=\sum_{P\in\mathcal{P}_{\text{aug}}}y(b_P)\leq\sum_{P\in\mathcal{P}_{\text{aug}}}\phi(P)\leq w(M'). \tag{B.12}$$

From invariant (I2), the dual weight of all free points of $B_\square$ equals $\lambda>\ell_\square n^{-1/5}$. Therefore,

$$|B_{\text{aug}}|=\frac{\sum_{b\in B_{\text{aug}}}y(b)}{\lambda}\leq\frac{w(M')}{\ell_\square n^{-1/5}}=O(n_\square^{3/5}n^{1/5})=O(n^{4/5}).$$

Combining the two bounds, the total number of free points with respect to $M^{\mathcal{C}},y(\cdot)$ is $|\mathcal{P}_{\text{alt}}|+|\mathcal{P}_{\text{aug}}|=O(n^{4/5})$. □

Define $\mathcal{C}_\square$ as the set of all cells of $\mathcal{H}$, including $\square$ itself, that are processed by the merge step of our algorithm while $\square\in\mathcal{C}$ and $\lambda\leq\ell_\square n^{-1/5}$. For any cell $\square'\in\mathcal{C}_\square$, let $\mathbb{B}_{\square'}$ denote the set of points of B that are matched to the divider $\Gamma_{\square'}$ at the beginning of the merge step at $\square'$. During the execution of our algorithm, before processing $\square'$ by the merge step, there are k free points across all cells is $\mathcal{C}$. After combining the two children of $\square'$ (and removing the divider of $\square'$), the number of free points across all cells in the partitioning is now increased to $k+|\mathbb{B}_{\square'}|$. Each iteration of the merge step either (i) finds an admissible augmenting path, which reduces the number of free points by one, or (ii) finds an admissible alternating

path to a matched point $b \in B_{\square'}$ with $y(b) = \lambda$, which does not change the number of free points. Therefore, after the merge step at $\square'$, the number of free points across all cells in the partitioning is at most $k+|\mathbb{B}_{\square'}|$. Our algorithm then iteratively executes the search procedure to reduce the number of free points across all cells to k. Hence, for each cell $\square' \in \mathcal{C}_{\square}$, the merge step at $\square'$ might lead to the execution of a low-net-cost extended Hungarian search procedure on $\square$ at most $|\mathbb{B}_{\square'}|$ times.

Lemma B.6. *For any cell $\square$, the number of low-net-cost executions of the extended Hungarian search procedure on $\square$ is at most $\sum_{\square' \in \mathcal{C}_{\square}} |\mathbb{B}_{\square'}|$.*

Recall that in the low-net-cost executions of the search procedure on $\square$, the value $\lambda \leq \ell_{\square} n^{-1/5}$. Using invariant (I2), the dual weight of all points of B are at most λ, and therefore, for any cell $\square' \in \mathcal{C}_{\square}$ and each point $b \in \mathbb{B}_{\square'}$, $y(b) \leq \ell_{\square} n^{-1/5}$, i.e., b is a boundary-matched point that is matched to the divider $\Gamma_{\square'}$ and has a dual weight at most $\ell_{\square} n^{-1/5}$. Therefore, $\mathrm{c}(b, \Gamma_{\square'}) = y(b) \leq \ell_{\square} n^{-1/5}$ and all points in $\mathbb{B}_{\square'}$ are at a distance at most $\ell_{\square} n^{-1/5}$ from the divider $\Gamma_{\square'}$. From Lemma 7.13, $\ell_{\square} \leq 3\ell_{\square'}$. Therefore, using Lemma 6.1, the number of points of $B_{\square'}$ at a distance at most $\lambda \ell_{\square'} \geq \ell_{\square} n^{-1/5}$ to the divider $\Gamma_{\square'}$ is $O(n_{\square'} n^{-1/5})$; hence, $|\mathbb{B}_{\square'}| = O(n_{\square'} n^{-1/5})$.

Lemma 7.13. *For any non-leaf cell $\square$ in $\mathcal{H}$ and any cell $\square' \in \mathcal{C}_{\square}$, $\frac{1}{3}\ell_{\square'} \leq \ell_{\square} \leq \frac{9}{4}\ell_{\square'}$.*

Proof. To prove this lemma, we first provide two useful relations between the sum of side-lengths $p_{\hat{\square}}$ and the largest side-length $\ell_{\hat{\square}}$ of any cell $\hat{\square}$ of $\mathcal{H}$. Let ℓ_x (resp. ℓ_y) denote the width (resp. height) of $\hat{\square}$. Since the aspect ratio of $\hat{\square}$ is at most 3, i.e., $\min\{\ell_x, \ell_y\} \geq \frac{1}{3}\ell_{\square} = \frac{1}{3}\max\{\ell_x, \ell_y\}$,

$$\frac{4}{3}\ell_{\square} \leq \ell_x + \ell_y = p_{\hat{\square}} = \ell_x + \ell_y \leq 2\ell_{\square}.$$

Furthermore, for the smaller child $\hat{\square}_1$ of $\hat{\square}$, if cell is divided on the x axis, then the width of

$\hat{\square}_1$ is within $\frac{1}{3}\ell_{\hat{\square}}$ and $\frac{1}{2}\ell_{\hat{\square}}$; therefore,

$$\frac{4}{3}p_{\hat{\square}_1} \leq p_{\hat{\square}} \leq 2p_{\hat{\square}_1}.$$

For the cell $\square$ (resp. $\square'$), suppose $\square_{\min}$ (resp. $\square'_{\min}$) denote the smaller child of $\square$ (resp. $\square'$). Since our algorithm picked $\square_{\min}$ for being merged as the smallest cell before $\square'_{\min}$,

$$\frac{4}{3}\ell_{\square} \leq p_{\square} \leq 2p_{\square_{\min}} \leq 2p_{\square'_{\min}} \leq \frac{3}{2}p_{\square'} \leq 3\ell_{\square'}. \tag{B.13}$$

Similarly, since our algorithm picked $\square'_{\min}$ as the smallest cell to be processed rather than $\square$,

$$\frac{4}{3}\ell_{\square'} \leq p_{\square'} \leq 2p_{\square'_{\min}} \leq 2p_{\square} \leq 4\ell_{\square}. \tag{B.14}$$

Combining Equations (B.13) and (B.14),

$$\frac{1}{3}\ell_{\square'} \leq \ell_{\square} \leq \frac{9}{4}\ell_{\square'}.$$

□

We next show that $\sum_{\square' \in \mathcal{C}_{\square}} n_{\square'} = O(n)$ and conclude that $\sum_{\square' \in \mathcal{C}_{\square}} |\mathbb{B}_{\square'}| = O(n^{4/5})$, which bounds the number of low-net-cost executions of the search procedure on $\square$ by $O(n^{4/5})$, as desired. By Lemma 7.13, for any cell $\square' \in \mathcal{C}_{\square}$, $\ell_{\square'} \in [\frac{1}{3}\ell_{\square}, \frac{9}{4}\ell_{\square}]$. By the construction of $\mathcal{H}$, for any cell $\square'$ and its grandparent $\square'_g$, $\ell_{\square'} \leq \frac{2}{3}\ell_{\square'_g}$. Therefore, for $\square' \in \mathcal{C}_{\square}$, only the ancestor of $\square'$ up to $2\log_{3/2}\frac{27}{4}$ levels higher can also be in $\mathcal{C}_{\square}$; therefore, for any point $u \in A \cup B$, the point u lies inside at most $O(1)$ cells of $\mathcal{C}_{\square}$, and $\sum_{\square' \in \mathcal{C}_{\square}} n_{\square'} = O(n)$, as claimed.

We conclude that the total number of executions of the extended Hungarian search for each cell $\square$ of $\mathcal{H}$ is $O(n^{4/5})$. The next lemma follows since each execution takes $\tilde{O}(n_{\square}\Phi(n))$ time.

Lemma B.7. *For any cell $\square$ of $\mathcal{H}$, the total execution time of the extended Hungarian search procedure on $\square$ takes $\tilde{O}(n^{4/5} n_{\square} \Phi(n))$ time.*

B.2.4 Extension to higher dimensions

Given a set of requests σ in the d dimensions, for any $d \geq 2$, let $\mathcal{G}_\sigma$ on point sets A and B denote the graph constructed for the k-SP problem on σ under any ℓ_p norm for some $p \geq 1$. Let $\mathcal{H}_d$ denote the hierarchical partitioning constructed for the point set $A \cup B$ with a parameter $\lambda = 9n^{-\frac{1}{2d+1}}$. The hierarchical partitioning has a height $O(d \log(n\Delta))$. Using $\mathcal{H}_d$, we run our algorithm as described in Section 7.2.

We summarize the efficiency analysis of our algorithm for d-dimensional point sets next and show that our algorithm runs in $\tilde{O}(n^{2-\frac{1}{2d+1}} \Phi(n) \log \Delta)$ time. In particular, we show that for any cell $\square$, the number of iterations of the merge step on $\square$ is $O(n^{1-\frac{1}{2d+1}})$, where each iteration takes $\tilde{O}(n_{\square} \Phi(n))$ time. We also show that our algorithm runs the extended Hungarian search procedure $O(n^{1-\frac{1}{2d+1}})$ time on $\square$, each in $\tilde{O}(n_{\square} \Phi(n))$ time. Adding these bounds for all cells of $\mathcal{H}_d$, the total running time of our algorithm is $\tilde{O}(n^{2-\frac{1}{2d+1}} \Phi(n) \log \Delta)$. We summarize the details below.

Given a feasible extended matching $M^{\mathcal{C}}, y(\cdot)$, for any cell $\square \in \mathcal{C}$, let $B^{\mathcal{C}}_{\square}$ denote the set of all points of $B_{\square}$ that are matched to the divider $\Gamma_{\square}$ in the matching before the execution of the merge step at $\square$. By Lemma B.5, the number of iterations of the merge step on $\square$ is $O(|B^{\mathcal{C}}_{\square}|)$. To bound the number of points in $B^{\mathcal{C}}_{\square}$, we first show that there exists a partial matching M' of a high cardinality and a low cost.

Lemma B.8. *For any cell $\square$ of $\mathcal{H}_d$, there exists a matching M' over $\mathcal{G}_\sigma$ inside $\square$ that matches all except $O(n_{\square}^{1-\frac{1}{2d+1}})$ points of $B_{\square}$ and has a cost $O(\ell_{\square} n_{\square}^{1-\frac{2}{2d+1}})$.*

Proof Sketch. Similar to our construction for Lemma 7.1, to construct the matching M' for Lemma B.8, we place a grid of cell-side-length $\ell_\square n_\square^{-\frac{2}{2d+1}}$ and compute the matching corresponding to the 1-partitioning of the requests inside each cell of this grid. It is easy to confirm that the matching M' achieves the bounds claimed in Lemma B.8. □

For the feasible extended matching $M^{\mathcal{C}} = (M, B^{\mathcal{C}}), y(\cdot)$, let $M_\square$ denote the subset of matching edges of M that lie inside $\square$. Let $\mathcal{P}_{\text{aug}}$ (resp. $\mathcal{P}_{\text{alt}}$) denote the set of augmenting paths (resp. alternating paths) with an endpoint in $|B^{\mathcal{C}}_\square|$ in the symmetric difference $M_\square \oplus M'$. As discussed above, $|\mathcal{P}_{\text{alt}}| \leq |F(M')| = O(n^{1-\frac{1}{2d+1}})$, where $F(M')$ denotes the set of free points of $B_\square$ with respect to M'. We partition the free endpoints of $\mathcal{P}_{\text{aug}}$ into the set $B^{\text{close}}_{\text{aug}}$ (resp. $B^{\text{far}}_{\text{aug}}$) that are at a distance closer than (resp. further than) $\lambda'_\square = \ell_\square n^{-\frac{1}{2d+1}}$ to the divider $\Gamma_\square$ of $\square$. From Lemma 6.1, $|B^{\text{close}}_{\text{aug}}| = O(n^{1-\frac{1}{2d+1}})$. Finally, by Equation (7.14), $\sum_{b \in B^{\text{far}}_{\text{aug}}} y(b) \leq w(M')$. Since the dual weight of each free point in $B^{\text{far}}_{\text{aug}}$ is at least $\lambda'_\square$, we get a bound of $O(n^{1-\frac{1}{2d+1}})$ on the number of such points, and a total execution time of $\tilde{O}(n^{1-\frac{1}{2d+1}} n_\square \Phi(n))$ for the merge step on a cell $\square$ of $\mathcal{H}$.

Next, we bound the number of executions of the extended Hungarian search procedure for each cell $\square$. An execution of extended Hungarian search procedure on $\square$ is low-net-cost if $\lambda \leq \ell_\square n^{-\frac{1}{2d+1}}$ and high-net-cost otherwise. For the high-net-cost executions, one can use the matching M' from Lemma B.8 and a similar argument as in Lemma 7.12 to show that when $\lambda > \ell_\square n^{-\frac{1}{2d+1}}$, the total number of free points inside $\square$ cannot exceed $O(n^{1-\frac{1}{2d+1}})$ and conclude an upper-bound of $O(n^{1-\frac{1}{2d+1}})$ on the number of high-net-cost executions of the extended Hungarian search procedure. To bound the number of low-net-cost executions of the extended Hungarian search procedure on $\square$, define $\mathcal{C}_\square$ as the set of all cells $\square' \in \mathcal{H}$ that are processed by the merge step while $\square$ is in $\mathcal{C}$ and $\lambda \leq \ell_\square n^{-\frac{1}{2d+1}}$. For any cell $\square' \in \mathcal{C}_\square$, let $\mathbb{B}_{\square'}$ denote the set of boundary-matched points of $B_{\square'}$ matched to the divider of $\square'$ before the execution of the merge step on $\square'$. By Lemma B.6, the number of low-net-cost

executions of the extended Hungarian search procedure on $\square$ is at most $\sum_{\square' \in \mathcal{C}_\square} |\mathbb{B}_{\square'}|$. Since $\lambda \leq \ell_\square n^{-\frac{1}{2d+1}}$, all points in $\mathbb{B}_{\square'}$, for each cell $\square' \in \mathcal{C}_\square$, are at a distance at most $\ell_\square n^{-\frac{1}{2d+1}}$ to the divider of $\square'$. Therefore, from Lemma 6.1 and 7.13 and using an identical discussion as above, $\sum_{\square' \in \mathcal{C}_\square} \mathbb{B}_{\square'} = O(dn^{1-\frac{1}{2d+1}})$. Therefore, the total execution time of the extended Hungarian search step on any cell $\square$ of $\mathcal{H}$ is $\tilde{O}(dn^{1-\frac{1}{2d+1}} n_\square \Phi(n))$.

Combining the total execution times of the merge step and the extended Hungarian search step, the running time of our algorithm would be

$$\tilde{O}(\sum_{\square \in \mathcal{H}} dn^{1-\frac{1}{2d+1}} n_\square \Phi(n)) = \tilde{O}(dn^{2-\frac{1}{2d+1}} \Phi(n) \log \Delta),$$

leading to the following theorem.

Theorem B.9. *Given any sequence σ of n requests in d dimensions with a spread of Δ, and a value $1 \leq k \leq n$, there exists a deterministic algorithm that computes the optimal solution for the instance of k-SP problem under the ℓ_p norm in $\tilde{O}(\min\{nk, n^{2-\frac{1}{2d+1}} \log \Delta\} \cdot \Phi(n))$ time.*