Deterministic, Near-Linear ε-Approximation Algorithm for Geometric Bipartite Matching*

Pankaj K. Agarwal
Duke University
USA

Hsien-Chih Chang
Dartmouth College
USA

Sharath Raghvendra
Virginia Tech
USA

Allen Xiao
Duke University
USA

ABSTRACT

Given two point sets A and B in $\mathbb{R}^d$ of size n each, for some constant dimension $d \geq 1$, and a parameter $\varepsilon > 0$, we present a deterministic algorithm that computes, in $n \cdot (\varepsilon^{-1} \log n)^{O(d)}$ time, a perfect matching between A and B whose cost is within a $(1+\varepsilon)$ factor of the optimal matching under any ℓ_p-norm. Although a Monte-Carlo algorithm with a similar running time is proposed by Raghvendra and Agarwal [J. ACM 2020], the best-known deterministic ε-approximation algorithm takes $\Omega(n^{3/2})$ time. Our algorithm constructs a (refinement of a) tree cover of $\mathbb{R}^d$, and we develop several new tools to apply a tree-cover based approach to compute an ε-approximate perfect matching.

CCS CONCEPTS

• **Theory of computation → Randomness, geometry and discrete structures**; **Graph algorithms analysis**.

KEYWORDS

matching, augmenting path, tree cover, regularizer, compression

ACM Reference Format:
Pankaj K. Agarwal, Hsien-Chih Chang, Sharath Raghvendra, and Allen Xiao. 2022. Deterministic, Near-Linear ε-Approximation Algorithm for Geometric Bipartite Matching. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC '22), June 20–24, 2022, Rome, Italy*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3519935.3519977

1 INTRODUCTION

Let A and B be two point sets in $\mathbb{R}^d$ of size n each, where the dimension d is a constant. Consider the complete weighted bipartite graph G with cost function $\mathfrak{c}(e) := \|a - b\|$, where $\|\cdot\|$ denotes the Euclidean norm.[1] A *matching* M is a set of vertex-disjoint edges in G. We say that a matching is *perfect* if $|M| = n$. The cost of M is the sum of its edge costs: $\mathfrak{c}(M) := \sum_{e \in M} \mathfrak{c}(e)$.

The *Euclidean minimum-weight matching* (EMWM) in G is denoted as $M_{\text{opt}} := \arg\min_{|M|=n} \mathfrak{c}(M)$. A perfect matching M is called an *ε-approximation* if $\mathfrak{c}(M) \leq (1+\varepsilon) \cdot \mathfrak{c}(M_{\text{opt}})$.

EMWM can be used to estimate the Wasserstein distance (a measure of similarity) between two probability distributions. Due to this, it has received considerable attention in machine learning and computer vision [28, 30, 33]. For instance, suppose we are given two (possibly unknown) probability distributions in $\mathbb{R}^d$, we can estimate their Wasserstein distance by taking n samples from each distribution and then computing a minimum-weight matching between them [9, 25]. These applications call for the design of exact and approximation algorithms for the EMWM problem. In this paper, we present a *deterministic* near-linear-time ε-approximation algorithm for the EMWM problem. All known near-linear-time algorithms for this problem are Monte-Carlo algorithms, and existing deterministic ε-approximations take $\Omega(n^{3/2})$ time.

Related work. The classical Hopcroft-Karp algorithm computes a maximum-cardinality matching in a bipartite graph with n vertices and m edges in $O(m\sqrt{n})$ time [19]. The first improvement in over thirty years, by Mądry [26], runs in $O(m^{10/7} \operatorname{polylog} n)$ time. The bound was further improved to $O((m + n^{3/2}) \operatorname{polylog} n)$ by Brand *et al.* [10]. The Hungarian algorithm computes the minimum-weight maximum cardinality matching in $O(mn+n^2 \log n)$ time [27]; see also [14, 16]. Faster algorithms for computing optimal matchings can be obtained by using recent min-cost max-flow algorithms [10]. There is also extensive work on computing optimal matchings matchings in non-bipartite graphs [15, 17, 36].

If A and B are points in $\mathbb{R}^2$, the best known algorithm for computing EMWM runs in $O(n^2 \operatorname{polylog} n)$ time [1, 2, 21]. If points have integer coordinates bounded by Δ, the running time can be improved to $O(n^{3/2} \operatorname{polylog} n \log \Delta)$ [31]. It is an open question whether a subquadratic algorithm exists for computing EMWM if coordinates of input points have real values. In contrast, Varadarajan [34] presented an $O(n^{3/2} \operatorname{polylog} n)$-time algorithm for the non-bipartite case under any ℓ_p-norm — this is surprising because the non-bipartite case seems harder for graphs with arbitrary edge costs. As for higher dimensions, Varadarajan and Agarwal [35] presented an $O(n^{3/2}\varepsilon^{-d} \log^d n)$-time ε-approximation algorithm for computing EMWM of points lying in $\mathbb{R}^d$. The running time was later improved to $O(n^{3/2}\varepsilon^{-d} \log^5 n)$ by Agarwal and Raghvendra [32]. For any $0 < \delta \leq 1$, they also proposed a deterministic $O(1/\delta)$-approximation algorithm that runs in $O(n^{1+\delta} \log^d n)$ time [4].

Randomly-shifted quadtrees have played a central role in designing Monte-Carlo approximation algorithm for EMWM. It is

*Full version of the paper can be found on arxiv.org/abs/2204.03875.
[1] Our algorithm works for any ℓ_p-norm; but to be concrete we focus on the ℓ_2-norm.

STOC '22, June 20–24, 2022, Rome, Italy

ACM ISBN 978-1-4503-9264-8/22/06…$15.00
https://doi.org/10.1145/3519935.3519977

well-known that a simple greedy algorithm on a randomly-shifted quadtree yields (in expectation) an $O(\log n)$-approximation algorithm of EMWM [12]. Agarwal and Varadarajan [5] build upon this observation and use a randomly-shifted-quadtree-type hierarchical structure to obtain an expected $O(\log 1/\delta)$-approximation of EMWM in $O(n^{1+\delta})$ time. Combining their approach with importance sampling, Indyk [20] presented an algorithm that approximates the cost of EMWM within an $O(1)$-factor with high probability in time $O(n \operatorname{polylog} n)$. His algorithm, however, only returns the optimal cost but not the matching itself. Similarly, Andoni *et al.* [6] gave an ε-approximation streaming algorithm to the cost that runs in $O(n^{1+o_\varepsilon(1)})$ time. Finally, Raghvendra and Agarwal [29] proposed a Monte-Carlo algorithm that computes an ε-approximate EMWM with high probability in $n \cdot (\varepsilon^{-1} \log n)^{O(d)}$ time. Roughly speaking, their algorithm embeds the Euclidean metric into a tree metric, based on a refinement of a randomly-shifted quadtree T, with at most $1+\varepsilon$ distortion in expectation. The algorithm iteratively computes the minimum net-cost augmenting path with respect to the tree metric and augments the matching along the path. To compute the minimum net-cost path, it stores $O(\operatorname{poly} \log n)$ sub-paths (alternating paths) at each cell of the quadtree. They show that any sub-path at a cell, including the minimum net-cost augmenting path, can be expressed as some combination of the $O(\operatorname{poly} \log n)$ sub-paths stored at its children. Using this observation, they build a dynamic data structure that stores a matching, and supports computing and augmenting along a minimum net-cost path in time proportional to its length. Similar to the Gabow-Tarjan algorithm [16], a small fixed penalty is added to the net cost of each edge so that the total length of the augmenting paths computed by the algorithm is $O(\varepsilon^{-1} n \log n)$, leading to a near-linear-time ε-approximation algorithm. The randomized quadtree framework also has been successfully applied to designing fast approximation algorithm for the transportation problem as well as for matching under different cost functions [3, 13, 22, 23].

Our results. The following theorem is our main result:

Theorem 1.1. *Let A and B be two point sets in $\mathbb{R}^d$ of size n each, where dimension d is a constant, and let $\varepsilon > 0$ be a parameter. A perfect matching of A and B of cost at most $(1+\varepsilon) \cdot \mathfrak{c}(M_{\text{opt}})$ can be computed in $n \cdot (\varepsilon^{-1} \log n)^{O(d)}$ time in the worst case.*

Extending Raghvendra-Agarwal approach [29] to develop a near-linear deterministic algorithm runs into several difficulties. Neither can we use a randomly-shifted quadtree, nor can we use a stochastic embedding of Euclidean metric into a tree metric. Instead we work with Euclidean metric directly. We replace a single randomly-shifted quadtree with 2^d deterministically shifted quadtrees and then define a refinement on each of these trees. Our shifted quadtrees can be viewed as a *tree cover* of Euclidean space, a notion introduced by Gupta, Kumar, and Rastogi [18]; the Euclidean distance between a pair of points is deterministically approximated in at least one of these 2^d quadtrees. The previous applications of tree covers were limited to *decomposable problems* where one can solve the problem by simply merging the solutions for all the trees, for example routing, spanners, nearest-neighbor searching, and network design [7, 8, 11, 18, 24]. It seems challenging to apply a tree-cover-based approach to non-decomposable geometric problems such as the TSP or the EMWM, whose approximation algorithms are based on dynamic programming. We create several new tools to overcome the challenges that arise in applying tree cover to EMWM.

When we work with a tree cover, portions of the min-net-cost augmenting path may appear at different levels in different trees. As a result, maintaining $O(\operatorname{poly} \log n)$ sub-paths per cell will not be sufficient to faithfully reconstruct the min-net-cost augmenting path. Nevertheless, we maintain only $O(\operatorname{poly} \log n)$ sub-paths per cell and settle with a weaker claim. We define the *θ-adjusted cost* of an augmenting path P to be its net cost plus $\theta \cdot \|P\|$, where $\|P\|$ is the Euclidean arc length of P. We show that using the sub-paths we can compute an augmenting path P whose θ-adjusted cost is at most the $O(\theta \log n)$-adjusted cost of the min-net-cost augmenting path (see FindPath procedure (A1) and Lemma 5.2). Choosing such sub-optimal paths in tree covers leads to the following problems.

First, augmenting along a path P now introduces an additive error $O(\theta \log n \cdot \|P\|)$ to the cost of the matching. Large errors in the net-cost are introduced by non-matching edges on the endpoints of its sub-paths, as a connection cost between sub-paths. This error accumulates over the execution of the algorithm, thereby making it difficult to bound the total error in the matching cost. We periodically repair the intermediate matching by canceling alternating cycles that have a negative θ-adjusted cost. Any such cycle C has a net-cost at most $-\theta\|C\|$; in other words, canceling this cycle reduces the matching cost by at least $\theta\|C\|$. We refer to them as *reducing cycles*. In fact, our algorithm only finds and cancels sufficiently many alternating cycles with a negative θ-adjusted cost so that a weaker invariant is satisfied, i.e., the $(\theta \log n)$-adjusted costs of all cycles is non-negative. By setting $\theta := \varepsilon/\log n$, we are able to bound the cost of the resulting matching to be $(1+\varepsilon)M_{\text{opt}}$. We show that the time spent in canceling reducing cycles is bounded by the total time spent augmenting matchings. All the earlier EMWM approximation algorithms computed a minimum-net-cost augmenting path under a suitable cost function and guaranteed that no negative net-cost cycles were created.

Second, to extract an augmenting path (or a reducing cycle) from a cell $\boxplus$ in the tree cover, the sub-paths are recursively expanded using those stored at the children cells of $\boxplus$. A major complication from the overlapping grid cells of the tree-cover is that the augmenting path we compute — by going down the hierarchy of cells and sub-paths — may be self-intersecting, in which a matching edge appears multiple times. Suppose $P = P_1 \circ P_2$, where P_1 and P_2 are expansions of sub-paths at the children cells of $\boxplus$. Assuming P_1 and P_2 are simple, to guarantee that P is also simple, we first check whether P_1 and P_2 share a matching edge. If the answer is yes, then P contains a cycle C, which we can extract from P. If C is a reducing cycle, then we update the current matching by "canceling" C and ignoring P. Otherwise we remove C from P. We repeat this step until P becomes simple (or a reducing cycle is found).

Unfortunately, it is too expensive to first compute the self-intersecting path P and then simplify it — namely, check whether a matching edge in P appears more than once, and if so, identify the cycle C in P containing the edge and splice out C. An important component of our algorithm is a dynamic data structure for quickly detecting cycles and simplifying augmenting paths. Roughly speaking, we maintain intersection information about a collection of *canonical paths* implicitly, using the hierarchy of residual graphs,

each of which is guaranteed to be simple. An augmenting path is constructed by concatenating these canonical paths at various levels. At the heart of the data structure is a compact representation of canonical paths (despite the total complexity of canonical paths being quadratic, we are able to store them using near-linear space) and efficient procedures to detect whether two canonical paths share a matching edge (the intersection of two canonical paths may consist of many components), to extract cycles in the implicit representation of an alternating path as concatenation of a sequence of canonical paths, and to splice a cycle from such a path.

The length-dependent penalty in the adjusted cost also acts as a regularizer, and we use it to bound the total Euclidean length and total number of edges in all the augmenting paths and reducing cycles we compute. The total running time remains near-linear (see AUGMENT procedure (A2)).

The paper is organized as follows: Section 2 gives an overview of the overall algorithm; many of its details and analysis are presented in Sections 5 and 6, respectively. The overall data structure based on tree-cover, that finds an augmenting path and augments the matching is presented in Section 3, and its performance is analyzed in Section 7. Section 4 present the data structure for storing canonical paths compactly and the procedures that this data structure supports, including the intersection query. Due to space constraint, many proofs and details are omitted from this version.

2 OVERALL ALGORITHM

Preliminaries. Given a matching M, the *residual graph* of G with respect to M, denoted by $\vec{G}_M = (V, E_M)$, is a directed graph on the vertices of G in which all non-matching edges are directed from A to B and all matching edges from B to A. A vertex of $\vec{G}_M$ is called *free* if it is not incident to any edge of M, and *matched* otherwise. An *alternating path* (cycle) Π in G is a path whose edges alternate between non-matching and matching edges; Π maps to a directed path (cycle) in $\vec{G}_M$. Define the *net cost* $\bar{\mathfrak{c}}_M \colon E_M \to \mathbb{R}$ on $\vec{G}_M$ as follows: $\bar{\mathfrak{c}}_M(a,b) := \mathfrak{c}(a,b)$ if $(a,b) \notin M$, and $\bar{\mathfrak{c}}_M(b,a) := -\mathfrak{c}(a,b)$ if $(a,b) \in M$. The net cost of any alternating path or cycle Π is $\bar{\mathfrak{c}}_M(\Pi) = \sum_{e \in \Pi \setminus M} \mathfrak{c}(e) - \sum_{e \in \Pi \cap M} \mathfrak{c}(e)$. If Π is an alternating path or cycle, we use the following shorthand for the arc length of Π (as a polygonal curve): $\|\Pi\| := \sum_{e \in \Pi} \|e\|$. A simple (non-intersecting) alternating path Π between two free vertices is called an *augmenting path*, and $M \oplus \Pi$ is a matching of size $|M|+1$ from augmenting M with Π. If Π is an alternating cycle, then $|M \oplus \Pi| = |M|$ instead. The Hungarian algorithm repeatedly finds an augmenting path Π in $\vec{G}_M$ of minimum net cost and augments the matching by Π.

Adjusted cost. Let c_0 be a constant whose value will be chosen later. Let M be any fixed matching. For a parameter $\theta \geq 0$, we define the *θ-adjusted cost* of an edge e to be $\alpha_{\theta,M}(e) := \bar{\mathfrak{c}}_M(e) + c_0\theta \cdot \mathfrak{c}(e)$, where $\bar{\mathfrak{c}}_M(e)$ is the net cost of e in the residual graph $\vec{G}_M$. For a set X of edges in $\vec{G}_M$, define $\alpha_{\theta,M}(X) := \sum_{e \in X} \alpha_{\theta,M}(e)$. We can interpret $\alpha_{\theta,M}$ as adding a *regularizer* to the net cost of a residual path or cycle. We fix two parameters: *upper* $\bar{\varepsilon} := \frac{\varepsilon}{c_1}$ and *lower* $\underline{\varepsilon} := \frac{\bar{\varepsilon}}{c_2 \log n}$ where $c_1 \geq 8c_0$ and $c_2 > 0$ are constants. For a matching M, we define $\alpha^*_{\bar{\varepsilon},M} := \min_\Pi \alpha_{\bar{\varepsilon},M}(\Pi)$, where the minimum is taken over all augmenting paths with respect to M. If the matching M is clear from the context, we sometimes drop M from the subscript. We call a cycle Γ in $\vec{G}_M$ *reducing* if $\alpha_{\underline{\varepsilon}}(\Gamma) < 0$. We note that if $\alpha_{\bar{\varepsilon}}(\Gamma) < 0$, then Γ is reducing (since $\underline{\varepsilon} \leq \bar{\varepsilon}$). Intuitively, canceling a reducing cycle decreases the matching cost significantly relative to the cycle length (which is proportional to the amount of time required to cancel): $\alpha_{\underline{\varepsilon}}(\Gamma) < 0$ implies $\bar{\mathfrak{c}}(\Gamma) < -c_0\underline{\varepsilon} \cdot \|\Gamma\|$.

Overview of the algorithm. We now present a high-level description of the algorithm. We begin by performing a preprocessing step, the details of which can be found in the full version so that the input is "well-conditioned" at a slight increase in the cost of optimal matching (within an $(\varepsilon/8)$-factor). After this preprocessing step, which takes $O(n \log^2 n)$ time in total, we have point sets A and B that satisfy the following three properties:

- P1. All input points have integer coordinates.
- P2. No integer grid point contains points of both A and B.
- P3. $\mathfrak{c}(M_{\text{opt}}) \in \left[\frac{3\sqrt{d}n}{\varepsilon}, \frac{9\sqrt{d}n}{\varepsilon}\right]$.

Our goal is to compute an ε-approximate matching of A and B satisfying (P1)–(P3) in $n \cdot (\varepsilon^{-1} \log n)^{O(d)}$ time in the worst case.

After the preprocessing, the algorithm works in rounds, each of which increases the size of the matching by one. The algorithm maintains the following *cycle invariant* (cf. Corollary 5.4):

- CI. $\alpha_{\bar{\varepsilon}}(\Gamma) \geq 0$ for every alternating cycle Γ at the beginning of each round.

Each round of the algorithm consists of two steps. The first step computes an augmenting path Π such that $\alpha_{\underline{\varepsilon}}(\Pi) \leq \alpha^*_{\bar{\varepsilon}}$ (recall that $\bar{\varepsilon} = O(\underline{\varepsilon} \log n)$, so we only compute an approximate min-adjusted-cost augmenting path). The second step updates M by augmenting it by Π. After augmentation, the matching may violate the cycle invariant CI, so to reinstate the invariant the algorithm finds and cancels a sequence of simple reducing cycles $\Gamma_1, \ldots, \Gamma_k$ by updating $M \leftarrow (((M \oplus \Gamma_1) \oplus \Gamma_2) \oplus \cdots)$. To perform these steps efficiently, we design a data structure, described in Sections 3 and 4, that maintains $\vec{G}_M$ and supports the following two operations:

- A1. FIND-PATH(): Returns an augmenting path Π with $\alpha_{\underline{\varepsilon}}(\Pi) \leq \alpha^*_{\bar{\varepsilon}}$.
- A2. AUGMENT(Π, M): Takes an augmenting path Π and the current matching M as input. First updates M to $M \oplus \Pi$, then identifies and cancels a sequence of simple reducing cycles $\Gamma_1, \ldots, \Gamma_k$.

Section 5 presents details of these operations, and Section 6 proves the correctness of the algorithm and analyzes its running time.

3 OVERALL DATA STRUCTURE

In this section we describe the overall data structure that maintains the current matching M and residual graph $\vec{G}_M$, and that supports FINDPATH and AUGMENT. The data structure constructs a hierarchical covering of $\mathbb{R}^d$ (an instance of the *tree cover* in $\mathbb{R}^d$ [18]) by overlapping hypercubes, called *cells*, which is fixed and independent of M. For each cell $\boxplus$, it maintains a weighted, directed graph $G_\boxplus$ that depends on M which can be viewed as a compressed representation of the subgraph of $\vec{G}_M$ of size $\text{poly}(\varepsilon^{-1} \log n)$ induced by $(A \cup B) \cap \boxplus$. The data structure detects negative cycles in $G_\boxplus$ and maintains shortest paths between pairs of nodes in $G_\boxplus$ if there are no negative cycles. For each cell $\boxplus$, it also uses an auxiliary data structure that maps a negative cycle or "augmenting path" π in $G_\boxplus$

to a simple (non-self-intersecting) negative cycle or augmenting path Π such that $\alpha_\varepsilon(\Pi)$ is at most the weight of π in $G_\boxplus$; we refer to this map as an *expansion* of π in $\vec{G}_M$. Path Π can be reported in time proportional to $|\Pi|$. The data structure maintains a priority queue $\mathcal{Q}$ that stores the cheapest "augmenting path" of each $G_\boxplus$.

Hierarchical covering and compressed graph. Let $\mathfrak{G}_0$ be the d-dimensional integer grid, that is, the collection of hypercubes $[0,1]^d+\mathbb{Z}^d$. We build a hierarchy of ever coarser *grids* $\mathfrak{G}_1, \ldots, \mathfrak{G}_{\log \Delta}$. Set $\ell_i := 2^i$. A hypercube in $\mathfrak{G}_i$ has side-length ℓ_i, and is obtained by merging 2^d smaller hypercubes of the grid $\mathfrak{G}_{i-1}$ in the previous level. For each i, we build a collection of *cells* at level i as follows: For any hypercube $\boxplus \in \mathfrak{G}_i$, we create 2^d cells at level i, namely, $\boxplus + \ell_{i-1}(b_1, \ldots, b_d)$ for all d-bits $b_1, \ldots, b_d \in \{0,1\}$, each corresponds to a shifting of $\boxplus$ by either $+0$ or $+\ell_{i-1}$ in each dimension; the cells at level i is the union of 2^d different translations $\mathfrak{G}_i + \ell_{i-1}(b_1, \ldots, b_d)$ of $\mathfrak{G}_i$. Every cell at level i has its boundary aligning with the grid boundaries of $\mathfrak{G}_{i-1}, \mathfrak{G}_{i-2}, \ldots, \mathfrak{G}_0$. As a consequence, cells are always perfectly tiled by lower-level grids, and cells at level j are a refinement of those at level i for all $j < i$. The *children* of $\boxplus$ at level i are the 3^d cells at level $i-1$ contained in $\boxplus$. We denote the set of children cells of $\boxplus$ as $\mathrm{Ch}(\boxplus)$.

Next, we describe the construction of compressed graphs. We fix two parameters: *height* $h := c_3 \lceil \log n \rceil$ and *penalty* $\delta := c_4\varepsilon = \Theta(\frac{\varepsilon}{\log n})$, where $c_3, c_4 > 0$ are constants. We translate the points slightly along each coordinate, say, by $\frac{\delta}{8\sqrt{d}}$, so that each point lies in the interior of a grid cell in $\mathfrak{G}_i$ for any $i > \lceil \log \frac{\delta}{4\sqrt{d}} \rceil$. Let $\mathcal{C}_i := \{\text{level-}i \text{ cell } \boxplus \mid \boxplus \cap (A \cup B) \neq \emptyset\}$ be the set of nonempty level-i cells for each i; set $\mathcal{C} = \bigcup_{i=1}^{h} \mathcal{C}_i$. For each $\boxplus \in \mathcal{C}$ we construct a weighted directed bipartite graph $G_\boxplus = (V_\boxplus, E_\boxplus)$ called the *compressed graph*.

Clustering and nodes of $G_\boxplus$. Each cell $\boxplus \in \mathcal{C}_i$ is partitioned into *subcells* by the hypercubes of $\mathfrak{G}_{i-\tau}$, where $\tau := \lceil \log_2(4\sqrt{d}/\delta) \rceil$ is the *level-offset*.[2] The diameter of each subcell is at most $\frac{\delta}{4}\ell_i$. For each subcell $\square$ of $\boxplus$, let $A_\square := A \cap \square$ and $B_\square := B \cap \square$. We refer to $A_\square, B_\square$ as the *A-clusters* and *B-clusters* respectively. We call a cluster $A_\square$ or $B_\square$ *unsaturated* if at least one of its points is free; otherwise we call it *saturated*. Set $\mathcal{A}_\boxplus$ and $\mathcal{B}_\boxplus$ to be the collections of nonempty A- and B-clusters of $\boxplus$, respectively. The set of nodes of $G_\boxplus$ is $V_\boxplus := \mathcal{A}_\boxplus \cup \mathcal{B}_\boxplus$; $s := |V_\boxplus| = (\varepsilon^{-1} \log n)^{O(d)}$. We note that each cell $\boxplus \in \mathcal{C}_0$ either contains only points of A or points of B. For level $i \geq 1$, let $\square$ be a subcell of $\boxplus$. Then $A_\square = \bigcup_{\Delta \in \mathrm{Ch}(\square)} A_\Delta$ and $B_\square = \bigcup_{\Delta \in \mathrm{Ch}(\square)} B_\Delta$, where $\mathrm{Ch}(\square)$ are the *children subcells* of $\square$. Hence, a cluster of $\boxplus$ is obtained by merging the at most 2^d clusters of children of $\boxplus$ that contain $\square$.

Arcs, compressed paths, and expansions. Graph $G_\boxplus$ is a directed complete bipartite graph with arc set $E_\boxplus := \mathcal{A}_\boxplus \times \mathcal{B}_\boxplus \cup \mathcal{B}_\boxplus \times \mathcal{A}_\boxplus$. There are two types of arcs in $E_\boxplus$: an arc between two clusters $A_\square$ and $B_{\square'}$ is a *bridge arc* if no child cell of $\boxplus$ contains both of $\square$ and $\square'$; otherwise it is an *internal arc*.

We will soon define a recursive weight function $w_\boxplus$ on the arcs in $E_\boxplus$. Using this weight function, we compute a minimum-weight path $\pi_\boxplus(X,Y)$ between all pairs of nodes/clusters (X,Y) in $G_\boxplus$; we refer to $\pi_\boxplus(X,Y)$ as a *compressed path*. For every pair $(X,Y) \in E_\boxplus$, we also compute a simple path in $\vec{G}_M$ that begins at a point in X and ends at a point in Y, which we call the *expansion* of $\pi_\boxplus(X,Y)$, denoted by $\Phi_\boxplus(X,Y)$. We refer to the endpoints of $\Phi_\boxplus(X,Y)$ as *tips*. If $(X,Y) = (B_\square, A_{\square'})$, then the first and last edges of $\Phi_\boxplus(B_\square, A_{\square'})$ are matching edges and the tips are never changed. On the other hand, if $(X,Y) = (A_\square, B_{\square'})$ then the first and last edges of $\Phi_\boxplus(B_\square, A_{\square'})$ are non-matching edges and we may change the tips as needed. That is, if the first (resp. last) edge of $\Phi_\boxplus(A_\square, B_{\square'})$ is (a,b) and $a' \in A_\square$ (resp. $b' \in B_{\square'}$) is another point, then we may replace (a,b) with (a',b) (resp. (a,b')).

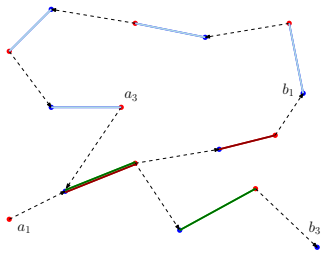

Figure 1: Concatenation of three expansions Φ_1 (red), Φ_2 (blue), Φ_3 (green); solid edges are matching edges. The ending (starting) tip of Φ_1 (Φ_3) is changed to connect it to Φ_2. Φ_1 and Φ_3 share a matching edge.

The asymmetry in the definition of $\Phi_\boxplus(B_\square, A_{\square'})$ and $\Phi_\boxplus(A_\square, B_{\square'})$ allows us to concatenate the expansion of two shortest paths in $G_\boxplus$ that share a node. Suppose $\Phi_1 = \Phi_\boxplus(B_{\square_1}, A_{\square_2})$, $\Phi_2 = \Phi_\boxplus(A_{\square_2}, B_{\square_3})$, and b_1 and a_1 the starting and ending tips of Φ_1, respectively. We change the starting tip of Φ_2 to a_1. By concatenating this modified Φ_2 with Φ_1, we obtain an alternating path in $\vec{G}_M$ from the starting tip of Φ_1 to the ending tip of Φ_2 (see Figure 1). For brevity, we will simply write this concatenation as $\Phi_1 \circ \Phi_2$ in the rest of the text. If $\square_3 = \square_1$, i.e., $\pi_\boxplus(B_{\square_1}, A_{\square_2}) \circ \pi_\boxplus(A_{\square_2}, B_{\square_3})$ is a cycle in $G_\boxplus$, then by *cyclically concatenating* Φ_1 and Φ_2, i.e, changing the starting tip of Φ_2 to a_1 and the ending tip of Φ_2 to b_1, $\Phi_1 \circ \Phi_2$ is an alternating cycle in $\vec{G}_M$.

There is a mutual recursion between the definition of arc weights and the construction of expansions: arc weights are defined using the residual cost of lower-level expansions, and expansions are constructed based on shortest paths $\pi_\boxplus(X,Y)$, described below.

Stability. During both the shortest path computation on $G_\boxplus$ and the construction of expansions we may discover a reducing cycle Γ in $\vec{G}_M$, in which case we update M by canceling Γ (i.e., setting $M \leftarrow M \oplus \Gamma$) and rebuilding the data structure at $\boxplus$ and its descendants (see Repair in Section 5). We say cell $\boxplus$ is *stable* if (i) the children of $\boxplus$ are stable, and (ii) no reducing cycle was detected while computing $\pi_\boxplus(\cdot,\cdot)$ and $\Phi_\boxplus(\cdot,\cdot)$ for all pairs of nodes in $G_\boxplus$.

Arc weights. Assuming all children cells of $\boxplus$ are stable, we define *arc weights* $w_\boxplus : E_\boxplus \to \mathbb{R}$ recursively as follows:

- **Bridge arcs**: Let (X,Y) be a bridge arc. Let cntr_X (resp. cntr_Y) be the center of the subcell containing the cluster X (resp. Y). For $(X,Y) = (A_\square, B_{\square'})$, we set

$$w_\boxplus(X,Y) := \|\mathrm{cntr}_X - \mathrm{cntr}_Y\| + \delta\ell_i \tag{1}$$

[2] In this paper, the base of log is always 2.

if $X \times Y \setminus M \neq \emptyset$ and ∞ otherwise. For $(X, Y) = (B_\square, A_{\square'})$,

$$w_\boxplus(X, Y) := -\|\text{cntr}_X - \text{cntr}_Y\| + \delta\ell_i \tag{2}$$

if $(X \times Y) \cap M \neq \emptyset$ and ∞ otherwise.

- **Internal arcs**: If a child $\boxplus'$ of $\boxplus$ contains both $\square$ and $\square'$, then we set

$$w_\boxplus(X_\square, Y_{\square'}) := \min_{\substack{\Delta \in \text{Ch}(\square), \Delta' \in \text{Ch}(\square') \\ \boxplus' \in \text{Ch}(\boxplus): \square, \square' \subset \boxplus'}} \alpha_{\bar{\varepsilon}}(\Phi_{\boxplus'}(X_\Delta, Y_{\Delta'})) + \delta\ell_i. \tag{3}$$

We call the triple $(\boxplus', \Delta, \Delta')$ that realizes the minimum in (3) as the *weight certificate* of the internal arc $(X_\square, Y_{\square'})$.

Expansions. We now define the expansions $\Phi_\boxplus(X, Y)$ using the weights of arcs in $G_\boxplus$ and their weight certificates. We first define an expansion $\Phi(\eta)$ of each arc $\eta \in E_\boxplus$. If η is a bridge arc of the form $\eta = (A_\square, B_{\square'})$, then we choose $\Phi(\eta)$ to be an arbitrary pair $(a, b) \in A_\square \times B_{\square'}$ with the caveat that if $A_\square$ or $B_{\square'}$ has a free point then we choose a or b to be a free point; and if it is a bridge arc of the form $\eta = (B_\square, A_{\square'})$ then we choose $\Phi(\eta) = \arg\max_{e \in M \cap (B_\square \times A_{\square'})} \|e\|$. Finally, if η is an internal arc with $(\boxplus', \Delta, \Delta')$ as the weight certificate of η, then we set $\Phi(\eta) := \Phi_{\boxplus'}(X_\Delta, Y_{\Delta'})$.

For a given pair $(X, Y) \in E_\boxplus$, we compute $\Phi_\boxplus(X, Y)$ as follows: Suppose $\pi_\boxplus(X, Y) = \langle \eta_1, \ldots, \eta_k \rangle$, where each η_i is an arc of $G_\boxplus$. Let $\tilde{\Phi} \leftarrow \Phi(\eta_1) \circ \cdots \circ \Phi(\eta_k)$, where concatenation is as defined above. Albeit each $\Phi(\eta_i)$ being a simple path, $\tilde{\Phi}$ may not be simple, i.e., some matching edges may appear more than once, so we cannot simply set $\Phi_\boxplus(X, Y)$ to $\tilde{\Phi}$. Instead, we use a greedy algorithm to simplify $\tilde{\Phi}$ by repeatedly removing cycles, as follows: We check whether any matching edge in $\tilde{\Phi}$ appears more than once. If there is no such edge, then we stop and set $\Phi_\boxplus(X, Y) \leftarrow \tilde{\Phi}$. Otherwise, suppose $\tilde{\Phi} = \Pi_1 \circ \langle e \rangle \circ \Pi_2 \circ \langle e \rangle \circ \Pi_3$, where e is a matching edge. Then $C := \langle e \rangle \circ \Pi_2$ is an alternating cycle in $\vec{G}_M$ (see Figure 1). If $\alpha_{\bar{\varepsilon}}(C) < 0$, i.e., C is a reducing cycle, then we compute a simple reducing subcycle of C using the procedure SimpleReducingSubcycle described below. Otherwise we set $\tilde{\Phi} \leftarrow \Pi_1 \circ \langle e \rangle \circ \Pi_3$, and repeat the above step. The correctness of the overall algorithm does not depend on the order in which we find the duplicate matching edges. We refer to this procedure of computing $\Phi_\boxplus(X, Y)$ from $\pi_\boxplus(X, Y)$ as ConstructExpansion. Maintaining expansions in a data structure so that they can be computed efficiently is a major component of our algorithm and is described in detail in Section 4.

The following lemmas summarize the relationship between the adjusted cost of paths in $\vec{G}_M$ and the weights of paths in $G_\boxplus$.

Lemma 3.1 (Expansion Inequality). *Let $\boxplus$ be a stable cell, let $(X, Y) \in E_\boxplus$. Then $\alpha_{\bar{\varepsilon}}(\Phi_\boxplus(X, Y)) \leq w_\boxplus(\pi_\boxplus(X, Y))$.*

Corollary 3.2. *Suppose all children of $\boxplus$ are stable. Let $C := \langle \eta_1, \ldots, \eta_k \rangle \subseteq E_\boxplus$ be a negative cycle in $G_\boxplus$, and let $\Gamma := \Phi(\eta_1) \circ \cdots \circ \Phi(\eta_k)$ be the alternating cycle formed by cyclically concatenating the expansions of $\eta_1, \ldots, \eta_k$. Then, $\alpha_{\bar{\varepsilon}}(\Gamma) < 0$.*

Lemma 3.3. *Let Π be either a reducing cycle, or the augmenting path (in $\vec{G}_M$) with the minimum $\bar{\varepsilon}$-adjusted cost α^*. Then there exists a level $i \in \{0, \ldots, h\}$ and $\boxplus \in \mathcal{C}_i$ such that Π lies completely in $\boxplus$.*

Lemma 3.4 (Lifting Inequality). *Let Π be an alternating path in $\vec{G}_M$ from point p to point q (possibly $p = q$ in the case Π is a cycle) that is completely contained in a cell of level at most h. Let $\boxplus$ be a cell at the smallest level that contains Π and X (resp. Y) the cluster in $V_\boxplus$ containing p (resp. q). Then either $\boxplus$ is not stable or $w_\boxplus(\pi_\boxplus(X, Y)) \leq \alpha_{\bar{\varepsilon}}(\Pi)$.*

Putting everything together. In view of the above lemmas, the data structure works as follows: If $\boxplus$ is not stable (but all its children are), then the data structure returns a reducing cycle $\Gamma_\boxplus$ in $\vec{G}_M$ that acts as a witness of its instability. If $\boxplus$ is stable, then for every pair $(X, Y) \in E_\boxplus$, the data structure maintains a minimum-weight path $\pi_\boxplus(X, Y)$ and its expansion $\Phi_\boxplus(X, Y)$. If both $A_\square, B_{\square'}$ are unsaturated, then we call $\pi_\boxplus(A_\square, B_{\square'})$ an *augmenting path* in $G_\boxplus$. Given an augmenting path $\pi_\boxplus(A_\square, B_{\square'})$ in $G_\boxplus$, our choice of the expansion of a bridge arc ensures that $\Phi_\boxplus(A_\square, B_{\square'})$ is an augmenting path in $\vec{G}_M$. Let $(\square_\boxplus, \square'_\boxplus) := \arg\min_{\square, \square'} \alpha_{\bar{\varepsilon}}(\Phi_\boxplus(A_\square, B_{\square'}))$ where the minimum is taken over all cluster pairs $A_\square, B_{\square'}$ of $\boxplus$ such that both are unsaturated. If $G_\boxplus$ does not have any augmenting path, the pair $(\square_\boxplus, \square'_\boxplus)$ is undefined. Set $\mathit{OptPairs} := \{(\square_\boxplus, \square'_\boxplus) \mid \boxplus \in \mathcal{C}\}$. We store *OptPairs* in a priority queue with $\alpha_{\bar{\varepsilon}}(\Phi_\boxplus(\square_\boxplus, \square'_\boxplus))$ as the key of $(\square_\boxplus, \square'_\boxplus)$.

FindPath and Augment procedures are described in Section 5. The information stored at $\boxplus$ depends on $M \cap ((A \cap \boxplus) \times (B \cap \boxplus))$. So whenever the matching edges change (e.g. by Augment), we update the information stored at the corresponding $\boxplus$. The Repair($\boxplus$) procedure, also described in Section 5, updates the data structure at $\boxplus$. Initially $M = \emptyset$, and the data structure can be built by calling Repair at all cells of $\mathcal{C}$ in a bottom-up manner.

4 MAINTAINING EXPANSIONS OF COMPRESSED PATHS

We now describe the algorithms and data structure for computing and maintaining the expansions of compressed paths at all cells. We begin by describing a high-level representation of expansions and the two high-level procedures ConstructExpansion and SimpleReducingSubcycle needed for computing them. Next, we describe the data structure to maintain the expansions compactly. Finally, we describe the Intersect, Report, and AdjCost procedures as well as the operations on the data structure needed by the high-level procedures.

Recall that cells in $\mathcal{C}$ are processed in a bottom up manner. We focus on computing expansions at a cell $\boxplus$, assuming (i) they have been computed at all children cells of $\boxplus$, (ii) the children of $\boxplus$ are stable, (iii) compressed paths between all pairs of subcells of $\boxplus$ have been computed, and (iv) no negative cycles have been detected in $G_\boxplus$. For an internal arc $\gamma = (A_\square, B_{\square'})$ at $\boxplus$, the expansion of γ, $\Phi(\gamma)$, is $\Phi_{\boxplus'}(A_\Delta, B_{\Delta'})$ for some child cell $\boxplus'$ of $\boxplus$ and children clusters $A_\Delta, B_{\Delta'}$ of $A_\square, B_{\square'}$, where $(\boxplus', \Delta, \Delta')$ is the weight certificate of γ. We have $\Phi(\gamma)$ at our disposal when we compute expansions at $\boxplus$.

4.1 Expansions and Pathlets

Representation of expansions. Let $\langle b_1, a_1, \ldots, b_k, a_k \rangle$ be a path in $\vec{G}_M$ from $b_1 \in B$ to $a_k \in A$, or a cycle (with the edge (a_k, b_1) also being present). The path (cycle) can be specified by the sequence $\langle (b_1, a_1), \ldots, (b_k, a_k) \rangle$ of matching edges. Conversely, since $(a, b) \in \vec{G}_M$ if $(b, a) \notin M$ for any pair $(a, b) \in A \times B$, any sequence $\Phi := \langle e_1, e_2, \ldots, e_k \rangle$ of matching edges defines a path $\langle b_1, a_1, \ldots, b_k, a_k \rangle$ in $\vec{G}_M$, where $e_i = (b_i, a_i)$. Φ also defines a unique cycle in $\vec{G}_M$. If

the first or the last edge of an alternating path is a non-matching edge, then the path can be represented by its sequences of matching edges and its tips. For example, a path $\langle a_0, b_1, a_1, \ldots, b_k, a_k, b_{k+1}\rangle$ can be represented by the edge sequence $\langle (b_1, a_1), \ldots, (b_k, a_k)\rangle$ and the starting and ending tips a_0, b_{k+1}, respectively. We represent an alternating paths/cycle Π as a tuple called the *compact representation*, denoted by $\langle\Pi\rangle$, comprised of:

- a sequence $\langle e_1, e_2, \ldots, e_k\rangle$ of edges in M, which we refer to as the *matching edge sequence* (MES); this sequence is empty if Π consists of a single non-matching edge;
- a *flag* which is set to 1 if Π is a cycle; and
- the *tips* if the first and/or last edge of Π is non-matching (otherwise *null*); tips are always *null* if π is a cycle.

Pathlets. For any MES $\Phi = \langle e_1, \ldots, e_k\rangle$ and two indices $i \le j$, we define the *splice* operators $e_i \blacktriangleright \Phi \blacktriangleleft e_j := \langle e_i, e_{i+1}, \ldots, e_{j-1}, e_j\rangle$ and $e_i \triangleright \Phi \triangleleft e_j := \langle e_{i+1}, e_{i+2}, \ldots, e_{j-2}, e_{j-1}\rangle$. We mix the inclusive and exclusive splices when convenient, e.g., $e_i \blacktriangleright \Phi \triangleleft e_j = \langle e_i, \ldots, e_{j-1}\rangle$. We call any contiguous subsequence of Φ a *pathlet* of Φ. Obviously, Φ is a pathlet of itself. If ϕ is a pathlet of Φ, we also allow further restriction of ϕ using splicing, e.g., to create new pathlets $\phi' \subseteq \phi$ of Φ. We say that ϕ *originates* from Φ, and we use "pathlet of an expansion" as shorthand for "pathlet of the MES of an expansion." The pathlet of an expansion is always simple, since an expansion's MES is simple. Later, we compose new MES by concatenating multiple (individually) simple pathlets — these concatenated sequences need not be simple. Two pathlets *intersect* if they have a common matching edge.

Although the compact representation and pathlets can be applied to any path/cycle in $\vec{G}_M$, we will deal mainly with those of three types of paths: (i) an expansion, (ii) pathlets of an expansion, and (iii) concatenations of pathlets of expansions. We work mostly with the MES except when we need to compute the adjusted cost (i.e., tips matter), so with a slight abuse of notation we will not distinguish between an expansion and its MES and use Φ to denote both. Note that (i) the MES of $\Phi_1 \circ \Phi_2$ is the concatenation of MES's of Φ_1, Φ_2, and (ii) an alternating path is non-simple if and only if its MES contains duplicate (matching) edges. These two simple observations will be crucial for our data structure.

Operations on pathlets. MES's and pathlets are maintained using a data structure, which consists of a collection of trees, called MES trees and pathlet trees, respectively, and Boolean look-up tables, described below in Section 4.3. This data structure supports the following operations. Here we assume that a pathlet is represented compactly using $O(\log n)$ size (see Section 4.3).

- Intersects(ϕ_1, ϕ_2): Given two pathlets ϕ_1 and ϕ_2 that originate from expansions of descendant cells of $\boxplus$, report whether $\phi_1 \cap \phi_2 \neq \emptyset$.
- LastCommonEdge(ϕ_1, ϕ_2): Given two pathlets ϕ_1 and ϕ_2 that originate from lower-level expansions where $\phi_1 \cap \phi_2 \neq \emptyset$, return *references* to four edges[3]: e_1, the last edge of ϕ_1 in the common intersection; e_2, the copy of e_1 in ϕ_2; e_3, the predecessor of e_1 in ϕ_1 (maybe *null*); e_4, the predecessor of e_2 in ϕ_2 (maybe *null*).
- Median(ϕ): Given a pathlet ϕ, return a reference to the median edge of ϕ. That is, given $\phi := \langle e_1, e_2, \ldots, e_k\rangle$, return a reference to $e_{\lceil k/2\rceil}$.
- AdjCost$(\langle\Pi\rangle)$: Given a possibly non-simple path $\Pi \in \vec{G}_M$ in its compact representation, where its MES is the concatenation of up to s pathlets, return $\alpha_{\xi}(\Pi)$.
- Report$(\langle\Pi\rangle)$ Given the compact representation of a simple path/cycle $\Pi \in \vec{G}_M$, with its MES being the concatenation of at most s pathlets, return the sequence of edges in Π.
- Splice$(\phi, e, \bowtie)$: Given a pathlet ϕ, an edge e of ϕ, and $\bowtie \in \{\blacktriangleright, \blacktriangleleft, \triangleleft\}$, splice the pathlet ϕ at e.
- Concatenate$(\phi_1, \phi_2, \ldots, \phi_t)$: Construct an MES composed of $\phi_1 \circ \phi_2 \circ \cdots \circ \phi_t$.

Let $t(n)$ denote the maximum time taken by the above procedures except Report. We will see below in Section 4.4 that $t(n) = (\varepsilon^{-1}\log n)^{O(d)}$ and Report takes $O(t(n) + kh)$ time where k is the length of the path returned by the procedure.

4.2 High-level Simplification Procedures

We now describe the two main procedures ConstructExpansion and SimpleReducingSubcycle.

ConstructExpansion. Let $\pi_{\boxplus}(X, Y) = \langle \eta_1, \ldots, \eta_k\rangle$ be a shortest path in $G_{\boxplus}$, where each η_i is an arc of $G_{\boxplus}$. We compute the MES of $\Phi_{\boxplus}(X, Y)$ and set the starting (resp. ending) tip of $\Phi_{\boxplus}(X, Y)$ to that of $\Phi(\eta_1)$ (resp. $\Phi(\eta_k)$). Let ϕ_i be the MES of $\Phi(\eta_i)$, for $1 \le i \le k$. We initially set

$$\tilde{\Phi} \leftarrow \phi_1 \circ \cdots \circ \phi_k. \tag{4}$$

We process the ϕ_i's in sequence and grow a pathlet sequence $\Xi = \langle \phi'_1, \ldots, \phi'_t\rangle$ where $\phi'_1 \circ \cdots \circ \phi'_t$ is simple.

Initially, $\Xi = \emptyset$ and the first pathlet we process is ϕ_1. To process ϕ_i, we compare it against each $\phi'_j \in \Xi$ in ascending order to determine whether $\phi'_0 \circ \cdots \circ \phi'_t \circ \phi_i$ is simple. Specifically, we query Intersects(ϕ_i, ϕ'_j) to find the first pathlet $\phi'_j \in \Xi$ that shares a matching edge with ϕ_i. If there is no intersection with any pathlet in Ξ, we simply set $\phi'_{t+1} \leftarrow \phi_i$, append ϕ'_{t+1} to Ξ, and continue on to ϕ_{i+1}. If there is an intersection against ϕ'_j, we find e_1, the *last* edge in ϕ_i that intersects ϕ'_j, by invoking LastCommonEdge(ϕ_i, ϕ'_j). Then, $\phi'_1 \circ \cdots \circ \phi'_{j-1} \circ (\phi'_j \triangleleft e_1) \circ (e_1 \blacktriangleright \phi_i)$ is a simple sequence of matching edges and

$$C := (e_1 \blacktriangleright \phi'_j) \circ \phi'_{j+1} \circ \cdots \circ \phi'_t \circ (\phi_i \triangleleft e_1)$$

is a cycle.[4] Next, compute $\alpha_{\xi}(C)$ using AdjCost$(\langle C\rangle)$. If C is reducing, we abort ConstructExpansion and return the simple reducing cycle $\hat{C}$ by calling SimpleReducingSubcycle(C). If C is not reducing, then we update $\phi'_j = (\phi'_j \triangleleft e_1)$, $\phi'_{j+1} = (e_1 \blacktriangleright \phi_i)$, and set $\Xi \leftarrow \langle \phi'_1, \ldots, \phi'_j, \phi'_{j+1}\rangle$. Then, we continue on to ϕ_{i+1}. If all elements of $\tilde{\Phi}$ are processed without aborting and $\Xi = \langle \phi'_0, \ldots, \phi'_t\rangle$, then we return the simple sequence $\phi'_1 \circ \phi'_2 \circ \cdots \circ \phi'_t$ as the MES of $\Phi_{\boxplus}(X, Y)$.

We assume that each input pathlet ϕ_i is represented as a pathlet tree; intermediate pathlets are also maintained using pathlet trees;

[3] We will explain the specific form of the references to these edges when we describe the data structure. We use the references to e_1, e_2 (resp. e_3, e_4) to implement inclusive (resp. exclusive) splices involving e_1 in both ϕ_1 and ϕ_2. None of the procedures will be invoking left exclusive splice operation, i.e., of the form $e \triangleright \phi$, so we do not need references to the successor edges.

[4] Note that the exclusive splices (e.g., $\phi'_j \triangleleft e_1$) may be empty pathlets. If that is the case, we simply drop the empty pathlet from the concatenation sequence.

pathlets are spliced using Splice; and Concatenate$(\phi'_1, \ldots, \phi'_t)$ is called at the end to represent the MES as an MES tree.

SimpleReducingSubcycle. This procedure is either called by ConstructMES or by Repair to expand a negative compressed cycle φ in $G_{\boxplus}$. In the latter case, we first construct an intermediate expansion of φ of the form (4) as above, so assume that (MES of) the reducing cycle is represented as a pathlet sequence $C = \langle \phi_1, \ldots, \phi_k \rangle$. Like ConstructExpansion, the general case of this procedure processes the pathlets of the input in sequence while building a simple prefix. Given a reducing cycle (as a pathlet sequence) C, we grow a pathlet sequence $\Xi = \langle \phi'_1, \ldots, \phi'_t \rangle$ where $\phi'_1 \circ \cdots \circ \phi'_t$ is simple. The case of $k \leq 2$ is a base case, handled differently, so assume first that $k \geq 3$.

Initially $\Xi = \emptyset$, and the first element we process is ϕ_1. Suppose we are processing ϕ_i in C. For each $\phi'_j \in \Xi$ in reverse order, we query Intersects(ϕ_i, ϕ'_j). If we fail to find any intersections between Ξ and ϕ_i, then $\phi'_1 \circ \cdots \phi'_t \circ \phi_i$ is simple, so we set $\phi'_{t+1} \leftarrow \phi_i$, append ϕ'_{t+1} to Ξ, and continue on to ϕ_{i+1}. Otherwise, let ϕ'_j be the last element of Ξ to intersect ϕ_i, and we invoke LastCommonEdge(ϕ_i, ϕ'_j) to acquire e_1, the *last* edge of ϕ_i in the intersection. There are two subcycles about e_1:

$$C_0 \coloneqq (e_1 \blacktriangleright \phi'_j) \circ \phi'_{j+1} \circ \cdots \circ \phi'_{i-1} \circ (\phi_i \blacktriangleleft e_1)$$
$$C_I \coloneqq \phi'_1 \circ \cdots \circ (\phi'_j \blacktriangleleft e_1) \circ (e_1 \blacktriangleright \phi_i) \circ \phi_{i+1} \circ \cdots \circ \phi_t$$

Neither C_0 nor C_I need be simple, but:

- In C_0, the only potentially intersecting pathlets are $e_1 \blacktriangleright \phi'_j$ and $\phi_i \blacktriangleleft e_1$.
- In C_I, $(\phi'_j \blacktriangleleft e_1) \cap (e_1 \blacktriangleright \phi_i) = \emptyset$, so the number of pairs of intersecting component pathlets in C_I is strictly less than the number in C.

Since C was reducing, at least one of C_0 and C_I must be reducing (the adjusted cost of C is simply the sum of adjusted costs of C_0 and C_I). Check $\alpha_{\underline{\varepsilon}}(C_0)$ using AdjCost$(\langle C_0 \rangle)$. If C_0 is reducing, we return the result of running the base-case algorithm on C_0, below. If C_0 is not reducing, then C_I is reducing and we return the result of recursively calling SimpleReducingSubcycle(C_I). If all elements of C are processed without finding an intersection, then C is simple and we return C.

There are two base cases. First, if C contains only one pathlet, then it is simple since the originating MES must also be simple. Next, if at most two of the component pathlets of C are intersecting, we use the following binary-search algorithm:

Given $C \coloneqq \phi_1 \circ \cdots \circ \phi_k$ where only ϕ_1 and ϕ_k intersect, the *only-two-intersecting* pathlets base-case is handled using a prune-and-search approach as follows. Let $\phi_1 \coloneqq e_L \blacktriangleright \Phi_1 \blacktriangleleft e^+$ and $\phi_k \coloneqq e^- \blacktriangleright \Phi_k \blacktriangleleft e_R$ for two edges e_L, e_R (initially, $e_L = e_R$), and keep an extra pointer $e_M = e_L \in \phi_1$. We recursively shrink ϕ_1 and ϕ_2 until we find a simple reducing cycle, while keeping the invariant that there are no intersections between $(e_L \blacktriangleright \Phi_1 \blacktriangleleft e_M)$ and $(e^- \blacktriangleright \Phi_k \blacktriangleleft e_R)$ At each step, we query Intersects$((e_M \blacktriangleright \Phi_1 \blacktriangleleft e^+), (e^- \blacktriangleright \Phi_k \blacktriangleleft e_R))$. If there is no intersection then C is simple, so we return C. If there is an intersection, then we invoke Median$(e_M \blacktriangleright \Phi_1 \blacktriangleleft e^+)$ to acquire f, the median edge of $e_M \blacktriangleright \Phi_1 \blacktriangleleft e^+$, and query Intersects$((e_M \blacktriangleright \Phi_1 \blacktriangleleft f), (e^- \blacktriangleright \Phi_k \blacktriangleleft e_R))$. If there is no intersection up to the median, then we shrink the left pathlet to $f \blacktriangleright \Phi_1 \blacktriangleleft e^+$ by setting $e_M \leftarrow f$, and attempt again. Otherwise, if $(e_M \blacktriangleright \Phi_1 \blacktriangleleft f) \cap (e^- \blacktriangleright \Phi_k \blacktriangleleft e_R) \neq \emptyset$, we invoke LastCommonEdge$((e_L \blacktriangleright \Phi_1 \blacktriangleleft f), (e^- \blacktriangleright \Phi_k \blacktriangleleft e_R))$ to acquire e_1, the *last* edge of $e_L \blacktriangleright \Phi_1 \blacktriangleleft f$ that appears in the intersection. There are two subcycles about e_1:

$$\widetilde{C}_0 \coloneqq (e_L \blacktriangleright \Phi_1 \blacktriangleleft e_1) \circ (e_1 \blacktriangleright \Phi_k \blacktriangleleft e_R)$$
$$\widetilde{C}_I \coloneqq (e_1 \blacktriangleright \Phi_1 \blacktriangleleft e^+) \circ \phi_2 \circ \cdots \circ \phi_{k-1} \circ (e^- \blacktriangleright \Phi_k \blacktriangleleft e_1)$$

In $\widetilde{C}_0$, $|e_L \blacktriangleright \Phi_1 \blacktriangleleft e_1| < |\phi_1|/2$. In $\widetilde{C}_I$, $e_1 \blacktriangleright \Phi_1 \blacktriangleleft e^+$ has no intersection edges before f and $|f \blacktriangleright \Phi_1 \blacktriangleleft e^+| \leq |\phi_1/2|$. Check $\alpha_{\underline{\varepsilon}}(\widetilde{C}_0)$ using AdjCost$(\langle \widetilde{C}_0 \rangle)$. If $\widetilde{C}_0$ is reducing, we restart the binary search with input $\widetilde{C}_0$. If $\widetilde{C}_0$ is not reducing then $\widetilde{C}_I$ must be reducing, and we continue the binary search on $\widetilde{C}_I$ by setting $e_L \leftarrow e_1$, $e_R \leftarrow e_1$, and $e_M \leftarrow f$. If $|\phi_1| = 1$, then $\phi_1 = \langle e_1 \rangle$ and the intersection with e_1 is eliminated in both $\widetilde{C}_0$ and $\widetilde{C}_I$, so the binary search eventually terminates.

The next lemma bounds the running time of the two procedures.

Lemma 4.1. *(i) Given a compressed path P in $G_{\boxplus}$, ConstructExpansion either returns an MES Φ of P or a simple reducing cycle in $O(s^{10}h^4)$ time. (ii) SimpleReducingSubcycle returns a simple reducing cycle in $O(s^{10}h^4)$ time.*

4.3 Compact Representation of Pathlets

We now describe the data structure to store MES's and pathlets compactly. The MES of an expansion $\Phi \coloneqq \Phi_{\boxplus}(X, Y)$ is stored in a tree $T(\Phi)$, called an *MES tree* (to be defined next), whose leaves are the matching edges of Φ, in sequence from left to right. To reference a matching edge e in $T(\Phi)$, we specify the root-leaf path to e whenever e is passed to a procedure or returned by a procedure. Let the *spine* of $T(\Phi)$ to e be the root-leaf path that ends at the leaf representing e. The *pathlet subtree* of a pathlet $\phi \coloneqq e^- \blacktriangleright \Phi \blacktriangleleft e^+$ for $e^-, e^+ \in \Phi$, denoted by $T(\Phi, \phi)$, is the subtree of $T(\Phi)$ lying between and including the two spines to e^- and e^+. Let u be a node in $T(\Phi, \phi)$; u is a node in $T(\Phi)$ as well. If u does not lie on the spines of e^- or e^+ then all children of u that appear in $T(\Phi)$ also appear in $T(\Phi, \phi)$; otherwise only those children of u in $T(\Phi)$ that lie between the spines appear in $T(\Phi, \phi)$. Hence, $T(\Phi, \phi)$ can be implicitly represented by storing a pointer to $T(\Phi)$ and spines of e^- and e^+, which takes $O(h) = O(\log n)$ space. See Figure 2.

MES trees are defined recursively. Recall that ConstructExpansion creates the MES for $\Phi \coloneqq \Phi_{\boxplus}(X, Y)$, that has the form $\Phi = \phi_1 \circ \phi_2 \circ \cdots \circ \phi_t$ for some $t \leq s$, where each ϕ_j is a pathlet of $\Phi(\eta_j)$ of an arc η_j of $G_{\boxplus}$; recall that we discard empty pathlets from the concatenation. $T(\Phi)$ consists of a root node plus t subtrees $T_1, \ldots, T_t$ from left to right, where the root of T_i is the i-th leftmost child of the root node. If ϕ_j is the pathlet of $\Phi(\eta_j)$ then $T_j = T(\Phi(\eta_j), \phi_j)$. If ϕ_j is a pathlet from a bridge arc then the T_j is a single-node tree (matching bridge arc). If η_j is an internal arc with weight certificate $(\boxplus', \Delta, \Delta')$ then $T(\Phi(\eta_j))$ is the MES tree of $\Phi_{\boxplus'}(X_\Delta, Y_{\Delta'})$, and the children of the root of T_j are a contiguous subsequence of the children of the root of $T(\Phi(\eta_j))$. Thus, the leaves of $T(\Phi)$ correspond to (matching) bridge arcs, and the internal nodes correspond to internal arcs of some descendant cells of $\boxplus$. Sometimes it will be convenient to identify a node of $T(\Phi)$ with the corresponding arc.

Canonical pathlets and canonical nodes. Let u be an internal non-root node of $T(\Phi)$. The subtree of $T(\Phi)$ rooted at u, denoted

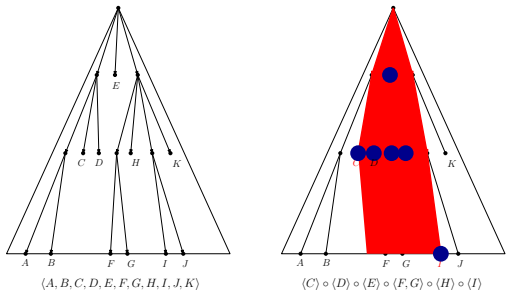

Figure 2: Left: An MES tree representing a sequence $\langle A, B, C, D, E, F, G, H, I, J, K\rangle$. Right: Pathlet-subtree (shaded) of the same MES tree representing subsequence $\langle C, D, E, F, G, H, I\rangle$. Its two spines are marked in red, and its canonical nodes are marked blue.

T_u, is a pathlet subtree $T(\Phi(\eta), \phi)$, where η is an internal arc at a descendant cell of $\boxplus$, corresponding to the node u, and ϕ is a pathlet of $\Phi(\eta)$. We call ϕ a *canonical pathlet* of $\Phi(\eta)$, denoted as ϕ_u. An MES Φ of length k has $\Theta(k^2)$ possible pathlets, but we show below that the number of canonical pathlets of an expansion is much smaller (cf. Lemma 4.4).

For a pathlet ϕ of an expansion Φ, we define the *canonical nodes* of $T(\Phi, \phi)$ to be the ordered sequence of non-spine children of spine nodes plus the leaves at the ends of the spines, denoted by $\mathrm{C}(\Phi, \phi) = \langle u_1, u_2, \ldots, u_k\rangle$. $|\mathrm{C}(\Phi, \phi)| = O(sh)$. Each canonical node u_i is a pathlet subtree of some MES tree $\Phi(\eta_i)$, i.e., represents a canonical pathlet ϕ_{u_i} (of $\Phi(\eta_i)$). We can write $\phi = \phi_{u_1} \circ \cdots \circ \phi_{u_k}$ (Figure 2) — any pathlet can be represented as the concatenation of $O(sh)$ canonical pathlets.

Compact form of MES trees. Let Φ be an MES computed by ConstructExpansion as $\Phi = \phi_1 \circ \cdots \circ \phi_t$, where pathlet ϕ_i originates from an MES Φ_i. Then the compact form of $T(\Phi)$ consists of the root of $T(\Phi)$ plus the sequence of pathlet subtrees $T(\Phi_1, \phi_1), \ldots, T(\Phi_t, \phi_t)$, each represented by a pointer to the root of the MES tree and by the (two) spines to the first and last edges. We call the root and the spine nodes of these pathlet subtrees the *exposed* nodes. We store only the compact form of $T(\Phi)$. For each exposed node u of $T(\Phi)$, we store the following auxiliary information about ϕ_u (say, originating from the expansion Φ_u):

(i) b_u and a_u, the first and last endpoints of ϕ_u;
(ii) $m_u := |\phi_u|$, which is equal to the number of leaves in T_u;
(iii) $\alpha_{\underline{\varepsilon}}(\phi_u)$, the $\underline{\varepsilon}$-adjusted cost of ϕ_u (here we view the canonical pathlet ϕ_u as a path in $\vec{G}_M$ and not just a sequence of matching edges).
(iv) Pointers to the root of $T(\Phi_u)$ and to its two children that correspond to the leftmost and the rightmost children of u in $T(\Phi)$.

Since there are up to s children of the root, the space used by this compact representation of $T(\Phi)$ is $O(sh)$. ConstructExpansion and SimpleReducingSubcycle procedures return the MES of the output alternating path/cycle in this compact form.

Lemma 4.2. *Let Φ be an MES, and let u be an unexposed child of an exposed node v in $T(\Phi)$. Suppose the canonical pathlet ϕ_u (resp. ϕ_v) originates from Φ_u (resp. Φ_v). Then the root of $T(\Phi_v, \phi_v)$ has a child u' such that (i) both u and u' are identified with the same arc, and (ii) $T(\Phi_u, \phi_u)$, which is also the subtree of $T(\Phi)$ rooted at u, is identical to the subtree of $T(\Phi_v)$ rooted at u'.*

Lemma 4.2 suggests how we can traverse $T(\Phi)$ recursively using its compact form: Suppose we are a node u of $T(\Phi)$. If u is a leaf, then we return. If u is an exposed node, then we visit the children of u using the information stored at u. Finally, if u is unexposed but its parent v is exposed, then we use Lemma 4.2 to find the (exposed) child u' of the root of $T(\Phi_v)$ that corresponds to u, and we recursively visit the subtree rooted at u' in $T(\Phi_v)$.

Cells, level, and rank. Let Φ be an expansion in a cell $\boxplus$ of level i. We assign the *cell* and *level* of Φ to be $\boxplus$ and i, respectively. Recall that expansion $\Phi(\gamma)$ of an internal arc γ at $\boxplus$ is an expansion from a child cell of $\boxplus$ (given by its weight certificate) — $\Phi(\gamma)$ inherits cell and level assigned to that expansion. If γ is a bridge arc at a cell Δ, then we assign Δ and the level of Δ as the cell and level of $\Phi(\gamma)$, respectively. For a pathlet ϕ originating from Φ, we assign the cell and level of ϕ to be those of Φ. For a node u of $T(\Phi)$, we define the cell of u, denoted by $\boxplus(u)$, to be the cell assigned to the canonical pathlet ϕ_u associated with u, and $\mathrm{level}(u)$ to be the level of $\boxplus(u)$. Hence, if a non-root node u is an internal node (resp. leaf) then $\mathrm{level}(u) = \mathrm{level}(p(u)) - 1$ (resp. $\mathrm{level}(u) = \mathrm{level}(p(u))$), where $p(u)$ is the parent of u in $T(\Phi)$.

The *rank* of a canonical pathlet ϕ originating from Φ, denoted $\mathrm{rank}(\phi)$, is the smallest value i for which there is a node u in the MES tree $T(\Phi')$ of some level-i expansion Φ' such that $\phi = \phi_u$, i.e., $T(\Phi, \phi)$ is identical to the subtree of $T(\Phi')$ rooted at u. Informally, i is the first level where an MES tree "creates" ϕ.

Lemma 4.3. *Let ϕ_u be a canonical pathlet associated with a node u of an MES tree $T(\Phi)$. Then there is another MES Φ' with $\mathrm{level}(\Phi') \le \mathrm{level}(\Phi)$ and a node u' in $T(\Phi')$ such that $\phi_u = \phi_{u'}$ and u' is an exposed node of $T(\Phi')$.*

The above lemma helps to prove Lemma 4.4, which bounds the number of canonical pathlets.

Lemma 4.4. *(i) There are $O(ns^3h^2)$ canonical pathlets. (ii) For any cell $\boxplus \in \mathcal{C}$, there are $O(s^3h)$ canonical pathlets ϕ whose cell is $\boxplus$.*

Intersection tables. Define $\mathcal{P}(\Phi)$ as the set of canonical pathlets that originate from Φ. Let η_1, η_2 be two arcs of a pair of sibling cells. Recall that if η is a matching bridge arc then $\Phi(\eta)$ is the longest matching edge corresponding to η. We maintain an *intersection table* $\beta_{\eta_1,\eta_2} : \mathcal{P}(\Phi(\eta_1)) \times \mathcal{P}(\Phi(\eta_2)) \to \{0, 1\}$ such that $\beta_{\eta_1,\eta_2}(\phi_1, \phi_2) = 1$ if the two pathlets share an edge and 0 otherwise.

The intersection table is populated dynamically. In particular, we maintain the following invariant: before we compute MES for any cells at level i, we have already computed $\beta_{\eta_1,\eta_2}(\phi_1, \phi_2)$ for all pairs of pathlets of rank at most $i - 1$. Therefore, after we finish computing MES for cells at level i, we compute all not-yet-computed $\beta_{\eta_1,\eta_2}(\phi_1, \phi_2)$ where ϕ_1, ϕ_2 are canonical pathlets of rank at most i (i.e., one of ϕ_1, ϕ_2 has rank i), using Intersect procedure, which we present later in this section.

Since $|\mathcal{P}(\Phi(\eta))| = O(s^3h)$ and the total number of canonical pathlets is $O(ns^3h^2)$ by Lemma 4.4, the total size of intersection tables is $O(ns^6h^3)$. We thus obtain the following.

Lemma 4.5. *The total size of the data structure that maintains MES is $O(ns^6h^3)$, where h is the height of the hierarchy and s is the maximum number of clusters in a cell $\boxplus$.*

4.4 Pathlet Operations

We now describe how operations on pathlets and expansions (cf. Section 4.1) are implemented using MES and pathlet trees. Recall that $\boxplus$ is the cell at which we are constructing expansions.

Intersects. Given two pathlets ϕ_1, ϕ_2 originating from lower-level MES's Φ_1, Φ_2 (which are expansions at descendant cells of $\boxplus$) respectively, the Intersects procedure determines whether $\phi_1 \cap \phi_2 \neq \emptyset$. The input pathlets ϕ_1 and ϕ_2 are given as (the spines of) pathlet subtrees $T(\Phi_1, \phi_1)$ and $T(\Phi_2, \phi_2)$. For $i = 1, 2$, let $\mathsf{C}(\Phi_i, \phi_i)$ be the set of canonical nodes in of $T(\Phi_i, \phi_i)$. We describe the intersection-detection procedure for a pair $(u_1, u_2) \in \mathsf{C}(\Phi_1, \phi_1) \times \mathsf{C}(\Phi_2, \phi_2)$. Using this procedure for all such pairs, we determine whether $\phi_1 \cap \phi_2 \neq \emptyset$.

Fix a pair $u_1 \in \mathsf{C}(\Phi_1, \phi_1)$ and $u_2 \in \mathsf{C}(\Phi_2, \phi_2)$. We want to determine whether $\phi_{u_1} \cap \phi_{u_2} \neq \emptyset$. If $\boxplus(u_1) \cap \boxplus(u_2) = \emptyset$ then $\phi_{u_1} \cap \phi_{u_2} = \emptyset$, so we return no. Assume that $\boxplus(u_1) \cap \boxplus(u_2) \neq \emptyset$ and $\mathrm{level}(u_1) \geq \mathrm{level}(u_2)$. If $\mathrm{level}(u_1) = \mathrm{level}(u_2)$, then $\boxplus(u_1), \boxplus(u_2)$ are sibling cells. Let η_i be the arc in $G_{\boxplus(u_i)}$ corresponding to u_i. If η_1 or η_2 is a bridge arc then we return yes only if both of them are bridge arcs and $\Phi(\eta_1)$ and $\Phi(\eta_2)$ are the same matching edge, otherwise return no. On the other hand, if both η_1, η_2 are internal arcs then $\mathrm{level}(\eta_i), \mathrm{rank}(\phi_i) < \mathrm{level}(\boxplus)$, so $\beta_{\eta_1,\eta_2}(\phi_{u_1}, \phi_{u_2})$ has been computed and we return this value.

We now assume that $\mathrm{level}(u_1) > \mathrm{level}(u_2)$. If u_1 is a leaf, then it is impossible for the matching bridge edge ϕ_{u_1} to appear in ϕ_{u_2} (which is composed of lower-level edges) so we return no. If u_1 is an internal node then we recursively test u_2 against every children v ($\mathrm{level}(v) \geq \mathrm{level}(u_2)$) of u_1, to determine whether $\phi_v \cap \phi_{u_2} \neq \emptyset$. If any of them returns yes, we return yes. The following lemma bounds the running time using a packing argument.

Lemma 4.6. *Given a pair of canonical pathlets ϕ_1, ϕ_2 originating from Φ_1, Φ_2, Intersects takes $O(s^4h^2)$ time to determine whether $\phi_1 \cap \Phi_2 \neq \emptyset$.*

Summing this bound over all pairs in $\mathsf{C}(\Phi_1, \phi_1) \times \mathsf{C}(\Phi_2, \phi_2)$, we obtain the following:

Corollary 4.7. *Intersects procedure takes $O(s^6h^4)$ time.*

LastCommonEdge. Given two pathlets ϕ_1, ϕ_2 originating from lower-level MES's $\Phi_1 = \Phi(\eta_1), \Phi_2 = \Phi(\eta_2)$, respectively, returns the *last* edge of ϕ_1 that appears in ϕ_2. If such an edge exists, let e_1 denote this edge. The procedure first finds the edge $e_1 \in \phi_1$, then its copy in ϕ_2, and then computes the spines of e_1, of its copy in ϕ_2, and of their predecessors.

By invoking Intersects, we first determine whether $\phi_1 \cap \phi_2 \neq \emptyset$. If the answer is yes, we also find the last canonical node $u_1 \in \mathsf{C}(\Phi_1, \phi_1)$ such that $\phi_{u_1} \cap \phi_2 \neq \emptyset$, i.e., $e_1 \in \phi_{u_1}$. Let $U_2 \coloneqq \{u_2 \in \mathsf{C}(\Phi_2, \phi_2) \mid \phi_{u_1} \cap \phi_{u_2} \neq \emptyset\}$. If u_1 is a leaf node with matching edge e, then we set $e_1 \leftarrow e$. We assume that we have the spine to u_1 in Φ_1. If u_1 is not a leaf, then by invoking Intersects for all pairs $\mathrm{Ch}(u_1) \times U_2$ (where $\mathrm{Ch}(u_1)$ is the set of children of u_1 in $T(\Phi_1)$), we find the last child $v \in \mathrm{Ch}(u_1)$ such that ϕ_v intersects the canonical pathlet corresponding to a node in U_2. We now set $u_1 \leftarrow v$ and U_2 as above.

After having computed e_1, we compute its copy in ϕ_2 as follows. Let $u_2 \in U_2$ be the node such that $e_1 \in \phi_{u_2}$. Since ϕ_2 does not have any duplicate edges, u_2 is unique. By following a variant of Intersects procedure, we can traverse to the unique leaf of $T(\Phi_2, \phi_2)$ that contains a copy of e_1. After computing spines for e_1 and e_2, the predecessors e_3, e_4 can be computed by traversing backwards up the spines to find the first leaf left of e_1, e_2, respectively. If e_1 (resp. e_2) happen to be the first edge in ϕ_1 (resp. ϕ_2), then we return e_3 (resp. e_4) as *null* instead.

Since the maximum degree of a node in $T(\Phi_1)$ is s, the procedure calls Intersects procedure for $O(s^2h)$ pairs of canonical nodes at each level of the recursion. The depth of the recursion is h, so by Lemma 4.6, we obtain the following:

Lemma 4.8. *LastCommonEdge runs in time $O(s^6h^4)$.*

Median. Let ϕ be a pathlet originating from an MES Φ, represented as the spine of $T(\Phi, \phi)$. Using the length information stored at the nodes of $T(\Phi)$, the spine of the median edge of ϕ can be computed in $O(sh)$ time by traversing $T(\Phi, \phi)$ in a top-down manner.

Lemma 4.9. *Median runs in time $O(sh)$.*

AdjCost. Given the compact representation $\langle\Pi\rangle$ of a path/cycle Π in $\vec{G}_M$, whose MES is represented as a sequence $\phi_1 \circ \cdots \circ \phi_t$ of at most s pathlets, the procedure returns $\alpha_{\underline{\varepsilon}}(\Pi)$. Let b_i and a_i be the starting and ending tips of ϕ_i, respectively, and suppose ϕ_i originates from the MES Φ_i. Since the compact representation of ϕ_i consists of the spines of $T(\Phi_i, \phi_i)$, by accessing the nodes of $\mathsf{C}(\Phi_i, \phi_i)$, we can compute $\alpha_{\underline{\varepsilon}}(\phi_i)$ in $O(sh)$ time. Next we compute

$$\theta \coloneqq \alpha_{\underline{\varepsilon}}(\phi_1 \circ \cdots \circ \phi_t) = \sum_{i=1}^{t} \alpha_{\underline{\varepsilon}}(\phi_i) + \sum_{i=1}^{t-1} \alpha_{\underline{\varepsilon}}(a_i b_{i+1}).$$

If Π is a cycle then we set $\alpha_{\underline{\varepsilon}}(\Pi) \coloneqq \theta + \alpha_{\underline{\varepsilon}}(a_t b_1)$. If Π is a path with its beginning and ending tips being b_1 and a_t respectively, then we set $\alpha_{\underline{\varepsilon}}(\Pi) \coloneqq \theta$. Finally, if the first and last edges of Π are non-matching edges with its starting and ending tips being a_0 and b_{t+1}, respectively, then we set $\alpha_{\underline{\varepsilon}}(\Pi) \coloneqq \theta + \alpha_{\underline{\varepsilon}}(a_0 b_1) + \alpha_{\underline{\varepsilon}}(a_t b_{t+1})$.

Lemma 4.10. *Given the compact representation of a path or a cycle in $\vec{G}_M$ with its MES composed of at most s pathlets, AdjCost runs in time $O(s^2h)$.*

Report. Given the compact representation of a path/cycle $\langle\Pi\rangle$ of a path/cycle Π in $\vec{G}_M$, whose MES is represented as a sequence $\phi_1 \circ \cdots \circ \phi_t$ of at most s pathlets, the procedure returns the sequence of edges in Π. Let b_i, a_i, Φ_i be as above.

We first compute the sequence of edges in the alternating path $\phi_1 \circ \cdots \circ \phi_t$ with b_1 and a_t as its tips. If $t = 1$ and $T(\Phi_1, \phi_1)$ is a leaf, then the desired path is the matching (bridge) edge associated with the leaf. If $t = 1$ but ϕ_1 consists of more than one edge, then we recursively traverse the subtree rooted at each canonical node of $\mathsf{C}(\Phi_1, \phi_1)$, compute the alternating path represented by each canonical pathlet of ϕ_1, concatenate the results in sequence, and

add the non-matching edges between two canonical pathlets. For $t > 1$, we repeat this procedure for each ϕ_i, concatenate the paths for all ϕ_i's, and add the non-matching edges between the paths of two consecutive pathlets. Let Γ be the resulting sequence of edges.

If Π is an alternating path from b_1 to a_t, we simply return Γ. If Π is a path with its starting and ending tips being a_0 and b_{t+1}, we return $a_0 b_1 \circ \Gamma \circ a_t b_{t+1}$. Finally, if Π is a cycle then we return the cycle $\Gamma \circ b_t a_1$.

Lemma 4.11. *Given the compact representation of a path or a cycle in $\vec{G}_M$ with its MES composed of at most s pathlets, Report run in time $O((s^2+k)h)$ where k is the output size.*

Splice and Concatenate. Assuming that an edge e of a pathlet ϕ, originating from an MES Φ, is specified by the root-to-leaf path in $T(\Phi, \phi)$. The splice operation $e \blacktriangleright \phi$ can be performed in $O(sh)$ time by creating a pathlet subtree for the spliced pathlet from the compact representation of $T(\Phi, \phi)$.

Given pathlet subtrees $T(\Phi_1, \phi_1), \ldots, T(\Phi_t, \phi_t)$ of $\phi_1, \ldots, \phi_t$ in their compact forms, the MES tree $T(\Phi)$ for $\Phi = \phi_1 \circ \cdots \circ \phi_t$ can be constructed in $O(sh)$ time by first creating the root of $T(\Phi)$, then copying the spines of each ϕ_j, and attaching it as the subtree rooted at the j-th child of the root. We omit the straightforward details of the two procedures.

Updating intersection tables. After we have computed MES's at cells of level i, we update the intersection tables to add the entries for canonical pathlets of rank i. Let Φ be an MES in a cell $\boxplus$ at level i that was newly constructed. The canonical pathlets associated with the exposed nodes of $T(\Phi)$ are new canonical pathlets for which the intersection-table entries need to be computed. Let φ_1 be such a canonical pathlet originating from an MES Φ_{η_1}. Let $\Delta_1 \coloneqq \boxplus(\Phi_{\eta_1})$. Then for all matching/internal arcs η_2 in sibling cells Δ_2 of Δ_1 and for all canonical pathlets of $\Phi(\eta_2)$ of rank at most i (which we already have at our disposal), we compute $\beta_{\eta_1,\eta_2}(\varphi_1, \varphi_2)$ by calling Intersects (φ_1, φ_2). Since there are $O(1)$ sibling cells of Δ_1, each with $O(s^2)$ arcs, and there are $O(s^3 h)$ canonical pathlets originating from each arc, the total number of entries computed for φ_1 is $O(s^5 h)$. MES Φ has $O(sh)$ canonical pathlets, and $\boxplus$ has $O(s^2)$ MES's, so the total number entries computed for $\boxplus$ is $O(s^8 h^2)$. Using Corollary 4.7, we obtain the following:

Lemma 4.12. *The total time spent in updating the intersection tables because of the new canonical pathlets generated in the computation of expansions at a cell is $O(s^{14} h^6)$.*

Putting everything together, we obtain the following:

Lemma 4.13. *For any cell $\boxplus$ at level i, the total time spent in computing expansions, constructing MES trees, and updating intersection tables, assuming that all nodes at level greater than i have been been processed, is $(\varepsilon^{-1} \log n)^{O(d)}$.*

5 FINDPATH, AUGMENT, AND REPAIR PROCEDURES

We describe the operations FindPath, Augment, and Repair performed on the data structure built on each cell $\boxplus$. For an alternating path Π in $\vec{G}_M$, we call a cell $\boxplus \in \mathcal{C}$ *affected* by Π if $\boxplus$ contains a vertex of Π (that is, a point in $\mathbb{R}^d$). Let $\mathcal{C}_\Pi \subseteq \mathcal{C}$ denote the set of cells affected by Π.

FindPath(): It performs a DeleteMin operation on the priority queue $\mathcal{Q}$ storing the candidate subcell pairs *OptPairs* and retrieves a pair $(\square_{\boxplus}, \square'_{\boxplus})$, such that $\Phi \coloneqq \Phi_{\boxplus}(A_{\square_{\boxplus}}, B_{\square'_{\boxplus}})$ is an augmenting path. Recall that the data structure at $\boxplus$ stores a compact representation $\langle\Phi\rangle$ of Φ. The procedure calls Report$(\langle\Phi\rangle)$ and returns the resulting sequences of edges. The total time spent is $|\Phi| \cdot (\varepsilon^{-1} \log n)^{O(d)}$.

Augment(M, Π): It first sets $M \leftarrow M \oplus \Pi$. We store $\mathcal{C}_\Pi$, the set of cells affected by Π, in a priority queue Ξ with the level of the cell as its key. At each step, we retrieve a cell $\boxplus$ from Ξ of the lowest level and update the data structure by calling Repair$(\boxplus)$. If the call to Repair returns a reducing simple cycle Γ, we set $M \leftarrow M \oplus \Gamma$ and insert all cells in $\mathcal{C}_\Gamma$ into Ξ and continue. This process continues until Ξ becomes empty. The procedure then returns M.

Let $\Gamma_1, \ldots, \Gamma_t$ be the reducing cycles returned by Repair. As we see below, Repair takes $(\varepsilon^{-1} \log n)^{O(d)}$ amortized time. Then Augment takes time $(|\Pi| + \sum_i |\Gamma_i|) \cdot (\varepsilon^{-1} \log n)^{O(d)}$ time.

```
Augment(M, Π):
  Ξ ← ∅,   M ← M ⊕ Π
  for all ⊞ ∈ C_Π:  Insert(⊞, Ξ)
  while Ξ ≠ ∅:
      ⊞ ← DeleteMin(Ξ),   Γ ← Repair(⊞)
      if Γ ≠ null:
          M ← M ⊕ Γ
          for all ⊞' ∈ C_Γ:  Insert(⊞', Ξ)
  return M
```

Figure 3: Implementation of Augment using Repair.

Repair$(\boxplus)$: Repair is always called in a bottom-up manner, so we assume that all children of $\boxplus$ that were affected by the update in M are stable before the invocation of Repair at $\boxplus$. Repair$(\boxplus)$ reconstructs all the information stored at $\boxplus$, in the following steps:

(i) **(Updating clusters)** For each subcell $\square$ of $\boxplus$, we update the saturation status of $A_\square$ and $B_\square$. We maintain the free vertices of $A_\square$ and $B_\square$ in linked lists, and as they get matched we delete them.

(ii) **(Assigning arc weights)** Let $\square, \square'$ be subcells of $\boxplus$.
- If no child cell of $\boxplus$ contains both $\square$ and $\square'$, then we update the set of matching edges whose endpoints are in $\square, \square'$. We use Eq. (1) and (2) to compute the weights of arcs between the clusters of $\square$ and $\square'$.
- If some child cell contains both $\square$ and $\square'$, we use the recursive expression in Eq. (3) to compute the weights of arcs between the clusters of $\square, \square'$.

(iii) **(APSP computation)** We compute the minimum-weight compressed paths between every pair of nodes in $G_{\boxplus}$. If during this computation, we find a negative cycle C, (i.e. $w_{\boxplus}(C) < 0$), we compute (the compact representation of) a simple reducing subcycle Γ by calling SimpleReducingSubCycle(C). In this case, we abort the update of $\boxplus$ and return Report$(\langle\Gamma\rangle)$, which retrieves the sequence of edges in Γ.

(iv) **(Computing expansions)** For every pair of subcells $\square, \square'$, we compute $\Phi_{\boxplus}(A_\square, B_{\square'})$ as described above. The procedure may abort and return a simple reducing cycle Γ, in which case we abort the update of $\boxplus$ and return Report$(\langle\Gamma\rangle)$. If we succeed in computing expansions for all pairs, $\boxplus$ is stable.

(v) (**Augmenting paths**) For each pair of subcells $\square, \square'$ of $\boxplus$ such that both $A_\square$ and $B_{\square'}$ are unsaturated, let $\Pi_{\square,\square'} := \Phi_\boxplus(\square, \square')$; by construction, its tips are free points. We computing $\alpha_{\bar{\varepsilon}}(\Pi_{\square,\square'})$ by calling AdjCost($\langle \Pi_{\square,\square'} \rangle$). Otherwise, $\Pi_{\square,\square'}$ is undefined and $\alpha_{\bar{\varepsilon}}(\Pi_{\square,\square'}) = \infty$. We compute $(\square_\boxplus, \square'_\boxplus) := \arg\min_{\square,\square'} \alpha_{\bar{\varepsilon}}(\Pi_{\square,\square'})$ and insert $(\square_\boxplus, \square'_\boxplus)$ into *OptPairs*.

Omitting the details, we conclude:

Lemma 5.1. *FindPath takes $|\Pi| \cdot (\varepsilon^{-1} \log n)^{O(d)}$ time, where Π is the augmenting path returned by the procedure. Augment(Π, M) takes $(|\Pi| + \sum_i |\Gamma_i|) \cdot (\varepsilon^{-1} \log n)^{O(d)}$ time, where $\{\Gamma_i\}$ is the set of reducing cycles canceled by the procedure.*

The correctness of FindPath and Augment and the invariant CI follow from the following lemmas:

Lemma 5.2. *FindPath returns an augmenting path Π such that $\alpha_{\bar{\varepsilon}}(\Pi) \le \alpha^*_{\bar{\varepsilon},M}$.*

Proof: Let M be the current matching in the beginning of an iteration (when FindPath is called), and let Π^* be the augmenting path in $\vec{G}_M$ with the minimum $\bar{\varepsilon}$-adjusted cost, i.e., $\alpha_{\bar{\varepsilon}}(\Pi^*) = \alpha^*_{\bar{\varepsilon},M}$. By Lemma 3.3, there is a cell in $\mathcal{C}$ that contains Π^*. Let $\boxplus$ be the smallest cell in $\mathcal{C}$ that contains Π^* (if there is more than one, choose an arbitrary one). Let $\square$ (resp. $\square'$) be the subcell of $\boxplus$ that contains the starting (resp. ending) tip of Π^*. By Lifting Inequality (Lemma 3.4),

$$w_\boxplus(\pi_\boxplus(A_\square, B_{\square'})) \le \alpha_{\bar{\varepsilon}}(\Pi^*) = \alpha^*_{\bar{\varepsilon}}.$$

Since both $A_\square$ and $B_{\square'}$ are unsaturated, Repair procedure considers the expansion $\Phi_\boxplus(A_\square, B_{\square'})$ with free vertices being its tips as a candidate augmenting path. By Expansion Inequality (Lemma 3.1),

$$\alpha_{\bar{\varepsilon}}(\Phi_\boxplus(A_\square, B_{\square'})) \le w_\boxplus(\pi_\boxplus(A_\square, B_{\square'})). \tag{5}$$

Let $(\square_\boxplus, \square'_\boxplus)$ be the pair chosen by Repair$(\boxplus)$ for $\boxplus$ in Step (v):

$$\alpha_{\bar{\varepsilon}}(\Phi_\boxplus(A_{\square_\boxplus}, B_{\square'_\boxplus})) \le \alpha_{\bar{\varepsilon}}(\Phi_\boxplus(A_\square, B_{\square'})) \le \alpha^*_{\bar{\varepsilon}}.$$

Since $(\square_\boxplus, \square'_\boxplus) \in$ *OptPairs*, FindPath returns an augmenting path Π with $\alpha_{\bar{\varepsilon}}(\Pi) \le \alpha_{\bar{\varepsilon}}(\Phi_\boxplus(A_{\square_\boxplus}, B_{\square'_\boxplus})) \le \alpha^*_{\bar{\varepsilon}}$. □

Lemma 5.3. *Let M be any matching. If $\vec{G}_M$ contains an alternating cycle Γ with $\alpha_{\bar{\varepsilon}}(\Gamma) < 0$ then Augment returns a reducing cycle.*

Corollary 5.4. *Invariant CI holds at the start of each iteration.*

6 ANALYSIS OF THE ALGORITHM

In this section, we analyze the performance of the overall algorithm. We first analyze the cost of M_{alg}, the matching computed by the algorithm, and then analyze the running time of the algorithm.

Lemma 6.1. *The following properties hold throughout the algorithm: (i) The cost of any intermediate matching is at most $(1 + \frac{\varepsilon}{2}) \cdot \mathfrak{c}(M_{\text{opt}})$. (ii) In the beginning of each iteration when the invariant CI is satisfied, let M be the current matching and Π any augmenting path (with respect to M) such that $\alpha_{\bar{\varepsilon}}(\Pi) \le \alpha^*_{\bar{\varepsilon},M}$. Then,*

$$\bar{\mathfrak{c}}(\Pi) \le (1 + \frac{\varepsilon}{2}) \cdot \mathfrak{c}(M_{\text{opt}}) - \mathfrak{c}(M).$$

Proof: We prove the two statements of the lemma together by induction on the number of iterations. Initially $M = \varnothing$ and each edge of M_{opt} is an augmenting path, so both claims hold trivially. Suppose the claims hold for the first $i - 1$ iterations. Consider the i-th iteration, and let M be the current matching at the beginning of the i-th iteration. By induction hypothesis, $\mathfrak{c}(M) \le (1 + \frac{\varepsilon}{2}) \cdot \mathfrak{c}(M_{\text{opt}})$.

If $\alpha^*_{\bar{\varepsilon}} < 0$, then

$$\bar{\mathfrak{c}}(\Pi) \le \alpha_{\bar{\varepsilon}}(\Pi) \le \alpha^*_{\bar{\varepsilon}} < 0 \le (1 + \frac{\varepsilon}{2}) \cdot \mathfrak{c}(M_{\text{opt}}) - \mathfrak{c}(M).$$

Since FindPath returns an augmenting path P with $\bar{\mathfrak{c}}(P) \le \alpha_{\bar{\varepsilon}}(P) \le \alpha^*_{\bar{\varepsilon}} < 0$,

$$\mathfrak{c}(M \oplus P) = \mathfrak{c}(M) + \bar{\mathfrak{c}}(P) \le \mathfrak{c}(M) \le (1 + \frac{\varepsilon}{2}) \cdot \mathfrak{c}(M_{\text{opt}}). \tag{6}$$

Next, consider the case when $\alpha^* \ge 0$. The symmetric difference $M \oplus M_{\text{opt}}$ consists of a nonempty set $\mathcal{P}$ of pairwise-disjoint augmenting paths and a (possibly empty) set $\mathcal{N}$ of alternating cycles. Every augmenting path $\Pi' \in \mathcal{P}$ has $\alpha_{\bar{\varepsilon}}(\Pi') \ge \alpha^*_{\bar{\varepsilon}} \ge 0$ by definition of $\alpha^*_{\bar{\varepsilon}}$, and every cycle $\Gamma \in \mathcal{N}$ has $\alpha_{\bar{\varepsilon}}(\Gamma) \ge 0$ by the cycle invariant CI. Let Π be an augmenting path with $\alpha_{\bar{\varepsilon}}(\Pi) \le \alpha^*_{\bar{\varepsilon}}$. Then

$$\begin{aligned}
\bar{\mathfrak{c}}(\Pi) &\le \alpha^*_{\bar{\varepsilon}} \le \sum_{\Pi' \in \mathcal{P}} \alpha_{\bar{\varepsilon}}(\Pi') + \sum_{\Gamma \in \mathcal{N}} \alpha_{\bar{\varepsilon}}(\Gamma) \\
&= \sum_{\Pi' \in \mathcal{P}} \left(\bar{\mathfrak{c}}(\Pi') + c_0 \bar{\varepsilon} \cdot \|\Pi'\|\right) + \sum_{\Gamma \in \mathcal{N}} \left(\bar{\mathfrak{c}}(\Gamma) + c_0 \bar{\varepsilon} \cdot \|\Gamma\|\right) \\
&\le \mathfrak{c}(M_{\text{opt}}) - \mathfrak{c}(M) + c_0 \bar{\varepsilon} \cdot \left(\mathfrak{c}(M_{\text{opt}}) + \mathfrak{c}(M)\right) \\
&= (1 + c_0 \bar{\varepsilon}) \cdot \mathfrak{c}(M_{\text{opt}}) + c_0 \bar{\varepsilon} \cdot \mathfrak{c}(M) - \mathfrak{c}(M) \\
&\le \left(1 + c_0 \bar{\varepsilon} \left(2 + \frac{\varepsilon}{2}\right)\right) \cdot \mathfrak{c}(M_{\text{opt}}) - \mathfrak{c}(M) \text{ (induction hypothesis)} \\
&\le \left(1 + \frac{\varepsilon}{8} \left(2 + \frac{\varepsilon}{2}\right)\right) \cdot \mathfrak{c}(M_{\text{opt}}) - \mathfrak{c}(M) \ \left(\bar{\varepsilon} = \frac{\varepsilon}{c_1} \text{ and } c_1 \ge 8c_0\right) \\
&\le \left(1 + \frac{\varepsilon}{2}\right) \cdot \mathfrak{c}(M_{\text{opt}}) - \mathfrak{c}(M). \quad (0 \le \varepsilon \le 1)
\end{aligned}$$

FindPath returns an augmenting path P with $\alpha_{\bar{\varepsilon}}(P) \le \alpha^*_{\bar{\varepsilon}}$. Thus

$$\begin{aligned}
\mathfrak{c}(M \oplus P) = \mathfrak{c}(M) + \bar{\mathfrak{c}}(P) &\le \mathfrak{c}(M) + \left(1 + \frac{\varepsilon}{2}\right) \cdot \mathfrak{c}(M_{\text{opt}}) - \mathfrak{c}(M) \\
&= \left(1 + \frac{\varepsilon}{2}\right) \cdot \mathfrak{c}(M_{\text{opt}}).
\end{aligned} \tag{7}$$

After augmenting M with P, the algorithm may cancel a sequence of reducing cycles. Since each of these cycles has negative net cost, the cycle cancellations only reduces the cost of the matching. We conclude that the cost of an intermediate matching remains at most $(1 + \frac{\varepsilon}{2}) \cdot \mathfrak{c}(M_{\text{opt}})$ during the i-th iteration. □

Lemma 6.2. *The following two statements hold:*

(i) Let M be an intermediate matching and Π an augmenting path in $\vec{G}_M$ with $\alpha_{\bar{\varepsilon}}(\Pi) \le \alpha^*_{\bar{\varepsilon}}$. Then, $\|\Pi\| \le \frac{27\sqrt{d}n}{\varepsilon}$.

(ii) Let M be an intermediate matching and Γ a reducing cycle in $\vec{G}_M$. Then, $\|\Gamma\| \le \frac{27\sqrt{d}n}{\varepsilon}$.

We now analyze the running time of the algorithm. Denote the sequence of augmenting paths computed by the algorithm as $\Pi_1, \ldots, \Pi_n$, let $\mathcal{N}_i$ be the set of alternating cycles that were canceled after augmenting by Π_i during a call to the Augment procedure, and let M_i be the matching returned by the Augment(Π_i, M_{i-1}). Then by Lemma 5.1, the total time spent by the algorithm is $(\sum_i |\Pi_i| + \sum_{\Gamma \in \mathcal{N}} |\Gamma|) \cdot (\varepsilon^{-1} \log n)^{O(d)}$, where $\mathcal{N} := \bigcup_i \mathcal{N}_i$. Since the cost of each edge in G is at least 1 by (P1), $|\Pi_i| \le \|\Pi_i\|$ and $|\Gamma| \le \|\Gamma\|$, so we will bound their Euclidean lengths instead.

We use the shorthand $\alpha_{\theta,i}$ to denote α_{θ,M_i}—the $\bar{\varepsilon}$-adjusted cost after the i-th iteration where M_i is a partial matching with i edges—and set $\alpha_i^* := \alpha^*_{\theta,M_i}$. We begin by bounding α_i^*.

Lemma 6.3. $\alpha_i^* \le O\left(\frac{n}{\varepsilon(n-i)}\right)$.

Proof: $M_{\text{opt}} \oplus M_i$ consists of a set of pairwise-disjoint alternating cycles and $n-i$ augmenting paths $P_1, \dots, P_{n-i}$. Since $\alpha^*_{\bar{\varepsilon},i}$ is the minimum $\bar{\varepsilon}$-adjusted cost of an augmenting path in $\vec{G}_{M_i}$, we have

$$\begin{aligned}(n-i)\cdot\alpha_i^* &\le \sum_{j=1}^{n-i}\alpha_{\bar{\varepsilon},i}(P_j) \le \sum_{j=1}^{n-i}\left(\bar{\mathfrak{c}}_{M_i}(P_j) + c_0\bar{\varepsilon}\cdot\|P_j\|\right)\\ &\le \sum_{j=1}^{n-i}\left(\mathfrak{c}(P_j\cap M_{\text{opt}}) - \mathfrak{c}(P_j\cap M_i)\right)\\ &\quad + \frac{c_0\varepsilon}{c_1}\sum_{j=1}^{n-i}\left(\mathfrak{c}(P_j\cap M_{\text{opt}}) + \mathfrak{c}(P_j\cap M_i)\right)\\ &\le \mathfrak{c}(M_{\text{opt}}) + \frac{\varepsilon}{8}\left(\mathfrak{c}(M_{\text{opt}}) + \mathfrak{c}(M_i)\right) = O(n/\varepsilon),\end{aligned}$$

where the last inequality follows from property (P3) of the input and combining Lemma 6.1(i) with (P3) and $\varepsilon < 1$. □

Corollary 6.4. $\sum_{i=0}^{n-1}\alpha_i^* = O(\varepsilon^{-1}n\log n)$.

Lemma 6.5. $\sum_i\|\Pi_i\| + \sum_{\Gamma\in\mathcal{N}}\|\Gamma\| = O(\varepsilon^{-2}n\log^2 n)$.

Proof: Recall that Find-Path guarantees $\alpha_{\underline{\varepsilon},i}(\Pi_{i+1}) \le \alpha_i^*$. Therefore, by Lemma 6.3,

$$\sum_{i=1}^{n}\alpha_{\underline{\varepsilon},i-1}(\Pi_i) = O(\varepsilon^{-1}n\log n).$$

Since $\alpha_{\underline{\varepsilon}}(\Gamma) < 0$ for all cycles $\Gamma\in\mathcal{N}$, we have

$$\sum_{i=1}^{n}\alpha_{\underline{\varepsilon},i-1}(\Pi_i) + \sum_{\Gamma\in\mathcal{N}}\alpha_{\underline{\varepsilon}}(\Gamma) = O(\varepsilon^{-1}n\log n).$$

On the other hand, by definition of the adjusted cost,

$$\begin{aligned}\sum_{i=1}^{n}\alpha_{\underline{\varepsilon},i-1}(\Pi_i) + \sum_{\Gamma\in\mathcal{N}}\alpha_{\underline{\varepsilon}}(\Gamma) &= \sum_{i=1}^{n}\bar{\mathfrak{c}}(\Pi_i) + \sum_{\Gamma\in\mathcal{N}}\bar{\mathfrak{c}}(\Gamma)\\ &\quad + c_0\underline{\varepsilon}\left(\sum_{i=1}^{n}\|\Pi_i\| + \sum_{\Gamma\in\mathcal{N}}\|\Gamma\|\right).\end{aligned}\tag{8}$$

Observe that the augmentation of all paths and cycles results in the final matching M_{alg}. Therefore, the net-cost terms in (8) sum to $\mathfrak{c}(M_{\text{alg}}) \ge 0$, and we obtain

$$\begin{aligned}\sum_{i=1}^{n}\|\Pi_i\| + \sum_{\Gamma\in\mathcal{N}}\|\Gamma\| &= \frac{1}{c_0\underline{\varepsilon}}\left(O\left(\frac{n\log n}{\varepsilon}\right) - \mathfrak{c}(M_{\text{alg}})\right)\\ &= \frac{c_1c_2}{c_0}\cdot\frac{\log n}{\varepsilon}\cdot O\left(\frac{n\log n}{\varepsilon}\right) = O\left(\frac{n\log^2 n}{\varepsilon^2}\right).\end{aligned}$$

The second equality is obtained by substituting the value of $\underline{\varepsilon}$ and using $\mathfrak{c}(M_{\text{alg}}) > 0$. □

Using (P1), (P2), and Lemmas 5.1 and 6.5, we obtain

Lemma 6.6. *The overall algorithm runs in* $n\cdot(\varepsilon^{-1}\log n)^{O(d)}$ *time.*

Lemmas 6.1 and 6.6 together prove Theorem 1.1.

7 PROOF OF EXPANSION AND LIFTING INEQUALITIES

Finally, we prove the expansion and lifting inequalities.

7.1 Proof of Expansion Inequality

Lemma 3.1. *Let* $\boxplus$ *be a stable cell, and let* $(X,Y)\in E_{\boxplus}$. *Then*

$$\alpha_{\underline{\varepsilon}}(\Phi_{\boxplus}(X,Y)) \le w_{\boxplus}(\pi_{\boxplus}(X,Y)).$$

Proof: We write $\pi_{\boxplus}(X,Y)$ as its arc sequence $\langle\eta_1\circ\eta_2\circ\cdots\circ\eta_k\rangle$. Let $\Pi := \Phi(\eta_1)\circ\Phi(\eta_2)\circ\cdots\circ\Phi(\eta_k)$. Recall that if η_i is an internal arc of the form $(A_{\square}, B_{\square'})$ and $i > 1$ (resp. $i < k$), then the starting (resp. ending) tip of $\Phi(\eta_i)$ is changed to the ending (resp. starting) tip of $\Phi(\eta_{i-1})$ (resp. $\Phi(\eta_{i+1})$) during the concatenation.

Π is a possibly self-intersecting path in $\vec{G}_M$, and $\Phi_{\boxplus}(X,Y)$ is formed by simplifying Π (cycle removal) as described in Section 4. Since $\boxplus$ is stable, none of these cycles were reducing. Therefore $\alpha_{\underline{\varepsilon}}(\Phi_{\boxplus}(X,Y)) \le \alpha_{\underline{\varepsilon}}(\Pi)$. We complete the proof by showing $\alpha_{\underline{\varepsilon}}(\Pi) \le w_{\boxplus}(\pi_{\boxplus}(X,Y))$.

Recall that $\delta := c_4\underline{\varepsilon}$. We choose $c_4 \ge 2c_0\sqrt{d}$. For each arc $\eta_j := (X_j, X_{j+1})$ in $\pi_{\boxplus}(X,Y)$, let $\square_j$ be the subcell containing X_j, and cntr_j the center of $\square_j$. Recall that $\text{diam}(\square_j) = \text{diam}(\square_{j+1}) \le \frac{\delta}{4}\ell_i$. We obtain a bound for each η_j:

Case 1: η_j is a bridge arc. First suppose that η_j is a non-matching arc, i.e., $X_j\subseteq A_{\square_j}, X_{j+1}\subseteq B_{\square_{j+1}}$ and $\Phi(\eta_j) = (a_j, b_{j+1})$ for some pair in $A_{\square_j}\times B_{\square_{j+1}}$ (even after modifying the tips).

$$\begin{aligned}\alpha_{\underline{\varepsilon}}(a_j,b_{j+1}) &= \|a_j - b_{j+1}\| + c_0\underline{\varepsilon}\cdot\|a_j - b_{j+1}\|\\ &\le \|\text{cntr}_j - \text{cntr}_{j+1}\| + 2\cdot\tfrac{\delta}{4}\ell_i + c_0\underline{\varepsilon}\sqrt{d}\ell_i\\ &\le \|\text{cntr}_j - \text{cntr}_{j+1}\| + \delta\ell_i = w_{\boxplus}(X_j,X_{j+1}).\end{aligned}$$

Next, if η_j is a matching arc then $\Phi(\eta_j) = (b_j, a_{j+1}) \in M\cap B_{\square_j}\times A_{\square_{j+1}}$. We repeat a similar analysis on this matching edge:

$$\begin{aligned}\alpha_{\underline{\varepsilon}}(\Phi(\eta_j)) &= -\|b_j - a_{j+1}\| + c_0\underline{\varepsilon}\cdot\|b_j - a_{j+1}\|\\ &\le -\|\text{cntr}_j - \text{cntr}_{j+1}\| + \delta\ell_i = w_{\boxplus}(X_j,X_{j+1}).\end{aligned}$$

Case 2: $\eta_j = (X_j, X_{j+1})$ is an internal arc. Let $(\boxplus',\Delta,\Delta')$ be the weight certificate of η_j, and let $X_j'\subset\Delta$ (resp. $X_{j+1}'\subset\Delta'$) be the child cluster of X_j (resp. X_{j+1}) in $\boxplus'$. Then $\Phi(\eta_j)$ is the expansion $\Phi_{\boxplus'}(X_j', X_{j+1}')$ with its starting and ending tips possibly being modified to another point in $\square_j$ and $\square_{j+1}$, respectively. Changing the tips increases the length of each of first and last edges in $\Phi(\eta_j)$ by at most $\text{diam}(\square_j) = \text{diam}(\square_{j+1}) \le \frac{\delta}{4}\ell_i$ each, thus

$$\alpha_{\underline{\varepsilon}}(\Phi(\eta_j)) \le \alpha_{\underline{\varepsilon}}(\Phi_{\boxplus'}(X_j', X_{j+1}')) + \frac{\delta}{2}\ell_i \le w_{\boxplus}(X_j,X_{j+1}).$$

The last inequality follows because $(\boxplus',\Delta,\Delta')$ is the weight certificate of η_j.

Summing over the expansions of all arcs in Π, we obtain $\alpha_{\underline{\varepsilon}}(\Pi) \le w_{\boxplus}(\pi_{\boxplus}(B_{\square}, A_{\square'}))$. □

7.2 Proof of Lifting Inequality

We first state two lemmas needed to prove the lifting inequality.

Lemma 7.1. *Let (p, q) be a pair of points in $A \cup B \subset \mathbb{R}^d$. Let i be the smallest value for which a cell $\square \in \mathcal{C}_i$ contains both p and q. Then*

(i) $\ell_{i-2} \le \|p-q\|_\infty \le \ell_i$ and
(ii) $\ell_{i-2} \le \|p-q\|_2 \le \sqrt{d}\ell_i$.

Lemma 7.2. *Let $\square$ be a level-i cell and $(X, Y) \in E_\square$ be a pair of clusters. Let $\square'$ be a level-i' ancestor cell of $\square$ for $i' > i$, and X' (resp. Y') the ancestor cluster of X (resp. Y) in $\square'$. Then,*

$$w_{\square'}(\pi_{\square'}(X', Y')) \le w_\square(\pi_\square(X, Y)) + 2\delta\ell_{i'}.$$

We are now ready to prove the lifting inequality.

Lemma 3.4. *Let Π be an alternating path in $\vec{G}_M$ from a point p to a point q (possibly $p = q$ in the case Π is a cycle) that is completely contained in a cell of level at most h. Let $\square$ be a cell at the smallest level that contains Π, and let X (resp. Y) be the cluster in $V_\square$ containing p (resp. q). Then either $\square$ is not stable or $w_\square(\pi_\square(X, Y)) \le \alpha_{\bar\varepsilon}(\Pi)$.*

Proof: If $\square$ is not stable then the lemma follows trivially, so assume that $\square$ is stable. Then all descendants of $\square$ are also stable, and arc weights and expansions are well-defined at all of them. We construct a compressed path P in $G_\square$ where

$$w_\square(P) \le \bar\phi(\Pi) + c_5 i\delta\|\Pi\| \tag{9}$$

for some constant $c_5 \ge 64$. If $\square$ is at level i, then by assumption $i \le h \le c_3 \log n$. Since $\delta = c_4\underline{\varepsilon}$ and $\underline{\varepsilon} = \frac{\bar\varepsilon}{c_2 \log n}$, we obtain

$$\begin{aligned} w_\square(P) &\le \bar\phi(\Pi) + c_5(c_3 \log n)c_4\left(\frac{\bar\varepsilon}{c_2 \log n}\right) \cdot \|\Pi\| \\ &\le \bar\phi(\Pi) + c_0\bar\varepsilon \cdot \|\Pi\| = \alpha_{\bar\varepsilon}(\Pi), \end{aligned} \tag{10}$$

provided that $c_0 \ge c_3c_4c_5/c_2$.

We prove (9) by induction over hierarchy levels. Let $\square$ be the smallest cell containing Π and let i be the level of $\square$. If $i = 0$ then Π is an empty path, as each cell at level 0 contains no edge of $\vec{G}_M$, so the lemma holds trivially. Assume that the lemma holds for all levels less than i.

Let $\Pi = \langle p_1 = p, p_2, \ldots, p_t = q\rangle$. We first construct a sequence P of nodes in $V_\square$ by traversing Π and using a greedy approach, and then refine it to ensure that the resulting sequence is a path in $G_\square$. For a point p_k, let $X(p_k)$ be the cluster in $V_\square$ containing p_k. To start, set $P := \langle X_1\rangle = \langle X(p_1)\rangle$. Suppose we have traversed a prefix of Π, constructed a prefix of $P = \langle X_1, X_2, \ldots, X_j\rangle$, and we are currently at point p_k where $X(p_k) = X_j$. Let p^* be the furthest point of $\langle p_k, p_{k+1}, \ldots, p_t\rangle$ such that the subpath $\langle p_k, \ldots, p^*\rangle \subseteq \Pi$ lies in a single child cell of $\square$. There are a few special cases where p^* may not be well-defined above.

- If $p^* = p_k$ (the longest single-child prefix is one point), then (p_k, p_{k+1}) corresponds to a bridge arc in $G_\square$, and we set $p^* \leftarrow p_{k+1}$.
- If the longest single-child prefix starting from p_k is at least two points, but one child cell contains the entire suffix of Π. We set $p^* \leftarrow q$.

After setting p^*, we choose $X_{k+1} := X(p^*)$. If p^* is the last point of Π, we stop. Otherwise, we continue.

We note that P is not necessarily a path in $G_\square$ because it might contain a consecutive pair X_j, X_{j+1} where X_j and X_{j+1} are both A-clusters or both B-clusters (hence, (X_j, X_{j+1}) is not an arc in the bipartite graph $G_\square$). We fix these pairs by inserting an extra cluster in between each such occurrence. For each X_j, let $u_j = k$ be the index for which $X(p_k) = j$. Let $\Pi_j = \langle p_{u_j}, \ldots, p_{u_{j+1}}\rangle \subseteq \Pi$. Assume both X_j and X_{j+1} are A-clusters (the other case can be handled in an analogous way), then $p_{u_j}, p_{u_{j+1}} \in A$ and $\Pi_j \cap B$ is nonempty. Choose any point $y \in \Pi_j \cap B$, set $Y_j = X(y)$, and replace the substring X_j, X_{j+1} in P with X_j, Y_j, X_{j+1}. Both (X_j, Y_j) and (Y_j, X_{j+1}) are internal arcs in $G_\square$, as a single child of $\square$ contains all of Π_j, and the sequence alternates between A- and B-clusters.

Let $P := \langle X_1 = X, X_2, \ldots, X_{s-1}, X_s = Y\rangle$ be the resulting compressed path from $X = X(p)$ to $Y = X(q)$ in $G_\square$. The above construction guarantees that for any $j \le s-3$, no contiguous quadruple $X_j, X_{j+1}, X_{j+2}, X_{j+3}$ lie in a single child cell of $\square$. For $1 \le j \le s$, let $\square_j$ be the subcell of $\square$ containing X_j, and let cntr_j be the center of $\square_j$. We now bound $w_\square(P)$ by relating $w_\square(X_j, X_{j+1})$ to the cost of Π_j. There are two main cases:

*Case 1: (X_j, X_{j+1}) is a **bridge arc**.* Since $\mathrm{diam}(\square_j) = \mathrm{diam}(\square_{j+1}) \le \frac{\delta\ell_i}{4}$, we have

$$\|p_{u_j} - p_{u_{j+1}}\| - \frac{\delta\ell_i}{2} \le \|\mathrm{cntr}_j - \mathrm{cntr}_{j+1}\| \le \|p_{u_j} - p_{u_{j+1}}\| + \frac{\delta\ell_i}{2}.$$

If (X_j, X_{j+1}) is a non-matching bridge arc, then

$$\begin{aligned} w_\square(X_j, X_{j+1}) &= \|\mathrm{cntr}_j - \mathrm{cntr}_{j+1}\| + \delta\ell_i \\ &\le \|p_{u_j} - p_{u_{j+1}}\| + \frac{\delta\ell_i}{2} + \delta\ell_i = \bar\phi(\Pi_j) + \frac{3}{2}\delta\ell_i. \end{aligned} \tag{11}$$

If (X_j, X_{j+1}) is a matching bridge arc, following a similar argument, we obtain

$$\begin{aligned} w_\square(X_j, X_{j+1}) &= -\|\mathrm{cntr}_j - \mathrm{cntr}_{j+1}\| + \delta\ell_i \\ &\le -\|p_{u_j} - p_{u_{j+1}}\| + \frac{3}{2}\delta\ell_i = \bar\phi(\Pi_j) + \frac{3}{2}\delta\ell_i. \end{aligned} \tag{12}$$

*Case 2: (X_j, X_{j+1}) is an **internal arc**.* (X_j, X_{j+1}) is a pair of clusters contained in a single child cell $\square'$ of $\square$. Let X'_j (resp. X'_{j+1}) be the child cluster of X_j (resp. X_{j+1}) in $\square'$ that contains p_{u_j} (resp. $p_{u_{j+1}}$). By the definition of internal arc weights, we have

$$\begin{aligned} w_\square(X_j, X_{j+1}) &\le \alpha_{\underline{\varepsilon}}(\Phi_{\square'}(X'_j, X'_{j+1})) + \delta\ell_i \\ &\le w_{\square'}(\pi_{\square'}(X'_j, X'_{j+1})) + \delta\ell_i, \end{aligned} \tag{13}$$

where the last inequality follows from Lemma 3.1 and the fact that $\square'$ is stable.

Let $\hat\square$ be the smallest descendant of $\square'$ that contains Π_j, and let $\hat\imath \le i-1$ be the level of $\hat\square$ (note that $\hat\square$ may be $\square'$ itself). Let $\hat X_j$ (resp. $\hat X_{j+1}$) be the descendant cluster of X_j (resp. X_{j+1}) in $\hat\square$ containing p_{u_j} (resp. $p_{u_{j+1}}$). Since $\pi_{\hat\square}(\hat X_j, \hat X_{j+1})$ is the minimum-weight path from $\hat X_j$ to $\hat X_{j+1}$ in $G_{\hat\square}$, by the induction hypothesis,

$$w_{\hat\square}(\pi_{\hat\square}(\hat X_j, \hat X_{j+1})) \le \bar\phi(\Pi_j) + c_5\hat\imath\delta\|\Pi_j\|.$$

Using Lemma 7.2, we obtain

$$\begin{aligned} w_{\square'}(\pi_{\square'}(X'_j, X'_{j+1})) &\le w_{\hat\square}(\pi_{\hat\square}(\hat X_j, \hat X_{j+1})) + 2\delta\ell_{i-1} \\ &\le \bar\phi(\Pi_j) + c_5\hat\imath\delta\|\Pi_j\| + \delta\ell_i \\ &\le \bar\phi(\Pi_j) + c_5(i-1)\delta\|\Pi_j\| + \delta\ell_i. \end{aligned} \tag{14}$$

Substituting (14) into (13) we obtain

$$w_\square(X_j, X_{j+1}) \le \bar\phi(\Pi_j) + c_5(i-1)\delta\|\Pi_j\| + 2\delta\ell_i. \tag{15}$$

Combining (11) and (15), the terms from (15) dominate, so

$$\begin{aligned} w_{\boxplus}(P) &= \sum_{j=1}^{s-1} w_{\boxplus}(X_j, X_{j+1}) \\ &\le \sum_{j=1}^{s-1} \left(\bar{\ell}(\Pi_j) + c_5(i-1)\delta\|\Pi_j\| + 2\delta\ell_i\right) \\ &= \bar{\ell}(\Pi) + c_5(i-1)\delta\|\Pi\| + 2(s-1)\delta\ell_i \end{aligned} \tag{16}$$

Recall that no contiguous quadruple $X_j, X_{j+1}, X_{j+2}, X_{j+3}$ (alternatively, three consecutive arcs of P) lie in a single child cell, for all $j \le s-3$. Then, by Lemma 7.1, $\|\Pi_j \circ \Pi_{j+1} \circ \Pi_{j+2}\| \ge \ell_{i-1}/4 = \ell_i/8$. Summing,

$$\sum_{j=1}^{s-3} \|\Pi_j \circ \Pi_{j+1} \circ \Pi_{j+2}\| \ge (s-3)\,\frac{\ell_i}{8}.$$

In this sum, each subpath Π_j is being counted at most three times. Hence, the left-hand quantity is at most $3\|\Pi\|$, therefore

$$(s-3)\ell_i \le 24\|\Pi\|. \tag{17}$$

Since $\boxplus$ is the smallest cell to contain Π, by Lemma 7.1, $\|\Pi\| \ge \ell_i/4$. Plugging (17) into (16),

$$\begin{aligned} w_{\boxplus}(P) &\le \bar{\ell}(\Pi) + c_5(i-1)\delta\|\Pi\| + 2(s-1)\delta\ell_i \\ &\le \bar{\ell}(\Pi) + c_5(i-1)\delta\|\Pi\| + 48\delta\|\Pi\| + 4\delta\ell_i \\ &\le \bar{\ell}(\Pi) + c_5(i-1)\delta\|\Pi\| + 64\delta\|\Pi\| \\ &\le \bar{\ell}(\Pi) + c_5 i\cdot\delta\|\Pi\|, \end{aligned}$$

provided that $c_5 \ge 64$. This completes the proof of the lemma. □

8 CONCLUDING REMARKS

In this paper, we presented the first near-linear deterministic ε-approximation algorithm for computing EMWM in any fixed dimension. We conclude by mentioning a few open problems: Can our approach be extended to attain a near-linear-time deterministic algorithm for computing an ε-optimal transport, or for computing an RMS matching? More broadly, can a cover-tree based approach lead to near-linear-time deterministic approximation algorithms for other network-design problems such as Euclidean TSP?

Acknowledgments. Work on this paper is supported by NSF under grants CCF-1909171, IIS-18-14493, and CCF-20-07556.

REFERENCES

[1] P. K. Agarwal, H.-C. Chang, and A. Xiao. Efficient algorithms for geometric partial matching. In *Proc. 35th Intl. Sympos. Comput. Geom.*, pages 6:1–6:14, 2019.
[2] P. K. Agarwal, A. Efrat, and M. Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM J. Comput.*, 29(3):42, 2000.
[3] P. K. Agarwal, K. Fox, D. Panigrahi, K. R. Varadarajan, and A. Xiao. Faster algorithms for the geometric transportation problem. In *Proc. 33rd Intl. Sympos. Comput. Geom.*, pages 7:1–7:16, 2017.
[4] P. K. Agarwal and R. Sharathkumar. Approximation algorithms for bipartite matching with metric and geometric costs. In *Proc. 46th Annu. ACM Sympos. Theory Comput.*, pages 555–564, 2014.
[5] P. K. Agarwal and K. Varadarajan. A near-linear constant-factor approximation for Euclidean bipartite matching? In *Proc. 20th Annu. Sympos. Comput. Geom.*, pages 247–252, 2004.
[6] A. Andoni, A. Nikolov, K. Onak, and G. Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Proc. 46th Annu. ACM Sympos. Theory Comput.*, pages 574–583, 2014.
[7] Y. Bartal, N. Fandina, and O. Neiman. Covering metric spaces by few trees. In *Proc. 46th Intl. Colloq. Auto., Lang., and Program.*, pages 20:1–20:16, 2019.
[8] Y. Bartal, N. Linial, M. Mendel, and A. Naor. On metric Ramsey-type phenomena. *Annals of Mathematics*, 162:643–709, 2005.
[9] G. Beugnot, A. Genevay, K. Greenewald, and J. Solomon. Improving approximate optimal transport distances using quantization. *arXiv:2102.12731*, 2021.
[10] J. v. d. Brand, Y. T. Lee, D. Nanongkai, R. Peng, T. Saranurak, A. Sidford, Z. Song, and D. Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In *Proc. 61st IEEE Annu. Sympos. Found. Comp. Sci.*, pages 919–930, 2020.
[11] T. M. Chan, S. Har-Peled, and M. Jones. On locality-sensitive orderings and their applications. In *Proc. 10th Innov. Theor. Comp. Sci. Conf.*, pages 21:1–21:17, 2019.
[12] M. Charikar. Similarity estimation techniques from rounding algorithms. In *Proc. 34th Annu. ACM Sympos. Theory Comput.*, pages 380–388, 2002.
[13] K. Fox and J. Lu. A near-linear time approximation scheme for geometric transportation with arbitrary supplies and spread. In *Proc. 36th Intl. Sympos. Comput. Geom.*, pages 45:1–45:18, 2020.
[14] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.
[15] H. N. Gabow. The weighted matching approach to maximum cardinality matching. *arXiv:1703.03998 [cs]*, Mar. 2017.
[16] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18:1013–1036, 1989.
[17] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for general graph-matching problems. *J. ACM*, 38(4):815–853, 1991.
[18] A. Gupta, A. Kumar, and R. Rastogi. Traveling with a Pez dispenser (or, routing issues in MPLS). *SIAM J. Comput.*, 34(2):453–474, 2005.
[19] J. Hopcroft and R. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.
[20] P. Indyk. A near linear time constant factor approximation for Euclidean bichromatic matching (cost). In *Proc. 18th Annu. ACM-SIAM Sympos. Discrete Algo.*, pages 29–32, 2007.
[21] H. Kaplan, W. Mulzer, L. Roditty, P. Seiferth, and M. Sharir. Dynamic planar voronoi diagrams for general distance functions and their algorithmic applications. *Discret. Comput. Geom.*, 64(3):838–904, 2020.
[22] A. B. Khesin, A. Nikolov, and D. Paramonov. Preconditioning for the geometric transportation problem. In *Proc. 35th Intl.. Sympos. Comput. Geom.*, pages 15:1–15:14, 2019.
[23] N. Lahn and S. Raghvendra. An $\tilde{O}(n^{5/4})$ time ε-approximation algorithm for rms matching in a plane. In *Proc. 31st Annu. ACM-SIAM Sympos. Discrete Algo.*, pages 869–888, 2021.
[24] H. Le and S. Solomon. Truly optimal Euclidean spanners. In *Proc. 60th IEEE Annu. Sympos. Found. Comp. Sci.*, pages 1078–1100, 2019.
[25] H. Liu, X. Gu, and D. Samaras. A two-step computation of the exact GAN Wasserstein distance. In *Proc. 35th Intl. Conf. Machine Learn.*, pages 3159–3168, 2018.
[26] A. Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *Proc. 54th IEEE Annu. Sympos. Found. Comp. Sci.*, pages 253–262, 2013.
[27] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Inc., 1982.
[28] G. Peyré and M. Cuturi. Computational optimal transport. *Found. Trends Mach. Learn.*, 11(5-6):355–607, 2019.
[29] S. Raghvendra and P. K. Agarwal. A near-linear time ε-approximation algorithm for geometric bipartite matching. *J. ACM*, 67(3):1–19, June 2020.
[30] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *Intl. J. Comp. Vision*, 40(2):99–121, 2000.
[31] R. Sharathkumar. A sub-quadratic algorithm for bipartite matching of planar points with bounded integer coordinates. In *Proc. 29th Annu. Sympos. Comput. Geom.*, pages 9–16, 2013.
[32] R. Sharathkumar and P. K. Agarwal. Algorithms for transportation problem in geometric settings. In *Proc. 23rd Annu. ACM-SIAM Sympos. Discrete Algo.*, pages 306–317, 2012.
[33] J. Solomon, F. De Goes, G. Peyré, M. Cuturi, A. Butscher, A. Nguyen, T. Du, and L. Guibas. Convolutional Wasserstein distances: Efficient optimal transportation on geometric domains. *ACM Trans. Graphics*, 34(4):66:1–66:11, 2015.
[34] K. R. Varadarajan. A divide-and-conquer algorithm for min-cost perfect matching in the plane. In *Proc. 39th Annu. IEEE Sympos. Found. Comp. Sci.*, pages 320–331, 1998.
[35] K. R. Varadarajan and P. K. Agarwal. Approximation algorithms for bipartite and non-bipartite matching in the plane. In *Proc. 10th Annu. ACM-SIAM Sympos. Discrete Algo.*, pages 805–814, 1999.
[36] V. V. Vazirani. A proof of the MV matching algorithm. *arXiv:2012.03582*, 2020.