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

(extended abstract)

Pankaj K. Agarwal[†] Hsien-Chih Chang[‡] Sharath Raghvendra[§] Allen Xiao[†]

June 3, 2021

Abstract

Given point sets A and B in $\mathbb{R}^d$ where A and B have equal size n for some constant dimension d and a parameter $\varepsilon > 0$, we present the first 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 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 approach can be viewed as derandomizing the framework of Raghvendra-Agarwal; to this end several new ideas are needed to overcome various technical difficulties.

[*]The extended abstract version aims for introducing our results and new techniques from afar. All the proofs and details of the algorithms can be found in the attached full version.

[†]Department of Computer Science, Duke University, USA.

[‡]Department of Computer Science, Dartmouth College, USA. The research was conducted when the coauthor was affiliated with Duke University.

[§]Department of Computer Science, Virginia Tech, USA.

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

Also referred to as the *Earth Mover's distance* or the 1-Wasserstein distance, the EMWM problem has received considerable attention because of its applications in machine learning and computer vision [7, 28, 31]. These applications have led to substantial effort toward designing exact and approximation algorithms for computing an optimal matching. 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 all existing deterministic ε-approximation 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 [17]. The first improvement in over thirty years was made by Mądry [24] which 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] (see also Kathuria-Liu-Sidford [20]). The Hungarian algorithm computes the minimum-weight maximum cardinality matching in $O(mn)$ time [26]. If the edge costs are integers bounded by C, the Gabow-Tarjan [15] algorithm computes an optimal matching in time $O(m\sqrt{n}\log(nC))$. Faster algorithms for computing optimal matchings can be obtained by using recent min-cost max-flow algorithms using interior point methods [9]. There is also extensive work on computing optimal matchings and maximum-cost matchings in non-bipartite graphs [13, 16, 25].

For the case where A and B are points in $\mathbb{R}^2$ with an ℓ_p-norm, Vaidya [32] shows that an EMWM can be computed in $O(n^{2.5})$ time. The running time has been improved to $O(n^2 \operatorname{polylog} n)$ time [2, 3, 19]. If points have integer coordinates bounded by Δ, Raghvendra presented an $O(n^{3/2} \operatorname{polylog} n \log \Delta)$-time algorithm for computing an EMWM [29]. It is an open question whether a subquadratic algorithm exists for computing EMWM if coordinates of input points have real values. In contrast, Varadarajan [33] 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. Combining the Gabow-Tarjan's scaling method with the weighted nearest-neighbor data structure, one can obtain an $O(n^{3/2} \operatorname{poly}\log n \log(1/\varepsilon))$-time algorithm [30] to compute an ε-approximate matching. All these algorithms are restricted to the two-dimensional case because of the limitations of the dynamic nearest neighbors data structure [19]. Varadarjan and Agarwal [34] 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 [30]. 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 [5].

Randomly-shifted quadtrees have played a central role in designing Monte-Carlo approximation algorithm for EMWM. It is well-known that a simple greedy algorithm on a randomly-shifted quadtree yields (in expectation) an $O(\log n)$-approximation algorithm of EMWM. Agarwal and Varadarajan [1] 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 [18] presented an algorithm that approximates the cost of EMWM within a $O(1)$-approximation with high probability in time $O(n \operatorname{polylog} n)$. His algorithm, however, only returns the optimal cost but not the matching itself. Similary, Andoni *et al.* [6] gave a $(1+\varepsilon)$-approximation

[1]Our algorithm works for any ℓ_p-norm, but for the sake of concreteness of the presentation we use the ℓ_2-norm.

streaming algorithm to the cost that runs in $O(n^{1+o_\varepsilon(1)})$ time. Finally, Raghvendra and Agarwal [27] proposed a Monte-Carlo algorithm that computes a $(1+\varepsilon)$-approximation with high probability in $n \cdot (\varepsilon^{-1} \log n)^{O(d)}$ time. Since then, the randomized quadtree framework developed has also been successfully applied to designing near-linear-time approximation algorithm for the transportation problem and an $O(n^{5/4} \operatorname{poly}\log n)$-time ε-approximate RMS matching for points in a plane [4, 14, 21, 22].

Our results. The following theorem states the main result of the paper:

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 $(1+\varepsilon) \cdot ¢(M_{\text{opt}})$ can be computed in $n \cdot (\log n/\varepsilon)^{O(d)}$ time in the worst case.*

At a high-level, our approach can be viewed as derandomizing the approach by Raghvendra and Agarwal [27]. Roughly speaking, their algorithm defines an edge-cost function based on randomly-shifted quadtree that (in expectation) ε-approximates the Euclidean costs and computes an ε-approximate matching with respect to the new cost function. Similar to the Gabow-Tarjan algorithm, their algorithm raises the cost of every non-matching edge by a carefully chosen term θ. Using the quadtree and a hierarchical representation of the residual graph, it computes a minimum-net-cost augmenting path at each step, in time proportional to the length of the path, and augments the matching. The parameter θ is chosen small enough so that the resulting matching is an ε-approximation but is sufficiently large so that the total length of the paths remains $O(n \log n)$, leading to a near-linear-time ε-approximation algorithm.

Our algorithm replaces the randomly-shifted quadtree with a hierarchy of 2^d overlapping grid cells (alternatively, a hierarchy over the nodes of 2^d quadtrees of fixed shifts). Although the idea of replacing a randomly-shifted quadtree with a constant number (depending on d) of fixed quadtrees has been used in the past [8, 11, 12, 23], the previous algorithms decompose the problem over different quadtrees and preform a simple "aggregate" operation. It is not clear how to extend these approaches to more complex optimization problems that are not decomposable and that involve nontrivial interaction among different subproblems. For example, no deterministic near-linear-time ε-approximation algorithm is known for Euclidean TSP or EMWM that replaces a randomly-shifted quadtree by $O(1)$ quadtrees. In this paper, we show that such an approach can be made to work for EMWM, but one has to overcome a number of technical difficulties.

First, since we are working with multiple deterministic quadtrees, we cannot define a simple tree-based distance function as in Raghvendra-Agarwal [27]. Instead we work directly with Euclidean distance. Furthermore, the earlier algorithms [15, 27] define the *adjusted cost* of edges that adds a fixed penalty term to all non-matching edges. In contrast, we define adjusted cost by adding a non-uniform penalty term to all edges, depending on their Euclidean lengths. Both of these issues make the analysis of the algorithm considerably more involved.

Second, almost all matching algorithms including Hungarian, Gabow-Tarjan, and Raghvendra-Agarwal rely on computing the minimum net-cost augmenting path under an appropriate edge-cost function, which guarantees that no negative net-cost cycles are created. Due to overlapping sub-problems and working with Euclidean distance, however, we are unable to compute a minimum net-cost augmenting path efficiently at each step. Instead, our algorithm only computes an augmenting path whose net cost is within $O(\log n)$ factor of the minimum one (see FINDPATH procedure (A1)). This results in the creation of many negative net-cost cycles. We carefully and efficiently detect these cycles and eliminate them without exceeding the near-linear time bound (see AUGMENT procedure (A2)). We show that it suffices to eliminate the cycles that have "high" negative net-cost (referred to as *reducing cycles*), that the non-reducing negative cycles can be ignored, and that the combined edge-length of the augmenting paths and the reducing cycles is $O(n \log^2 n)$ (cf. Section 2).

Finally, we use quadtrees to represent the residual graph hierarchically as in [27]. In particular, at each cell $\boxplus$ of every quadtree, we compress the subgraph of the residual graph induced by $(A \cup B) \cap \boxplus$ to a properly-weighted graph $G_{\boxplus}$ of $O(\operatorname{polylog} n)$ size and compute the minimum-cost paths between all pairs of nodes of $G_{\boxplus}$ (cf. Section 3.1). To extract an augmenting path, the compressed paths are recursively expanded using those stored at the children cells of $\boxplus$. Since the children cells are overlapping and the expansions of min-cost paths in compressed graphs only correspond to approximate min-net-cost paths in the residual graph, the expansions may lead to a non-simple path P, in which a matching edge appears multiple times. Suppose $P = P_1 \circ P_2$, where P_1 and P_2 are expansions of compressed 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 "cancelling" C and ignoring P. Otherwise we remove C from P. We repeat this step until P becomes simple or a reducing cycle is found. Finding common matching edges, extracting C, splicing C, and computing the net cost of C naïvely are too expensive. So, we design a data structure that stores implicit representation of the "simple" expansions of compressed paths using $O(n \operatorname{polylog} n)$ space. It can identify common edges, simplify a path/cycle by creating a shortcut, and compute the net cost in $O(\operatorname{polylog} n)$ time each, using the implicit representation. Furthermore, the data structure can expand a compressed path from its implicit representation in time proportional to the length of the simplified path/cycle (cf. Section 3.2).

2 The 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* Π in G is a path whose edges alternate between non-matching and matching edges; Π maps to a (directed) path in $\vec{G}_M$. Alternating cycles are defined similarly. Define the *net cost* $\bar{¢}_M : E_M \to \mathbb{R}$ on $\vec{G}_M$ as follows: If $(a, b) \in A \times B$ is a non-matching edge then $\bar{¢}_M(a, b) := ¢(a, b)$, and if it is a matching edge then $\bar{¢}_M(b, a) := -¢(a, b)$. For any alternating path or cycle Π, the net cost of Π is equal to $\bar{¢}_M(\Pi) = \sum_{e \in \Pi \setminus M} ¢(e) - \sum_{e \in \Pi \cap M} ¢(e)$. If Π is a simple (non-intersecting) alternating path between two free vertices, then Π is called an *augmenting path* and $M \oplus \Pi$ is a matching of size $|M| + 1$. 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.

Preprocessing step. We perform a preprocess step so that the input is "well-conditioned" at a slight increase in the cost of optimal matching. 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; and (P3) $¢(M_{\text{opt}}) \in \left[\frac{3\sqrt{d}n}{\varepsilon}, \frac{9\sqrt{d}n}{\varepsilon}\right]$. 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.

Overview of the algorithm. We now present a high-level description of the algorithm. We begin by defining the notion of *adjusted cost* of a path, which will be crucial for our algorithm.

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 $\alpha_{\theta,M}(e) := \bar{¢}_M(e) + c_0\theta \cdot ¢(e)$, where $\bar{¢}_M(e)$ 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, cancelling a reducing cycle decreases the matching cost significantly relative to the cycle length (which is proportional to the amount of time required to cancel): $\alpha_{\bar{\varepsilon}}(\Gamma) < 0 \implies \bar{¢}(\Gamma) < -c_0\bar{\varepsilon} \cdot \|\Gamma\|$. The algorithm maintains the following invariant:

CI [Cycle Invariant]. $\alpha_{\bar{\varepsilon}}(\Gamma) \geq 0$ for every alternating cycle Γ in the beginning of each iteration.

Each iteration of the algorithm consists of two steps. First, it computes an augmenting path Π such that $\alpha_{\underline{\varepsilon}}(\Pi) \leq \alpha^*_{\bar{\varepsilon}}$. Next, it 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, 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 $\Gamma_1, \ldots, \Gamma_k$.

Lemma 3.2 in Section 3 implies that FIND-PATH takes $|\Pi| \cdot (\varepsilon^{-1} \log n)^{O(d)}$ time, and AUGMENT(Π, M) takes $(|\Pi| + \sum_i |\Gamma_i|) \cdot (\varepsilon^{-1} \log n)^{O(d)}$ time. This completes the description of the algorithm.

Analysis of the algorithm. We now analyze the cost of the matching returned by the algorithm and the running time, assuming the cycle invariant CI. Our data structure, presented later, guarantees that the cycle invariant holds.

Lemma 2.1. *The cost of any intermediate matching is at most* $(1 + \frac{\varepsilon}{2}) \cdot ¢(M_{\text{opt}})$.

Let $\Pi_1, \ldots, \Pi_n$ be the sequence of augmenting paths computed by the algorithm, and let M_i be the matching returned by the AUGMENT(Π_i, M_{i-1}). Then by Lemma 3.2, the total time spent by the algorithm is $(|\Pi| + \sum_{\Gamma \in \mathcal{N}} |\Gamma|) \cdot (\varepsilon^{-1} \log n)^{O(d)}$, where $\mathcal{N}$ is the set of alternating cycles that were cancelled throughout the calls to the AUGMENT procedure. Since the cost of each edge in G is at least 1, $|\Pi_i| \leq \|\Pi_i\|$ and $|\Gamma| \leq \|\Gamma\|$, so we will bound their Euclidean lengths instead.

Lemma 2.2. $\displaystyle\sum_i \|\Pi_i\| + \sum_{\Gamma \in \mathcal{N}} \|\Gamma\| = O\left(\frac{n \log^2 n}{\varepsilon^2}\right)$.

Lemmas 2.1, 2.2, 3.2, and 3.5 together prove Theorem 1.1.

3 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 $\mathbb{R}^d$ 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 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 $\vec{G}_M$. Π can be reported in time proportional to $|\Pi|$. The data structure also maintains a priority queue $\mathscr{Q}$ that stores the cheapest "augmenting path" of each $G_{\boxplus}$.

3.1 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* $\mathfrak{G}_1, \mathfrak{G}_2, \dots, \mathfrak{G}_{\log \Delta}$. 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. In notation, $\mathfrak{G}_i := [0,\ell_i]^d + (\ell_i\mathbb{Z})^d$. For each i, we build a collection of *cells* $\mathfrak{C}_i$ as follows: For any hypercube $\boxplus \in \mathfrak{G}_i$, we add 2^d cells into $\mathfrak{C}_i$, namely, $\boxplus + \ell_{i-1}(b_1,\dots,b_d)$ for all d-bits $b_1,\dots,b_d \in \{0,1\}$, each corresponds to a shifting of $\boxplus$ by either $+0$ or $+\ell_{i-1}$ in each dimension. Alternatively, $\mathfrak{C}_i$ is the union of 2^d different translations $\mathfrak{G}_i + \ell_{i-1}(b_1,\dots,b_d)$ of $\mathfrak{G}_i$. We refer to each hypercube $\boxplus$ as a *cell*.

Every cell in $\mathfrak{C}_i$ has its boundary lying completely on grid boundaries of $\mathfrak{G}_{i-1}, \mathfrak{G}_{i-2}, \dots, \mathfrak{G}_0$. As a consequence, cells are always perfectly tiled by lower-level grids, and grids $\mathfrak{C}_j$ are a refinement of $\mathfrak{C}_i$ for all $j<i$. The *children* of a cell $\boxplus \in \mathfrak{C}_i$ are the 3^d cells of $\mathfrak{C}_{i-1}$ contained in $\boxplus$. For any cell $\boxplus' \in \mathfrak{C}_i$ such that $\boxplus \cap \boxplus' \neq \varnothing$, we say $\boxplus'$ is a *sibling* of $\boxplus$. (For convenience $\boxplus$ is a sibling of itself.) If $\boxplus'$ is a child of $\boxplus$ then we call $\boxplus$ a *parent* of $\boxplus'$. We can now define descendants/ancestors of a cell in standard manner.

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 $i > \lceil \log \frac{\delta}{4\sqrt{d}} \rceil$ (we define grids for $i \geq 0$ but they can also be defined for $i<0$ in a straightforward manner). For each $i \in \{0,\dots,h\}$, let the set of non-empty level-i cells be $\mathscr{C}_i := \left\{\boxplus \in \mathfrak{C}_i \mid \boxplus \cap (A\cup B) \neq \varnothing\right\}$ and $\mathscr{C} = \bigcup_{i=1}^{h} \mathscr{C}_i$. For each cell $\boxplus \in \mathscr{C}$, we construct a weighted directed graph $G_{\boxplus} = (V_{\boxplus}, E_{\boxplus})$ called the *compressed graph*.

Clustering and nodes of $G_{\boxplus}$. Suppose $\boxplus \in \mathscr{C}_i$, i.e., the level of $\boxplus$ is i. Set the *level-offset* $\tau := \lceil \log(4\sqrt{d}/\delta)\rceil$. We partition $\boxplus$ into *subcells* by the hypercubes of $\mathfrak{G}_{i-\tau}$. By construction, $\boxplus$ is aligned with the grid boundaries in $\mathfrak{G}_{i-\tau}$. (When $i-\tau<0$ the grid $\mathfrak{G}_j$ for $j<0$ is still well-defined.) Note that we use the same grid to define the subcells of all cells at level i. The value of τ is chosen so that the side-length of a subcell of $\boxplus$ is at most $\frac{\delta}{4\sqrt{d}}\ell_i$ and its diameter is at most $\frac{\delta}{4}\ell_i$. For a subcell $\square$, its *children subcells* $\mathrm{Ch}(\square)$ are the hypercubes of $\mathfrak{G}_{i-\tau-1}$ that lie in $\square$. Let $X_{\boxplus}$ denote the set of subcells of $\boxplus$; we have $|X_{\boxplus}| = (\varepsilon^{-1}\log n)^{O(d)}$. Recall that $\boxplus$ has 3^d children cells; a subcell $\square$ of $\boxplus$ lies in 2^d of the children of $\boxplus$. 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 $\mathscr{A}_{\boxplus} := \left\{A_{\square} \mid \square \in X_{\boxplus} \text{ and } A_{\square} \neq \varnothing\right\}$ and $\mathscr{B}_{\boxplus} := \left\{B_{\square} \mid \square \in X_{\boxplus} \text{ and } B_{\square} \neq \varnothing\right\}$.

The nodes of $G_{\boxplus}$ are the A- and B-clusters of $\boxplus$, i.e., $V_{\boxplus} := \mathscr{A}_{\boxplus} \cup \mathscr{B}_{\boxplus}$. Observe that $|V_{\boxplus}| = (\varepsilon^{-1}\log n)^{O(d)}$. We note that each cell $\boxplus \in \mathscr{C}_0$ either contains points of A or points of B, so either $\mathscr{A}_{\boxplus} = \varnothing$ or $\mathscr{B}_{\boxplus} = \varnothing$. For level $i \geq 1$, let $\square$ be a subcell of $\boxplus$. Then $A_{\square} = \bigcup_{\Delta \in \mathrm{Ch}(\square)} A_{\Delta}$ and $B_{\square} = \bigcup_{\Delta\in\mathrm{Ch}(\square)} B_{\Delta}$. Hence, a cluster of $\boxplus$ is obtained by merging the at most 2^d clusters of children of $\boxplus$ that contain $\square$. In other words, a node of $G_{\boxplus}$ is obtained by compressing (up to) 2^d nodes from the compressed graphs of children of $\boxplus$.

Arcs and arc weights of $G_{\boxplus}$. We regard $G_{\boxplus}$ as a compressed graph of the subgraph of $\vec{G}_M$ induced by $(A\cup B)\cap\boxplus$ in which, for each A- or B-cluster, all cluster points are compressed into a single node. There are two types of arcs in $E_{\boxplus}$, the edge set of $G_{\boxplus}$:

- **Matching arcs**: For a pair of subcells $\square$ and $\square'$ in $X_{\boxplus}$, if M contains an edge $(b,a) \in B_{\square} \times A_{\square'}$ then we add an arc $(B_{\square}, A_{\square'})$ to $E_{\boxplus}$. For each pair of subcells $\square, \square'$ of $\boxplus$ such that any child cell of $\boxplus$ contains at most one of $\square, \square'$, we store the matching edges from $B_{\square}$ to $A_{\square'}$, denoted as $M_{\square,\square'} := M \cap (B_{\square} \times A_{\square'})$, in a priority queue so that we can update $M_{\square,\square'}$ and retrieve the longest edge in $M_{\square,\square'}$ in $O(\log n)$ time. We note that each edge of M appears in exactly one $M_{\square,\square'}$, so the total size of $\sum |M_{\square,\square'}|$, summed over all subcell pairs, is equal to $|M|$, which is at most n.
- **Non-matching arcs**: For a pair of subcells $\square, \square' \in X_{\boxplus}$ such that $A_{\square} \neq \varnothing, B_{\square'} \neq \varnothing$, we add the arc $(A_{\square}, B_{\square'})$ to $E_{\boxplus}$. If there is no child cell of $\boxplus$ that contains both $\square$ and $\square'$, we refer to $(A_{\square}, B_{\square'})$ as a *bridge* arc, otherwise it is an *internal* arc.

We now define a weight function $w_{\boxplus}$ on the arcs $E_{\boxplus}$ as described below. Using this weight function, we compute a minimum-weight path $\pi_{\boxplus}(X,Y)$ between all pairs of nodes X, Y in $G_{\boxplus}$. For every pair $A_{\square} \in \mathscr{A}_{\boxplus}$ and $B_{\square'} \in \mathscr{B}_{\boxplus}$, we also compute the *expansion* of $\pi_{\boxplus}(A_{\square}, B_{\square})$ to a simple (possibly empty) alternating path $\Phi_{\boxplus}(A_{\square}, B_{\square'})$ in $\vec{G}_M$ whose first and last edges are matching edges. Let b_0 (resp. a_0) be the first (resp. last) vertex of $\Phi_{\boxplus}(A_{\square}, B_{\square'})$. For a given pair $(a,b) \in A_{\square} \times B_{\square'}$, we construct an alternating path from a to b by adding the (non-matching) edges (a, b_0) and (a_0, b) at each end; if $\Phi_{\boxplus}(A_{\square}, B_{\square'}) = \varnothing$ then the path is simply (a,b). We denote this path as $(a, \Phi_{\boxplus}(A_{\square}, B_{\square'}), b)$ and refer to $\Phi_{\boxplus}(A_{\square}, B_{\square'})$ as the *flex-tip* expansion of $\pi_{\boxplus}(A_{\square}, B_{\square'})$. The construction guarantees the following condition, which we refer to as *expansion inequality*:

$$\alpha_{\underline{\varepsilon}}\left((a, \Phi_{\boxplus}(A_{\square}, B_{\square'}), b)\right) \leq w_{\boxplus}(\pi_{\boxplus}(A_{\square}, B_{\square'})). \tag{1}$$

$\Phi_{\boxplus}(A_{\square}, B_{\square'})$ is stored implicitly and can be retrieved in time proportional to its size.

While computing minimum weight paths $\pi_{\boxplus}(\cdot,\cdot)$ or their expansions $\Phi_{\boxplus}(\cdot,\cdot)$, we may discover a reducing cycle Γ in $\vec{G}_M$ (e.g., if the compressed graph $G_{\boxplus}$ contains a negative cycle) in which case we update M by cancelling Γ (i.e., setting $M = M \oplus \Gamma$) and rebuilding the data structure at $\boxplus$ and its descendants. We say cell $\boxplus$ is *stable* if (i) the children of $\boxplus$ are stable, and (ii) no reducing cycle was detected while computing $\pi_{\boxplus}(\cdot,\cdot)$ or $\Phi_{\boxplus}(\cdot,\cdot)$ for all pairs of nodes in $\mathscr{A}_{\boxplus} \times \mathscr{B}_{\boxplus}$. Assuming all children cells of $\boxplus$ are stable, then we define *arc weights* $w_{\boxplus} : E_{\boxplus} \to \mathbb{R}$ recursively as follows:

- **Matching arcs**: For a matching arc $(B_{\square}, A_{\square'}) \in E_{\boxplus}$, we define $w_{\boxplus}(B_{\square}, A_{\square'})$ to be

$$w_{\boxplus}(B_{\square}, A_{\square'}) := -\left(\max_{e \in M_{\square,\square'}} \cent(e)\right) + \delta \ell_i; \tag{2}$$

  the maximum is taken over all matching edges between $B_{\square}$ and $A_{\square'}$. If there is a child cell $\boxplus'$ of $\boxplus$ that contains both $\square$ and $\square'$, then $w_{\boxplus}(B_{\square}, A_{\square'})$ can be defined recursively as follows: Observe that

$$\bar{\cent}(B_{\square}, A_{\square'}) = \min_{e \in M_{\square,\square'}} \bar{\cent}(e) = \min_{\substack{\Delta \in \mathrm{Ch}(\square) \\ \Delta' \in \mathrm{Ch}(\square')}} \bar{\cent}(B_{\Delta}, A_{\Delta'}). \tag{3}$$

  Otherwise $\bar{\cent}(B_{\square}, A_{\square'})$ can be obtained from the priority queue that stores $M_{\square,\square'}$. We set

$$w_{\boxplus}(B_{\square}, A_{\square'}) = \bar{\cent}(B_{\square}, A_{\square'}) + \delta \ell_i. \tag{4}$$

- **Non-matching arcs**: Let $(A_{\square}, B_{\square'}) \in E_{\boxplus}$ be a non-matching arc. If $(A_{\square}, B_{\square'})$ is a bridge arc, i.e., there is no child of $\boxplus$ that contains both $\square$ and $\square'$, then we set

$$w_{\boxplus}(A_{\square}, B_{\square'}) := \|\mathrm{cntr}_{\square} - \mathrm{cntr}_{\square'}\| + \delta \ell_i, \tag{5}$$

where $\mathrm{cntr}_{\square}$ (resp. $\mathrm{cntr}_{\square'}$) is the center of $\square$ (resp. $\square'$). If $(A_{\square}, B_{\square'})$ is an internal arc, i.e., there is a child that contains both $\square$ and $\square'$, we set $w_{\boxplus}(A_{\square}, B_{\square'})$ to

$$w_{\boxplus}(A_{\square}, B_{\square'}) := \min_{\substack{\triangle \in \mathrm{Ch}(\square), \triangle' \in \mathrm{Ch}(\square') \\ \boxplus' \in \mathrm{Ch}(\boxplus) : \triangle, \triangle' \in \boxplus'}} \alpha_{\underline{\varepsilon}}\left((a_{\triangle}, \Phi_{\boxplus'}(A_{\triangle}, B_{\triangle'}), b_{\triangle'})\right) + \delta \ell_i \tag{6}$$

where $(a_{\triangle}, b_{\triangle'}) \in A_{\triangle} \times B_{\triangle'}$ is an arbitrary pair that is used to compute the weight, and $\mathrm{Ch}(\boxplus)$ is the set of children cells of $\boxplus$.

Given the weight function $w_{\boxplus}$, if $\boxplus$ is not stable (but all its children are), then let $\Gamma_{\boxplus}$ be a reducing cycle in $\vec{G}_M$ that acts as a witness of its instability. If $\boxplus$ is stable, the data structure maintains a minimum-weight path $\pi_{\boxplus}(A_{\square}, B_{\square'})$ for every pair $A_{\square} \in \mathscr{A}_{\boxplus}$ and $B_{\square'} \in \mathscr{B}_{\boxplus}$. If both $A_{\square}, B_{\square'}$ are unsaturated, then we call $\pi_{\boxplus}(A_{\square}, B_{\square'})$ an *augmenting path* in $G_{\boxplus}$. Recall that for each pair $A_{\square}, B_{\square'}$, we also compute an expansion $\Phi_{\boxplus}(A_{\square}, B_{\square'})$ of $\pi_{\boxplus}(A_{\square}, B_{\square'})$ to a simple alternating path in $\vec{G}_M$. If $\pi_{\boxplus}(A_{\square}, B_{\square'})$ is an augmenting path in $G_{\boxplus}$, we choose a pair of free points $(a_{\square}, b_{\square'}) \in A_{\square} \times B_{\square'}$ and set $\Pi_{\square,\square'} = (a_{\square}, \Phi_{\boxplus}(A_{\square}, B_{\square'}), b_{\square})$; $\Pi_{\square,\square'}$ is an augmenting path in $\vec{G}_M$. Let $(\square, \square')^*_{\boxplus} := \arg\min_{\square,\square'} \alpha_{\underline{\varepsilon}}(\Pi_{\square,\square'})$ where the minimum is taken over all pairs $A_{\square}, B_{\square'}$ such that both are unsaturated. If $G_{\boxplus}$ does not have any augmenting path, the pair $(\square, \square')^*_{\boxplus}$ is undefined. Set *OptPairs* $:= \left\{(\square, \square')^*_{\boxplus} \mid \boxplus \in \mathscr{C}\right\}$. We store *OptPairs* in a priority queue with $\alpha_{\underline{\varepsilon}}(\Pi_{(\square,\square')^*_{\boxplus}})$ as the key of $(\square, \square')^*_{\boxplus}$.

We conclude by remarking that the information stored at $\boxplus$ depends on $M \cap ((A \cap \boxplus) \times (B \cap \boxplus))$. So whenever the matching edges change, we have to update the information stored at the corresponding $\boxplus$. The REPAIR($\boxplus$) procedure, described in Section 3.3, updates the data structure at $\boxplus$. Initially $M = \varnothing$, and the data structure can be built by calling REPAIR at all cells of $\mathscr{C}$ in a bottom-up manner.

3.2 Flex-tip expansion of compressed paths

Let $\langle b_1, a_1, \ldots, a_k \rangle$ be a path in $\vec{G}_M$ from $b_1 \in B$ to $a_k \in A$. The path 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 $\langle b_1, a_1, \ldots, b_k, a_k, b_1 \rangle$. Given a pair $(a, b) \in A \times B$, we use (a, Φ, b) to denote the path $\langle a, b_1, a_1, \ldots, b_k, a_k, b \rangle$ and refer to a, b as the *tips* of Φ.

FTEs and pathlets. For a path $\pi_{\boxplus}(A_{\square}, B_{\square'})$, let the *flex-tip expansion (FTE)* $\Phi_{\boxplus}(A_{\square}, B_{\square'})$ be a simple path in $\vec{G}_M$, represented as a sequence of matching edges whose both endpoints lie in $\boxplus$ as described above. For any pair $(a, b) \in A_{\square} \times B_{\square'}$, $(a, \Phi_{\boxplus}(A_{\square}, B_{\square'}), b)$ gives a simple path in $\vec{G}_M$ from $a \in A_{\square}$ to $b \in B_{\square'}$, and the expansion inequality (1) implies that $\alpha_{\underline{\varepsilon}}\left((a, \Phi_{\boxplus}(A_{\square}, B_{\square'}), b)\right) \le w_{\boxplus}(\pi_{\boxplus}(A_{\square}, B_{\square'}))$. Recall that for an internal arc $\gamma \in G_{\boxplus}$, $w_{\boxplus}(\gamma) = \alpha_{\underline{\varepsilon}}((a_{\triangle}, \Phi_{\boxplus'}(A_{\triangle}, B_{\triangle'}), b_{\triangle'}))$ for some child cell $\boxplus'$ of $\boxplus$ and children clusters $A_{\triangle}$, $B_{\triangle'}$ of $A_{\boxplus'}$, $B_{\boxplus'}$, respectively. We define $\Phi(\gamma)$ — the FTE of γ — to be $\Phi_{\boxplus'}(A_{\triangle}, B_{\triangle'})$, again represented as a sequence of matching edges. If γ is a bridge arc then $\Phi(\gamma) = \varnothing$, and if $\gamma = (B_{\square}, A_{\square'})$ is a matching arc then we define $\Phi(\gamma)$ to be longest matching edge between $B_{\square}$ and $A_{\square'}$.

Given an FTE $\Phi = \langle e_1, \ldots, e_k \rangle$ 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 define a *pathlet* ϕ of Φ to be $e_i \blacktriangleright \Phi \blacktriangleleft e_j$ for some $1 \le i \le j \le k$. We say that ϕ *originates* from Φ.

Computing FTEs. Assuming we have constructed FTEs at the children cells of $\boxplus$ (and thus we have FTEs of all arcs of $G_{\boxplus}$ at our disposal), we construct $\Phi_{\boxplus}(A_{\square}, B_{\square'})$ as follows. Let $\pi_{\boxplus}(A_{\square}, B_{\square'}) = \langle \eta_1, \ldots, \eta_k \rangle$, for $k \le |V_{\boxplus}|$, where each η_i is an arc of $G_{\boxplus}$. We set $\tilde{\Phi} = \phi_1 \circ \phi_2 \circ \cdots \circ \phi_k$, where $\phi_i = \Phi(\eta_i)$. For every pair $i < j$, we check whether $\phi_i \cap \phi_j \neq \varnothing$. If the answer is yes, then let e be the last edge in ϕ_i that appears in ϕ_j. $\tilde{\Phi}$ contains the cycle $C := (e \blacktriangleright \phi_i) \circ \phi_{i+1} \circ \cdots \circ (\phi_j \triangleleft e)$. If $\alpha_{\underline{\varepsilon}}(C) < 0$, then C is a reducing cycle, so we abort the process and return a simple subcycle of C (which is computed using a similar procedure). Otherwise we set $\tilde{\Phi} = \phi_1 \circ \cdots \circ \phi_{i-1} \circ (\phi_i \triangleleft e) \circ (e \blacktriangleright \phi_j) \circ \cdots \phi_k$ and repeat this step until no cycles are found. Let $\Phi = \phi'_1 \circ \cdots \circ \phi'_t$ be the final sequence, assuming the process was not aborted. We set $\Phi_{\boxplus}(A_{\Box}, B_{\Box'}) \leftarrow \Phi$. Assuming that we can detect whether $\phi_i \cap \phi_j \neq \varnothing$ and compute $\alpha_{\underline{\varepsilon}}(C)$ in $(\varepsilon^{-1} \log n)^{O(d)}$ time, the total time spent in computing Φ is also $(\varepsilon^{-1} \log n)^{O(d)}$ (with a larger exponent).

Hence, it suffices to describe how we represent FTEs and pathlets compactly using a data structure, how we detect intersections between a pair of pathlets, and how we compute the adjusted cost of a path/cycle represented as a sequence of pathlets. Note that the above procedure constructs Φ as a sequence of at most $|V_{\boxplus}|$ pathlets originating from FTEs of internal/matching arcs of $G_{\boxplus}$, and these FTEs were computed at children cells of $\boxplus$. This recursive construction of FTEs suggests a natural tree data structure for storing FTEs and pathlets compactly.

FTE and pathlet trees. An FTE $\Phi = \Phi_{\boxplus}(A_{\Box}, B_{\Box'})$ is stored as a tree $T(\Phi)$ whose leaves are its matching edges, in sequence from left to right, that we call an *FTE-tree*. For a pathlet $\phi = e^- \blacktriangleright \Phi \blacktriangleleft e^+$ with $e^-, e^+ \in \Phi$, the *pathlet subtree* $T(\Phi, \phi)$ is the subtree of $T(\Phi)$ lying between and including the two root-leaf paths to e^- and e^+, which we call *spines*. $T(\Phi, \phi)$ is represented by simply storing the spines of e^- and e^+, which takes $O(h) = O(\log n)$ space. A matching edge e is represented as a single-node tree. $T(\Phi)$ is defined recursively, as follows: Recall that Φ is constructed as a sequence $\Phi = \phi_1 \circ \phi_2 \circ \cdots \circ \phi_t$, where each ϕ_j is a pathlet of a matching/internal arc η_j of $G_{\boxplus}$. $T(\Phi)$ consists of a root node plus t subtrees $T_1, \ldots, T_t$ from left to right, i.e., 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 $T_j = T(\Phi(\eta_j), \phi_j)$. We identify the j-th child of the root of $T(\Phi)$ with η_j. If ϕ_j is a trivial pathlet of a single matching edge g_j, then T_j is a single-node tree $T(g_j)$, which is a leaf of $T(\Phi)$. Recursively, each internal node u of $T(\Phi)$ is identified with an internal arc η_u of a descendant cell Δ_u of $\boxplus$, and the subtree of $T(\Phi)$ rooted at u is a pathlet subtree $T(\eta_u, \phi_u)$, where ϕ_u is the pathlet originating from η_u and consisting of matching edges stored at the leaves of this subtree. We call ϕ_u a *canonical pathlet* of η_u, and we set $\boxplus(u)$, the cell of u to be Δ_u and $\mathrm{level}(u)$ to be the level of Δ_u.

Instead of storing the entire tree explicitly, which will be too expensive, $T(\Phi)$ is compactly stored as the root of $T(\Phi)$ plus the sequence of pathlet subtrees $\phi_1, \ldots, \phi_t$ in their compact forms, i.e., simply storing the two spines of each pathlet. We call the root as well as the spine nodes of these pathlet-subtrees *exposed* nodes of $T(\Phi)$. Since there are up to $|V_{\boxplus}|$ children, the space used by this compact representation of $T(\Phi)$ is $O(|V_{\boxplus}| \cdot h)$. We show that this compact form is sufficient to access/traverse the entire FTE tree. We also show that for every canonical pathlet ϕ, there is an exposed node of an FTE tree such that ϕ is associated with that node. Using this observation, we prove that there are a total of $n \cdot (\varepsilon^{-1} \log n)^{O(d)}$ canonical pathlets, and that for any cell Δ there are $(\varepsilon^{-1} \log n)^{O(d)}$ canonical pathlets ϕ_u associated with a node of an FTE with $\boxplus(u) = \Delta$. For a pathlet ϕ of an FTE 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 as $N(T(\Phi, \phi)) = \langle N_1, N_2, \ldots, N_k \rangle$; one has $|N(T(\Phi, \phi))| = O(|V_{\boxplus}| \cdot h)$. Note that each canonical node N_i has a canonical pathlet ϕ_{N_i} associated with it and $\phi = \phi_{N_1} \circ \cdots \circ \phi_{N_k}$. Alternatively, any pathlet can be partitioned into $O(|V_{\boxplus}| \cdot h)$ canonical pathlets.

Intersection table. For an FTE Φ, we maintain the set of canonical pathlets $\mathscr{P}(\Phi)$ that originate from Φ. Let η_1, η_2 be two internal/matching arcs in sibling cells (of the same level). We maintain an *intersection table* $\beta_{\eta_1,\eta_2} : \mathscr{P}(\Phi(\eta_1)) \times \mathscr{P}(\Phi(\eta_2)) \rightarrow \{0, 1\}$ such that $\beta_{\eta_1,\eta_2}(\phi_1, \phi_2) = 1$ if $\phi_1 \cap \phi_2 \neq \varnothing$ and 0 otherwise. The total size of all intersection tables is $n \cdot (\varepsilon^{-1} \log n)^{O(d)}$. The table entries of β_{η_1,η_2} are computed dynamically as the new canonical pathlets of $\Phi(\eta_1)$ and $\Phi(\eta_2)$ are created while computing FTEs at the ancestor cell of $\boxplus$. New table values are computed through the following intersection-query algorithm, which also enables FTE-tree construction. we ensure that a table entry is in place before it is accessed.

Intersection query. Given two pathlets ϕ_1, ϕ_2 originating from two FTEs Φ_1, Φ_2 of lower levels (actually of arcs of $G_{\boxplus}$), we quickly detect whether $\phi_1 \cap \phi_2 \neq \emptyset$ as follows. The canonical pathlets associated with canonical nodes in $N(T(\Phi_i, \phi_i))$ for $i \in \{1,2\}$ partition ϕ_i into $O(|V_{\boxplus}| \cdot h)$ canonical pathlets. By testing for every pair $(N_1, N_2) \in N(T(\Phi_1, \phi_1)) \times N(T(\Phi_2, \phi_2))$, we can answer the query. Let (N_1, N_2) be a fixed pair. We detect whether $\phi_{N_1} \cap \phi_{N_2} \neq \emptyset$ as follows: If $\boxplus(N_1) \cap \boxplus(N_2) = \emptyset$, then $\phi_{N_1} \cap \phi_{N_2} = \emptyset$ so we return no. Assume that $\boxplus(N_1) \cap \boxplus(N_2) \neq \emptyset$. If $\text{level}(N_1) = \text{level}(N_2)$, then we are able to directly compare ϕ_{N_1}, ϕ_{N_2} through the intersection table: Let $\Phi(\eta_1)$ (resp. $\Phi(\eta_2)$) be the FTE from which ϕ_{N_1} (resp. ϕ_{N_2}) originates, then the table $\beta_{\eta_1,\eta_2}(\phi_{N_1}, \phi_{N_2})$ exists and stores a bit $\beta_{\eta_1,\eta_2}(\phi_{N_1}, \phi_{N_2})$ since both canonical pathlets were created at a lower level. Otherwise, assume that $\text{level}(N_1) > \text{level}(N_2)$. We recursively test the pair (N', N_2) for each child N' of N_1. Using the observation that we recurse on the children of N_1 only if $\boxplus(N_1) \cap \boxplus(N_2) \neq \emptyset$ and $\text{level}(N_1) > \text{level}(N_2)$, we show that the procedure visits only $(\varepsilon^{-1} \log n)^{O(d)}$ descendants of N_1. Summing over all pairs of canonical nodes, we conclude that the procedure takes $(\varepsilon^{-1} \log n)^{O(d)}$ time in total.

Finally, we also show that the intersection tables can also be updated within the same time bound. We also need procedures to compute adjusted costs of paths/cycles represented as a sequence of pathlets as well as to report actual paths/cycles, but they are relatively straightforward.

Omitting many details, we state the main result of this subsection:

Lemma 3.1. *After having computed FTEs at the children cells of $\boxplus$ and the minimum weight paths for all pairs $(A_{\square}, B_{\square'})$ between subcells $\square, \square'$ of $\boxplus$, implicit representations of their FTEs can be computed in $(\varepsilon^{-1} \log n)^{O(d)}$ time, and the data structure can updated within the same time bound. The process may abort and return a reducing cycle.*

3.3 FindPath, Augment, and Repair procedures

We describe the operations FindPath, Augment, and Repair performed on the data structure built on each cell $\boxplus$. The first two procedures have been defined in Section 2. Repair($\boxplus$) builds the data structure at $\boxplus$, and is a subroutine for Augment. For an alternating path Π in $\vec{G}_M$, we call a cell $\boxplus \in \mathscr{C}$ *affected* by Π if $\boxplus$ contains a vertex of Π (that is, a point in $\mathbb{R}^d$). Let $\mathscr{C}_\Pi \subseteq \mathscr{C}$ denote the set of cells affected by Π.

FindPath(): It performs a DeleteMin operation on the priority queue storing the candidate subcell pairs *OptPairs* and retrieves a pair $(\square, \square')^*_{\boxplus}$, such that $\Pi_{(\square,\square')^*_{\boxplus}} = (a_{\square}, \Phi_{\boxplus}(A_{\square}, B_{\square'}), b_{\square'})$ is an augmenting path, i.e., $a_{\square}, b_{\square'}$ are free vertices. Given an implicit representation of $\Phi := \Phi_{\boxplus}(A_{\square}, B_{\square'})$, we call Report to retrieve Φ, add the tips $a_{\square}, b_{\square'}$ to Φ, and return the path $(a_{\square}, \Phi, b_{\square'})$. Since Φ is simple, $a_{\square}, b_{\square'}$ are free vertices and not in Φ, (a, Φ, b) is a (simple) augmenting path. 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 $\mathscr{C}_\Pi$, the set of cells affected by Π, in a priority queue $\mathscr{Q}_\Pi$ with the level of the cell as its key. At each step, we retrieve a cell $\boxplus$ from $\mathscr{Q}_\Pi$ 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 $M = M \oplus \Gamma$ and insert all cells in $\mathscr{C}_\Gamma$ into $\mathscr{Q}_\Pi$ and continue. This process continues until $\mathscr{Q}_\Pi$ becomes empty. The procedure then returns the matching 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 $\left(\sum_i |\Gamma_i| + |\Pi|\right) \cdot (\varepsilon^{-1} \log n)^{O(d)}$ time.

Repair($\boxplus$): Recall that 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 a pair of subcells of $\boxplus$.

- If no child cell of $\boxplus$ contains both $\square$ and $\square'$, then we update $M_{\square,\square'}$, i.e., delete the edges that are no longer in M and insert newly created edges of M whose endpoints lie in $\square$ and $\square'$. We use (4) and (5) 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 (3) and (6) to compute the weights of arcs between the clusters of $\square, \square'$. We compute $\alpha_{\bar{\varepsilon}}\big((a_\triangle, \Phi_{\boxplus'}(A_\triangle, B_{\triangle'}), b_\triangle)\big)$, which we need in (6).

iii. (**APSP computation**) We compute the minimum-weight 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 (an implicit representation of) a simple reducing cycle Γ. In this case, we abort the update of $\boxplus$ and return the cycle Γ. which retrieves the actual simple cycle $\hat{\Gamma}$.

iv. (**Computing expansions**) For every pair of subcells $\square, \square'$, we compute FTEs $\Phi_\boxplus(A_\square, B_{\square'})$ 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 Γ. If we succeed in computing expansions for all pairs, we mark $\boxplus$ as stable.

v. (**Augmenting paths**) For each pair of subcells $\square, \square'$ of $\boxplus$ such that both $A_\square$ and $B_{\square'}$ are unsaturated, we choose a pair of free vertices $a \in A_\square$ and $b \in B_{\square'}$ and set $\Pi_{\square,\square'} = (a, \Phi_\boxplus(\square, \square'), b)$. Otherwise, $\Pi_{\square,\square'}$ is undefined and $\alpha_{\underline{\varepsilon}}(\Pi_{\square,\square'}) = \infty$. We compute $(\square, \square')^*_\boxplus = \arg\min_{\square,\square'} \alpha_{\underline{\varepsilon}}(\Pi_{\square,\square'})$ and insert $(\square, \square')^*_\boxplus$ into *OptPairs*.

Putting everything together, we obtain the following:

Lemma 3.2. 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* $\big(|\Pi| + \sum_i |\Gamma_i|\big) \cdot (\varepsilon^{-1} \log n)^{O(d)}$ *time, where* $\{\Gamma_i\}$ *are the reducing cycles canceled by the procedure.*

Proof of correctness. The correctness of FINDPATH and AUGMENT and the invariant CI are implied by Lemmas 3.3 and 3.4, which relate the net costs of residual paths and cycles to the weights of compressed paths.

Lemma 3.3 (Expansion Inequality). *(i) Let* $\boxplus$ *be a cell, let* $\square, \square'$ *be two subcells in* $\boxplus$*, and let* $\Phi_\boxplus(A_\square, B_{\square'})$ *be the flex-tip expansion of* $\pi_\boxplus(A_\square, B_{\square'})$*. For any pair* $(a, b) \in A_\square \times B_{\square'}$*, let* $\Pi_{ab} = (a, \Phi_\boxplus(A_\square, B_{\square'}), b)$ *be the alternating path in* $\vec{G}_M$ *from* a *to* b *induced by* $\Phi_\boxplus(A_\square, B_{\square'})$*. Then,*

$$\alpha_{\underline{\varepsilon}}(\Pi_{ab}) \le w_\boxplus(\pi_\boxplus(A_\square, B_{\square'})) \qquad \text{for all } (a, b) \in A_\square \times B_{\square'}.$$

(ii) If C *is a negative cycle in* $G_\boxplus$ *for some* $\boxplus \in \mathscr{C}$*, then* $\alpha_{\underline{\varepsilon}}(\Phi_\boxplus(C)) < 0$*.*

Lemma 3.4 (Lifting Inequality). *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). Let* $\boxplus$ *be a cell at the smallest level that contains* Π *and* $X(p)$ *(resp.* $X(q)$*) the cluster in* $V_\boxplus$ *containing* p *(resp.* q*). Then* $w_\boxplus(\pi_\boxplus(X(p), X(q))) \le \alpha_{\bar{\varepsilon}}(\Pi)$*.*

By combining the Lifting and Expansion Inequalities, we obtain the following statements that prove the correctness of the algorithm.

Lemma 3.5. *(i)* FINDPATH *returns an augmenting path* Π *such that* $\alpha_{\underline{\varepsilon}}(\Pi) \le \alpha^*_{\bar{\varepsilon}, M}$*. (ii) If* $\vec{G}_M$ *contains an alternating cycle* Γ *with* $\alpha_{\bar{\varepsilon}}(\Gamma) < 0$ *then* AUGMENT *returns a reducing cycle. (iii) The invariant* CI *holds at the beginning of each iteration.*

References

[1] P. Agarwal and K. Varadarajan. A near-linear constant-factor approximation for Euclidean bipartite matching? In *Proceedings of the twentieth annual symposium on Computational geometry*, page 247, 2004.

[2] P. K. Agarwal, H.-C. Chang, and A. Xiao. Efficient Algorithms for Geometric Partial Matching. In G. Barequet and Y. Wang, editors, *35th International Symposium on Computational Geometry (SoCG 2019)*, volume 129 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[3] P. K. Agarwal, A. Efrat, and M. Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM Journal on Computing; Philadelphia*, 29(3):42, 2000.

[4] P. K. Agarwal, K. Fox, D. Panigrahi, K. R. Varadarajan, and A. Xiao. Faster algorithms for the geometric transportation problem. In *Proc. 33rd International Symposium on Computational Geometry*, pages 7:1–7:16, 2017.

[5] P. K. Agarwal and R. Sharathkumar. Approximation algorithms for bipartite matching with metric and geometric costs. In *Proc. ACM Symposium on Theory of Computing*, pages 555–564, 2014.

[6] A. Andoni, A. Nikolov, K. Onak, and G. Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 574–583, New York New York, May 2014. ACM.

[7] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *Proc. 34th International Conference on Machine Learning*, pages 214–223, 2017.

[8] S. Ashur and S. Har-Peled. On Undecided LP, Clustering and Active Learning. In K. Buchin and É. Colin de Verdière, editors, *37th International Symposium on Computational Geometry (SoCG 2021)*, volume 189 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:15, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[9] J. v. d. Brand, Y. T. Lee, Y. P. Liu, T. Saranurak, A. Sidford, Z. Song, and D. Wang. Minimum cost flows, mdps, and ℓ_1-regression in nearly linear time for dense instances. *arXiv e-prints*, 2101:arXiv:2101.05719, Jan. 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 Annual Symposium on Foundations of Computer Science*, pages 919–930, 2020.

[11] T. M. Chan. Approximate nearest neighbor queries revisited. *Discret. Comput. Geom.*, 20(3):359–373, 1998.

[12] T. M. Chan, S. Har-Peled, and M. Jones. On locality-sensitive orderings and their applications. In *10th Innovations in Theoretical Computer Science Conference*, volume 124, pages 21:1–21:17, 2019.

[13] R. Duan and S. Pettie. Approximating maximum weight matching in near-linear time. In *Proc. 51st Annual IEEE Sympos. on Foundat. Comput. Sc.*, pages 673–682, 2010.

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

[15] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18(5):1013–1036, Oct. 1989.

[16] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for general graph-matching problems. *J. ACM*, 38(4):815–853, 1991.

[17] J. Hopcroft and R. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[18] P. Indyk. A near linear time constant factor approximation for Euclidean bichromatic matching (cost). In *SODA 2007*, page 4, 2007.

[19] H. Kaplan, W. Mulzer, L. Roditty, P. Seiferth, and M. Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2495–2504. Society for Industrial and Applied Mathematics, Jan. 2017.

[20] T. Kathuria, Y. P. Liu, and A. Sidford. Unit capacity maxflow in almost $o(m^4/3)$ time. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 119–130, Nov. 2020.

[21] A. B. Khesin, A. Nikolov, and D. Paramonov. Preconditioning for the geometric transportation problem. In *Proc. 35th Annual Symposium on Computational Geometry*, pages 15:1–15:14, 2019.

[22] N. Lahn and S. Raghvendra. An $\tilde{O}(n^{5/4})$ time ε-approximation algorithm for rms matching in a plane. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, pages 869–888. Society for Industrial and Applied Mathematics, Jan. 2021.

[23] H. Le and S. Solomon. Truly optimal Euclidean spanners. In D. Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science*, pages 1078–1100, 2019.

[24] A. Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *Foundations of Computer Science (FOCS)*, pages 253–262. IEEE, 2013.

[25] S. Micali and V. V. Vazirani. An $O(\sqrt{|v|} \cdot |E|)$ algorithm for finding maximum matching in general graphs. In *FOCS*, pages 17–27, 1980.

[26] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., 1982.

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

[28] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.

[29] R. Sharathkumar. A sub-quadratic algorithm for bipartite matching of planar points with bounded integer coordinates. In *Proc. 29th Annual ACM Symposium on Computational Geometry*, pages 9–16, 2013.

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

[31] 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 Transactions on Graphics*, 34(4):66, 2015.

[32] P. M. Vaidya. Geometry helps in matching. *SIAM J. Comput.*, 18:1201–1225, December 1989.

[33] K. R. Varadarajan. A divide-and-conquer algorithm for min-cost perfect matching in the plane. In *Proc. 39th Annual IEEE Symposium on Foundations of Computer Science*, pages 320–331, 1998.

[34] K. R. Varadarajan and P. K. Agarwal. Approximation algorithms for bipartite and non-bipartite matching in the plane. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 17-19 January 1999, Baltimore, Maryland, USA.*, pages 805–814, 1999.

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

Pankaj K. Agarwal[*] Hsien-Chih Chang[†] Sharath Raghvendra[‡] Allen Xiao[*]

June 3, 2021

Abstract

Given point sets A and B in $\mathbb{R}^d$ where A and B have equal size n for some constant dimension d and a parameter $\varepsilon > 0$, we present the first 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 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 approach can be viewed as derandomizing the framework of Raghvendra-Agarwal; to this end several new ideas are needed to overcome various technical difficulties.

[*]Department of Computer Science, Duke University, USA.

[†]Department of Computer Science, Dartmouth College, USA. The research was conducted when the coauthor was affiliated with Duke University.

[‡]Department of Computer Science, Virginia Tech, USA.

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

Also referred to as the *Earth Mover's distance* or the 1-Wasserstein distance, the EMWM problem has received considerable attention because of its applications in machine learning and computer vision [7, 30, 33]. These applications have led to substantial effort toward designing exact and approximation algorithms for computing an optimal matching. 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 all existing deterministic ε-approximation 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 was made by Mądry [26] which 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] (see also Kathuria-Liu-Sidford [22]). The Hungarian algorithm computes the minimum-weight maximum cardinality matching in $O(mn)$ time [28]. If the edge costs are integers bounded by C, the Gabow-Tarjan [16] algorithm computes an optimal matching in time $O(m\sqrt{n}\log(nC))$. Faster algorithms for computing optimal matchings can be obtained by using recent min-cost max-flow algorithms using interior point methods [9]. There is also extensive work on computing optimal matchings and maximum-cost matchings in non-bipartite graphs [14, 17, 27].

For the case where A and B are points in $\mathbb{R}^2$ with an ℓ_p-norm, Vaidya [34] shows that an EMWM can be computed in $O(n^{2.5})$ time. The running time has been improved to $O(n^2 \operatorname{polylog} n)$ time [2, 3, 21]. If points have integer coordinates bounded by Δ, Raghvendra presented an $O(n^{3/2} \operatorname{polylog} n \log \Delta)$-time algorithm for computing an EMWM [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 [35] 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. Combining the Gabow-Tarjan's scaling method with the weighted nearest-neighbor data structure, one can obtain an $O(n^{3/2} \operatorname{poly}\log n \log(1/\varepsilon))$-time algorithm [32] to compute an ε-approximate matching. All these algorithms are restricted to the two-dimensional case because of the limitations of the dynamic nearest neighbors data structure [21]. Varadarjan and Agarwal [36] 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 $0 < \delta \le 1$, they also proposed a deterministic $O(1/\delta)$-approximation algorithm that runs in $O(n^{1+\delta} \log^d n)$ time [5].

Randomly-shifted quadtrees have played a central role in designing Monte-Carlo approximation algorithm for EMWM. It is well-known that a simple greedy algorithm on a randomly-shifted quadtree yields (in expectation) an $O(\log n)$-approximation algorithm of EMWM. Agarwal and Varadarajan [1] 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 a $O(1)$-approximation with high probability in time $O(n \operatorname{polylog} n)$. His algorithm, however, only returns the optimal cost but not the matching itself. Andoni *et al.* [6] gave a $(1+\varepsilon)$-approximation streaming

[1] Our algorithm works for any ℓ_p-norm, but for the sake of concreteness of the presentation we use the ℓ_2-norm.

algorithm that runs in $O(n^{1+o_\varepsilon(1)})$ time. Finally, Raghvendra and Agarwal [29] proposed a Monte-Carlo algorithm that computes a $(1+\varepsilon)$-approximation with high probability in $n \cdot (\varepsilon^{-1} \log n)^{O(d)}$ time. Since then, the randomized quadtree framework developed has also been successfully applied to designing near-linear-time approximation algorithm for the transportation problem and an $O(n^{5/4} \operatorname{poly}\log n)$-time ε-approximate RMS matching for points in a plane [4, 15, 23, 24].

Our results. The following theorem states the main result of the paper:

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 $(1+\varepsilon) \cdot ¢(M_{\text{opt}})$ can be computed in $n \cdot (\log n/\varepsilon)^{O(d)}$ time in the worst case.*

At a high-level, our approach can be viewed as derandomizing the approach by Raghvendra and Agarwal [29]. Roughly speaking, their algorithm defines an edge-cost function based on randomly-shifted quadtree that (in expectation) ε-approximates the Euclidean costs and computes an ε-approximate matching with respect to the new cost function. Similar to the Gabow-Tarjan algorithm, their algorithm raises the cost of every non-matching edge by a carefully chosen term θ. Using the quadtree and a hierarchical representation of the residual graph, it computes a minimum-net-cost augmenting path at each step, in time proportional to the length of the path, and augments the matching. The parameter θ is chosen small enough so that the resulting matching is an ε-approximation but is sufficiently large so that the total length of the paths remains $O(n \log n)$, leading to a near-linear-time ε-approximation algorithm.

Our algorithm replaces the randomly-shifted quadtree with a hierarchy of 2^d overlapping grid cells (alternatively, a hierarchy over the nodes of 2^d quadtrees of fixed shifts). Although the idea of replacing a randomly-shifted quadtree with a constant number (depending on d) of fixed quadtrees has been used in the past [8, 12, 13, 25], the previous algorithms decompose the problem over different quadtrees and preform a simple "aggregate" operation. It is not clear how to extend these approaches to more complex optimization problems that are not decomposable and that involve nontrivial interaction among different subproblems. For example, no deterministic near-linear-time ε-approximation algorithm is known for Euclidean TSP or EMWM that replaces a randomly-shifted quadtree by $O(1)$ quadtrees. In this paper, we show that such an approach can be made to work for EMWM, but one has to overcome a number of technical difficulties.

First, since we are working with multiple deterministic quadtrees, we cannot define a simple tree-based distance function as in Raghvendra-Agarwal [29]. Instead we work directly with Euclidean distance. Furthermore, the earlier algorithms [16, 29] define the *adjusted cost* of edges that adds a fixed penalty term to all non-matching edges. In contrast, we define adjusted cost by adding a non-uniform penalty term to all edges, depending on their Euclidean lengths. Both of these issues make the analysis of the algorithm considerably more involved.

Second, almost all matching algorithms including Hungarian, Gabow-Tarjan, and Raghvendra-Agarwal rely on computing the minimum net-cost augmenting path under an appropriate edge-cost function, which guarantees that no negative net-cost cycles are created. Due to overlapping sub-problems and working with Euclidean distance, however, we are unable to compute a minimum net-cost augmenting path efficiently at each step. Instead, our algorithm only computes an augmenting path whose net cost is within $O(\log n)$ factor of the minimum one (see FINDPATH procedure (A1)). This results in the creation of many negative net-cost cycles. We carefully and efficiently detect these cycles and eliminate them without exceeding the near-linear time bound (see AUGMENT procedure (A2)). We show that it suffices to eliminate the cycles that have "high" negative net-cost (referred to as *reducing cycles*), that the non-reducing negative cycles can be ignored, and that the combined edge-length of the augmenting paths and the reducing cycles is $O(n \log^2 n)$ (cf. Section 2).

Finally, we use quadtrees to represent the residual graph hierarchically as in [29]. In particular, at each cell $\boxplus$ of every quadtree, we compress the subgraph of the residual graph induced by $(A \cup B) \cap \boxplus$ to a properly-weighted graph $G_{\boxplus}$ of $O(\operatorname{polylog} n)$ size and compute the minimum-cost paths between all pairs of nodes of $G_{\boxplus}$ (cf. Section **??**). To extract an augmenting path, the compressed paths are recursively expanded using those stored at the children cells of $\boxplus$. Since the children cells are overlapping and the expansions of min-cost paths in compressed graphs only correspond to approximate min-net-cost paths in the residual graph, the expansions may lead to a non-simple path P, in which a matching edge appears multiple times. Suppose $P = P_1 \circ P_2$, where P_1 and P_2 are expansions of compressed 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 "cancelling" C and ignoring P. Otherwise we remove C from P. We repeat this step until P becomes simple or a reducing cycle is found. Finding common matching edges, extracting C, splicing C, and computing the net cost of C naïvely are too expensive. So, we design a data structure that stores implicit representation of the "simple" expansions of compressed paths using $O(n \operatorname{polylog} n)$ space. It can identify common edges, simplify a path/cycle by creating a shortcut, and compute the net cost in $O(\operatorname{polylog} n)$ time each, using the implicit representation. Furthermore, the data structure can expand a compressed path from its implicit representation in time proportional to the length of the simplified path/cycle (cf. Section 3.3). Our data structure has similarities to the one-dimensional range-searching data structure and may be of independent interest.

2 The Overall Algorithm

In this section we give a high-level description of our algorithm and analyze its performance. We begin by describing a few basic definitions related to matching. Next, we describe the preprocessing step that ensures the input to be "well-conditioned". We then describe the algorithm and finally analyze its performance.

Due to the amount of ideas, tools, and technicalities introduced, a table of notation is included at the end of the paper for easy reference.

2.1 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. A free vertex in A (resp. B) must be a *source* (resp. *sink*), that is, its in-degree (resp. out-degree) is 0. A matched vertex in A (resp. B) has in-degree (resp. out-degree) 1. An *alternating path* Π in G is a path whose edges alternate between non-matching and matching edges; Π maps to a (directed) path in $\vec{G}_M$. Conversely, a path in $\vec{G}_M$ corresponds to an alternating path in G. Alternating cycles are defined similarly.

If graph G has edge costs $¢ : E \to \mathbb{R}_{\geq 0}$, then we define edge costs in $\vec{G}_M$, called the *net cost* and denoted by $\bar{¢}_M : E_M \to \mathbb{R}$, as follows.

- If $(a, b) \in A \times B$ is a non-matching edge, then $\bar{¢}_M(a, b) := ¢(a, b)$.
- If $(a, b) \in A \times B$ is a matching edge, then $\bar{¢}_M(b, a) := -¢(a, b)$.

For a (multi-)subset $X \subseteq E_M$, we set $\bar{¢}_M(X) := \sum_{e \in X} \bar{¢}(e)$. For any alternating path or cycle Π, the net cost of Π is equal to

$$\bar{¢}(\Pi) = \sum_{e \in \Pi \setminus M} ¢(e) - \sum_{e \in \Pi \cap M} ¢(e).$$

If Π is an alternating path or cycle, we use the following shorthand for the arc length of Π (as a polygonal curve):

$$\|\Pi\| := \sum_{e \in \Pi} \|e\|.$$

If Π is a simple (non-intersecting) alternating path between two free vertices, then Π is called an *augmenting path* and $M \oplus \Pi$ is a matching of size $|M|+1$. If Π is an alternating cycle, then $|M \oplus \Pi| = |M|$ instead. We refer to the operation $M \oplus \Pi$ as *augmenting* M by Π, and in the specific case where Π is an alternating cycle, *cancelling* the alternating cycle Π. If no augmenting path exists, then M is maximum in size. We observe that $¢(M \oplus \Pi) = ¢(M) + \bar{¢}(\Pi)$.

The classical Hungarian algorithm for computing a minimum-cost maximum matching in G repeatedly finds an augmenting path Π in $\vec{G}_M$ of minimum net cost and augments the matching by Π. Since it finds a minimum-net-cost augmenting path at each step, the residual graph never contains a cycle of negative net cost, which implies that optimality of the matching computed by the algorithm. Conversely, if a perfect matching M is not optimal, there is always a residual cycle Γ in $\vec{G}_M$ with $\bar{¢}_M(\Gamma) < 0$, that is, $¢(M \oplus \Gamma) < ¢(M)$.

2.2 Preprocessing step

We now describe the preprocessing step that makes the input "well-conditioned" at a slight increase in the cost of optimal matching. It consists of two stages.

A coarse approximation. The first stage computes an $O(n^2)$ approximation of $¢(M_{\text{opt}})$, as follows: Following the algorithm of Callahan-Kosaraju [11] (see also Har-Peled [18, Chapter 4]) we can construct in $O(n \log n)$ time a weighted graph H on $A \cup B$ with edge weights $w_H(p,q) = \|p-q\|$, such that for any pair of points (p,q) in $A \cup B$, there is a path P in H from p to q such that $w_H(P) \le 2\|p-q\|$. (In other words, graph H is a *Euclidean 2-spanner* of $A \cup B$.) Next, we execute Kruskal's minimum-spanning tree algorithm on H until each connected component X of the forest has equal number of points of A and B. Let w_0 be the weight of the last edge added by the algorithm.

Lemma 2.1. $\frac{w_0}{2} \le ¢(M_{\text{opt}}) < n^2 \cdot w_0$.

Proof: Let e_0 be the last edge added by the algorithm that connected two clusters X_1 and X_2 and merged them into a single cluster X. By assumption, $|X \cap A| = |X \cap B|$ but $|X_i \cap A| \ne |X_i \cap B|$ for $i \in \{1,2\}$ (because otherwise the algorithm would have stopped sooner). Suppose $|X_1 \cap A| > |X_1 \cap B|$. Then there is an edge $(p,q) \in M_{\text{opt}}$ whose one endpoint is in $A \cap X_1$ and another endpoint in $B \setminus X_1$. By construction of H, there is a path π in H from p to q such that $w_H(\pi) \le 2\|p-q\|$. The path must contain an edge $e' = (p', q')$ such that $p' \in X_1$ to $q' \notin X_1$. Since the algorithm chose to add e_0 instead of (p',q'), we have

$$w_0 = \|e_0\| \le \|p' - q'\| \le w_H(\pi) \le 2\|p-q\| \le 2¢(M_{\text{opt}}).$$

Hence, $¢(M_{\text{opt}}) \ge w_0/2$.

Next, let $X_1, \ldots, X_k$ be the connected components of the forest maintained by Kruskal's algorithm after e_0 was added. For each X_i, let M_i be an arbitrary perfect matching between $A \cap X_i$ and $B \cap X_i$. By triangle inequality, the distance between any two points of X_i (and in particular, any matching edge) is at most $(|X_i|-1)w_0 < nw_0$. Hence, $¢(M_{\text{opt}}) \le ¢(\bigcup_i M_i) \le n^2 \cdot w_0$, as claimed. $\square$

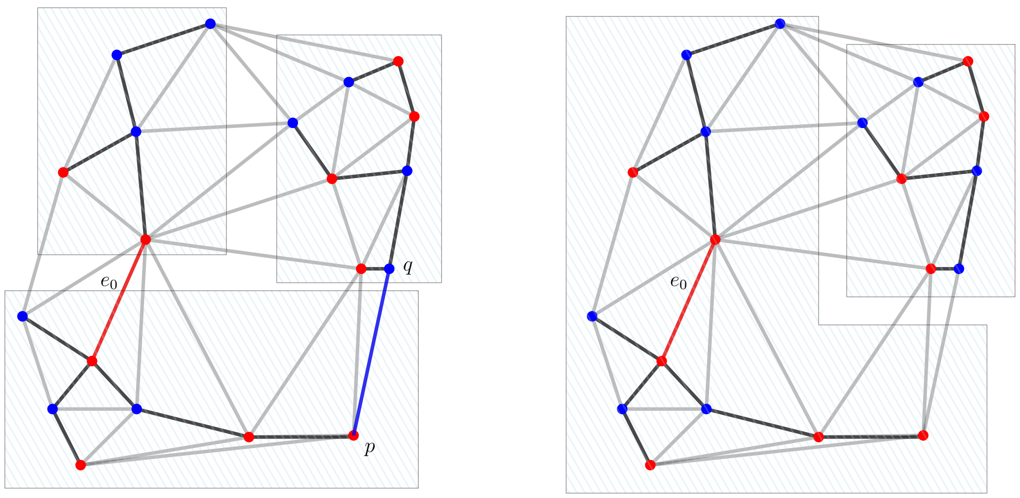


Figure 2.1. Proof of Lemma 2.1. Edges of the 2-spanner (H) are in gray, the included MST edges are in black, shaded regions denote connected components. The last edge added, e_0, is in red. **Left**: Partitions reflect the state right before e_0 is added. $(p, q) \in M_{\text{opt}}$ (blue) crosses out of the bottom partition. **Right**: e_0 is the longest MST edge out of all included edges, and component diameter is bounded by applying triangle inequality.

For any integer $i \in [-1, 2\log n]$[2], define $\beta_i := 2^i \cdot w_0$. By Lemma 2.1, there is an i such that $\beta_{i-1} \leq ¢(M_{\text{opt}}) \leq \beta_i$. We run our algorithm for at most $c_0 \cdot n \operatorname{polylog} n$ steps, where c_0 is a sufficiently large constant specified below, on each choice of β_i. In the i-th iteration, either the algorithm returns a perfect matching of A and B or terminates without computing a perfect matching. Among the perfect matchings computed by the algorithm, we return the one with the smallest cost. Theorem 1.1 ensures that if $\beta_{i-1} \leq ¢(M_{\text{opt}}) \leq \beta_i$ then the algorithm returns a perfect matching of cost at most $(1+\varepsilon) \cdot ¢(M_{\text{opt}})$. Now forward, we assume that we have computed a value $\beta > 0$ such that

$$¢(M_{\text{opt}}) \leq \beta \leq 2 \cdot ¢(M_{\text{opt}}). \tag{1}$$

Conditioning the input. We multiply the coordinates of each point by a factor of $\frac{8\sqrt{d}n}{\varepsilon\beta}$. By (1), the cost of the optimal matching now lies in the range $\left[\frac{4\sqrt{d}n}{\varepsilon}, \frac{8\sqrt{d}n}{\varepsilon}\right]$. Next, we move each input point to its nearest integer grid point. Since each point is moved a distance of at most $\frac{\sqrt{d}}{2}$, the cost of optimal matching of perturbed points increases by $\frac{\sqrt{d}}{2}n \leq \frac{\varepsilon}{8}¢(M_{\text{opt}})$. If a grid point contains both red and blue points, we match them and delete them — this is optimal by triangle inequality. Hence, we assume each grid point contains points of at most one color. Note that multiple points of one color may map to the same grid point, so we have a multiset of points. Abusing the notation a little, we treat each copy as a separate point and let A and B denote the resulting (multi-)sets of perturbed points. By construction, the cost of optimal matching on the perturbed points $¢(M_{\text{opt}})$ lies in

$$\left[\frac{4\sqrt{d}n}{\varepsilon} - \frac{\sqrt{d}n}{2}, \frac{8\sqrt{d}n}{\varepsilon} + \frac{\sqrt{d}n}{2}\right] \subseteq \left[\frac{3\sqrt{d}n}{\varepsilon}, \frac{9\sqrt{d}n}{\varepsilon}\right]$$

since we assume $\varepsilon \leq 1$.

After this preprocessing step, which takes $O(n\log^2 n)$ time in total for any fixed d, we have point sets A and B that satisfy the following three properties:

[2]unless specified otherwise, all the logarithms are of base 2 throughout this paper

P1. All input points have integer coordinates.

P2. No integer grid point contains points of both A and B.

P3. $\text{¢}(M_{\text{opt}}) \in \left[\frac{3\sqrt{d}n}{\varepsilon}, \frac{9\sqrt{d}n}{\varepsilon}\right]$.

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.

2.3 An overview of the algorithm

We now present a high-level description of the algorithm. We begin by defining the notion of *adjusted cost* of a path, which will be crucial for our algorithm.

Adjusted cost. Let c_0 be a constant whose value will be chosen later (see Lemma 3.7). Let M be any fixed matching. For a parameter $\theta \geq 0$, we define the *θ-adjusted cost* of an edge $e = (a, b)$ to be

$$\alpha_{\theta,M}(e) := \bar{\text{¢}}_M(e) + c_0\theta \cdot \|a - b\|,$$

where $\bar{\text{¢}}_M(e)$ is the net cost of e in the residual graph $\vec{G}_M$. For a set X of edges in $\vec{G}_M$, define

$$\alpha_{\theta,M}(X) := \sum_{e\in X} \alpha_{\theta,M}(e) = \bar{\text{¢}}_M(X) + c_0\theta \cdot \|X\|.$$

Set X will often be an alternating path or cycle. We can interpret $\alpha_{\theta,M}$ as adding a *regularizer* to the net cost of a residual path or cycle; see the remark at the end of Section 2.

We are now ready to describe the algorithm. 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 whose values will be specified in the next section (see the proof of Lemma 3.9). For a matching M, we define

$$\alpha^*_{\bar{\varepsilon},M} = \min_{\Pi:\text{ augmenting path in }\vec{G}_M} \alpha_{\bar{\varepsilon},M}(\Pi).$$

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, cancelling a reducing cycle decreases the matching cost significantly relative to the cycle length (which is proportional to the amount of time required to cancel): $\alpha_{\bar{\varepsilon}}(\Gamma) < 0 \implies \bar{\text{¢}}(\Gamma) < -c_0\bar{\varepsilon} \cdot \|\Gamma\|$. The algorithm maintains the following invariant:

CI [Cycle Invariant]. $\alpha_{\bar{\varepsilon}}(\Gamma) \geq 0$ for every alternating cycle Γ in the beginning of each iteration.

Each iteration of the algorithm consists of two steps. First, it computes an augmenting path Π such that $\alpha_{\underline{\varepsilon}}(\Pi) \leq \alpha^*_{\bar{\varepsilon}}$. Next, it 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 set of simple reducing cycles $\Gamma_1, \ldots \Gamma_k$ such that $\alpha_{\underline{\varepsilon}}(\Gamma_i) < 0$ for all i.

Lemma 3.2 in Section 3 implies that Find-Path takes $|\Pi| \cdot (\varepsilon^{-1}\log n)^{O(d)}$ time, and Augment(Π, M) takes $(|\Pi| + \sum_i |\Gamma_i|) \cdot (\varepsilon^{-1}\log n)^{O(d)}$ time. This completes the description of the algorithm. Let M_{alg} be the perfect matching returned by the algorithm.

```
FAST-MATCHING(A, B):
    Input: Point sets $A$ and $B$ satisfying (P1)–(P3) and $\varepsilon > 0$
    $M \leftarrow \varnothing$
    repeat:
        $\Pi \leftarrow$ FIND-PATH()
        $M \leftarrow$ AUGMENT($\Pi, M$)
    until $|M| = |A| = |B|$
```

Figure 2.2. Implementation of the main algorithm (after preprocessing) using the data structure.

2.4 Analysis of the algorithm

We now analyze the cost of M_{alg} and the running time of the algorithm, assuming the cycle invariant CI. Our data structure, presented later, guarantees that the cycle invariant holds (see Corollary 3.11).

Lemma 2.2. *The following two statements hold throughout the algorithm:*

i. The cost of any intermediate matching is at most $(1+\frac{\varepsilon}{2})\cdot ¢(M_{\text{opt}})$.

ii. In the beginning of each iteration when the invariant CI is satisfied, let M be the current matching and let Π be any augmenting path such that $\alpha_{\underline{\varepsilon}}(\Pi) \le \alpha^*_{\bar{\varepsilon}}$. Then,

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

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, $¢(M) \le (1+\frac{\varepsilon}{2})\cdot ¢(M_{\text{opt}})$.

If $\alpha^* < 0$, then

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

Since FINDPATH returns an augmenting path P with $\bar{¢}(P) \le \alpha_{\underline{\varepsilon}}(P) \le \alpha^* < 0$,

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

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

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

Again, FINDPATH returns an augmenting path P with $\alpha_{\underline{\varepsilon}}(P) \le \alpha^*_{\bar{\varepsilon}}$. Therefore

$$¢(M \oplus P) = ¢(M) + \bar{¢}(P) \le ¢(M) + \left(1+\frac{\varepsilon}{2}\right) \cdot ¢(M_{\text{opt}}) - ¢(M) = \left(1+\frac{\varepsilon}{2}\right) \cdot ¢(M_{\text{opt}}).$$

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 $(1+\frac{\varepsilon}{2}) \cdot ¢(M_{\text{opt}})$ during the i-th iteration. This completes the proof of the lemma. □

Lemma 2.3. *The following to statements hold:*

i. Let M by the matching at the beginning of an iteration and Π an augmenting path in $\vec{G}_M$ with $\alpha_{\underline{\varepsilon}}(\Pi) \le \alpha^*$. Then, $\|\Pi\| \le \frac{27\sqrt{d}n}{\varepsilon}$.

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

Proof: Consider the matching M at the beginning of an iteration, and an augmenting path Π with $\alpha_{\underline{\varepsilon}}(\Pi) \le \alpha^*$. By applying Lemma 2.2, (P3), and $\varepsilon < 1$,

$$\begin{aligned} \|\Pi\| = ¢(\Pi) &\le ¢(M) + ¢(M \oplus \Pi) = ¢(M) + \bar{¢}(\Pi) + ¢(M) \\ &\le ¢(M) + \left(1+\frac{\varepsilon}{2}\right) \cdot ¢(M_{\text{opt}}) - ¢(M) + ¢(M) \\ &\le (2+\varepsilon) \cdot ¢(M_{\text{opt}}) \le \frac{27\sqrt{d}n}{\varepsilon}. \end{aligned}$$

If M is an intermediate matching and Γ is a reducing cycle, the same bound holds without applying the second statement of Lemma 2.2. Instead, we can argue that $¢(M \oplus \Pi) < ¢(M)$ since Π is reducing:

$$\|\Gamma\| = ¢(\Gamma) \le ¢(M) + ¢(M \oplus \Gamma) < 2 \cdot ¢(M) \le 2\left(1+\frac{\varepsilon}{2}\right) \cdot ¢(M_{\text{opt}}) \le (2+\varepsilon) \cdot ¢(M_{\text{opt}}) \le \frac{27\sqrt{d}n}{\varepsilon}. \quad \square$$

We now analyze the running time of the algorithm. Let $\Pi_1, \ldots, \Pi_n$ be the sequence of augmenting paths computed by the algorithm, let $\mathcal{N}_i$ be the set of alternating cycles that were cancelled 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 3.2, the total time spent by the algorithm is $(|\Pi| + \sum_{\Gamma \in \mathcal{N}} |\Gamma|) \cdot (\varepsilon^{-1} \log n)^{O(d)}$, 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 $\alpha_{\bar{\varepsilon},i}$ to denote $\alpha_{\bar{\varepsilon},M_i}$—the $\bar{\varepsilon}$-adjusted cost after the i-th iteration where M_i is a partial matching with i edges—and set $\alpha^*_i := \alpha^*_{M_i}$. We begin by bounding α^*_i.

Lemma 2.4. $\alpha^*_i \le O\left(\frac{n}{\varepsilon(n-i)}\right)$. *As a corollary,* $\sum_{i=0}^{n-1} \alpha^*_i = O\left(\frac{n \log n}{\varepsilon}\right)$.

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

$$\begin{aligned}
(n-i)\cdot \alpha^*_i &\le \sum_{j=1}^{n-i} \alpha_{\bar{\varepsilon},i}(P_j) \le \sum_{j=1}^{n-i}\left(\bar{¢}_{M_i}(P_j) + c_0\bar{\varepsilon}\cdot \|P_j\|\right)\\
&\le \sum_{j=1}^{n-i}\left(¢(P_j\cap M_{\text{opt}}) - ¢(P_j \cap M_i) + \frac{c_0\varepsilon}{c_1}\left(¢(P_j\cap M_{\text{opt}}) + ¢(P_j\cap M_i)\right)\right)\\
&\le ¢(M_{\text{opt}}) + \frac{\varepsilon}{8}\left(¢(M_{\text{opt}}) + ¢(M_i)\right)\\
&\le \frac{9\sqrt{d}n}{\varepsilon} + \frac{\varepsilon}{8}\left(\frac{9\sqrt{d}n}{\varepsilon} + \frac{18\sqrt{d}n}{\varepsilon}\right) = O\left(\frac{n}{\varepsilon}\right),
\end{aligned}$$

where the last inequality follows from property (P3) of the input and combining Lemma 2.2.i with (P3) and $\varepsilon < 1$. The corollary follows from a harmonic sum. □

Lemma 2.5. $\displaystyle\sum_i \|\Pi_i\| + \sum_{\Gamma\in\mathcal{N}} \|\Gamma\| = O\left(\frac{n\log^2 n}{\varepsilon^2}\right).$

Proof: Recall that FIND-PATH guarantees

$$\alpha_{\underline{\varepsilon},i}(\Pi_{i+1}) \le \alpha^*_i,$$

which implies that $\sum_{i=1}^n \alpha_{\underline{\varepsilon},i-1}(\Pi_i) = O(n\log n/\varepsilon)$ by Lemma 2.4. Since $\alpha_{\underline{\varepsilon}}(\Gamma) < 0$ for all cycles $\Gamma \in \mathcal{N}$, we have

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

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

$$\sum_{i=1}^n \alpha_{\underline{\varepsilon},i-1}(\Pi_i) + \sum_{\Gamma\in\mathcal{N}} \alpha_{\underline{\varepsilon}}(\Gamma) = \sum_{i=1}^n \bar{¢}(\Pi_i) + \sum_{\Gamma\in\mathcal{N}} \bar{¢}(\Gamma) + c_0\underline{\varepsilon}\left(\sum_{i=1}^n \|\Pi_i\| + \sum_{\Gamma\in\mathcal{N}}\|\Gamma\|\right). \tag{2}$$

Observe that the augmentation of all paths and cycles results in the final matching M_{alg}. Therefore, the net-cost terms in Equation 2 sum to $¢(M_{\text{alg}}) \ge 0$, and we obtain

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

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

Putting everything together, we obtain Theorem 1.1.

Remark. As mentioned above, the second term in the adjusted cost acts as a regularizer. The Hungarian algorithm computes the minimum net-cost augmenting path at each step, which ensures that the residual graph has no negative cycles, but the total number of edges in augmenting paths can be as bad as $\Omega(n^2)$. By adding the regularizer term, we ensure that the edge-length of the augmenting paths to be $O((n/\varepsilon^2)\log^2 n)$. The residual graph may now contain negative cycles, but we ensure that the net-cost of these cycles is not too negative by enforcing that the $\bar{\varepsilon}$-adjusted cost of a cycle to be non-negative (see the cycle invariant CI) through select reducing cycle cancellations.

3 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 $\mathbb{R}^d$ 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, described more in detail in Sections 3.4 and 4, 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 $\alpha_{\underline{\varepsilon}}(\Pi)$ is at most the weight of π in $G_\boxplus$; we refer to this map as an *expansion* of π in $\vec{G}_M$. Π can be reported in time proportional to $|\Pi|$. The data structure also maintains a priority queue $\mathcal{Q}$ that stores the cheapest "augmenting path" of each $G_\boxplus$.

FINDPATH involves retrieving the overall cheapest augmenting path π^* from $\mathcal{Q}$ and using the appropriate $\mathcal{D}_\boxplus$ to expand π^* to an augmenting path Π^* and returning Π^*. AUGMENT(M, Π^*) involves augmenting $M \leftarrow M \oplus \Pi^*$ and updating the data structures for all cells that contain at least one point of Π^*. During the update if a reducing cycle Γ is encountered, Γ is cancelled by setting $M \leftarrow M \oplus \Gamma$ and again updating the data structure. This process is repeated until all cells have been updated and no reducing cycles have been detected. By constructing $G_\boxplus$ carefully, we prove the cycle invariant CI and the correctness of FINDPATH and AUGMENT procedures. We now describe each of the components in detail. We note that even though the data structure depends on the current matching, for simplicity, we omit the dependency on M from our notation.

3.1 Hierarchical covering

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* $\mathfrak{G}_1, \mathfrak{G}_2, \dots, \mathfrak{G}_{\log\Delta}$. 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. In notation,

$$\mathfrak{G}_i := [0, \ell_i]^d + (\ell_i \mathbb{Z})^d.$$

For each i, we build a collection of *cells* $\mathfrak{C}_i$ as follows: For any hypercube $\boxplus \in \mathfrak{G}_i$, we add 2^d cells into $\mathfrak{C}_i$, namely, $\boxplus + \ell_{i-1}(b_1, \dots, b_d)$ for all d-bits $b_1, \dots, b_d \in \{0,1\}$, each corresponds to a shifting of $\boxplus$ by either $+0$ or $+\ell_{i-1}$ in each dimension. Alternatively, $\mathfrak{C}_i$ is the union of 2^d different translations $\mathfrak{G}_i + \ell_{i-1}(b_1, \dots, b_d)$ of $\mathfrak{G}_i$. We refer to each hypercube $\boxplus$ as a *cell*.

Every cell in $\mathfrak{C}_i$ has its boundary lying completely on grid boundaries of $\mathfrak{G}_{i-1}, \mathfrak{G}_{i-2}, \dots, \mathfrak{G}_0$. As a consequence, cells are always perfectly tiled by lower-level grids, and grids $\mathfrak{C}_j$ are a refinement of $\mathfrak{C}_i$ for all $j < i$. The *children* of a cell $\boxplus \in \mathfrak{C}_i$ are the 3^d cells of $\mathfrak{C}_{i-1}$ contained in $\boxplus$. For any cell $\boxplus' \in \mathfrak{C}_i$ such that $\boxplus \cap \boxplus' \neq \emptyset$, we say $\boxplus'$ is a *sibling* of $\boxplus$. (For convenience $\boxplus$ is a sibling of itself.) Let $\text{Ch}(\boxplus)$ and $\text{Sb}(\boxplus)$ denote the sets of children and siblings of $\boxplus$, respectively. If $\boxplus'$ is a child of $\boxplus$ then we call $\boxplus$ a *parent* of $\boxplus'$. We can now define descendants/ancestors of a cell in standard manner.

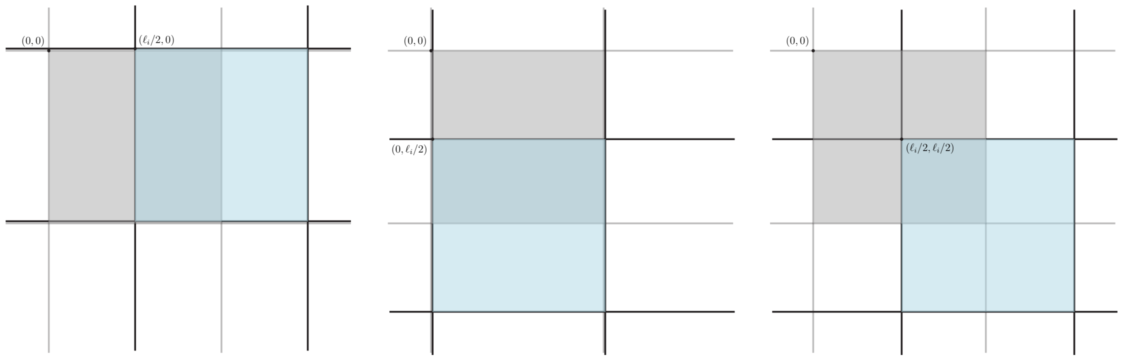


Figure 3.1. The 2^d-1 additional shifts of the gray-shaded hypercube from $\mathfrak{G}_i$. There are a total of 2^d shifts, and 2^d cells originating from this hypercube.

3.2 Compressed graph

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 whose values will be specified later (See Lemma 3.3 for δ and Lemma 3.8 for h). 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 $i > \left\lceil \log\frac{\delta}{4\sqrt{d}}\right\rceil$ (we define grids for $i \geq 0$ but they can also be defined for $i < 0$ in a straightforward manner). For each $i \in \{0,\dots,h\}$, let the set of non-empty level-i cells be

$$\mathscr{C}_i := \left\{\boxplus \in \mathfrak{C}_i \mid \boxplus \cap (A\cup B) \neq \varnothing\right\}$$

and $\mathscr{C} = \bigcup_{i=1}^{h}\mathscr{C}_i$. For each cell $\boxplus \in \mathscr{C}$, we construct a weighted directed graph $G_\boxplus = (V_\boxplus, E_\boxplus)$ called the *compressed graph*, as follows.

Clustering and nodes of $G_\boxplus$. Suppose $\boxplus \in \mathscr{C}_i$, i.e., the level of $\boxplus$ is i. Set the *level-offset* $\tau := \lceil \log(4\sqrt{d}/\delta)\rceil$. We partition $\boxplus$ into *subcells* by the hypercubes of $\mathfrak{G}_{i-\tau}$. By construction, $\boxplus$ is aligned with the grid boundaries in $\mathfrak{G}_{i-\tau}$. (When $i-\tau<0$ the grid $\mathfrak{G}_j$ for $j<0$ is still well-defined.) Note that we use the same grid to define the subcells of all cells at level i. The value of τ is chosen so that the side-length of a subcell of $\boxplus$ is at most $\frac{\delta}{4\sqrt{d}}\ell_i$ and its diameter is at most $\frac{\delta}{4}\ell_i$. For a subcell $\square$, its *children subcells* $\mathrm{Ch}(\square)$ are the hypercubes of $\mathfrak{G}_{i-\tau-1}$ that lie in $\square$. Let $X_\boxplus$ denote the set of subcells of $\boxplus$; we have $|X_\boxplus| = (\varepsilon^{-1}\log n)^{O(d)}$. Recall that $\boxplus$ has 3^d children cells; a subcell $\square$ of $\boxplus$ lies in 2^d of the children of $\boxplus$.

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

$$\mathscr{A}_\boxplus := \left\{A_\square \mid \square\in X_\boxplus \text{ and } A_\square \neq\varnothing\right\} \quad\text{and}\quad \mathscr{B}_\boxplus := \left\{B_\square \mid \square\in X_\boxplus \text{ and } B_\square \neq\varnothing\right\}.$$

$\mathscr{A}_\boxplus$ (resp. $\mathscr{B}_\boxplus$) induces a clustering of $A\cap\boxplus$ (resp. $B\cap\boxplus$).

The nodes of $G_\boxplus$ are the A- and B-clusters of $\boxplus$, i.e., $V_\boxplus := \mathscr{A}_\boxplus \cup \mathscr{B}_\boxplus$. Observe that $|V_\boxplus| = (\varepsilon^{-1}\log n)^{O(d)}$. We note that each cell $\boxplus\in\mathscr{C}_0$ either contains points of A or points of B, so either $\mathscr{A}_\boxplus=\varnothing$ or $\mathscr{B}_\boxplus=\varnothing$. For level $i\geq 1$, let $\square$ be a subcell of $\boxplus$. Then

$$A_\square = \bigcup_{\Delta\in\mathrm{Ch}(\square)} A_\Delta \quad\text{and}\quad B_\square = \bigcup_{\Delta\in\mathrm{Ch}(\square)} B_\Delta.$$

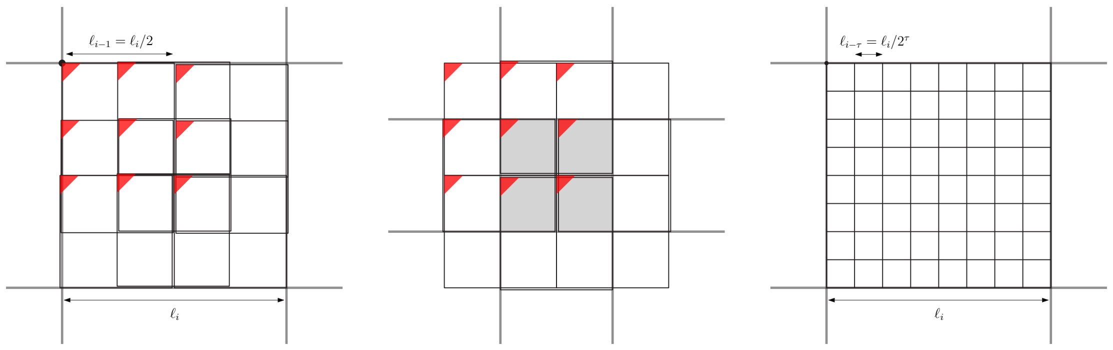


Figure 3.2. **Left**: The 3^d children of a cell. Top-left corner of each child is marked in red. Drawn slightly askew for visibility. **Center**: The 3^d siblings of the shaded cell. Top-left corner of each sibling is marked in red. Note that the cell is a sibling of itself. Also drawn slightly askew. **Right**: The subcells of a cell.

Hence, a cluster of $\boxplus$ is obtained by merging the at most 2^d clusters of children of $\boxplus$ that contain $\square$. In other words, a node of $G_\boxplus$ is obtained by compressing (up to) 2^d nodes from the compressed graphs of children of $\boxplus$.

Arcs and arc weights of $G_\boxplus$. We regard $G_\boxplus$ as a compressed graph of the subgraph of $\vec{G}_M$ induced by $(A \cup B) \cap \boxplus$ in which, for each A- or B-cluster, all cluster points are compressed into a single node. There are two types of arcs in *$E_\boxplus$*, the edge set of $G_\boxplus$:

- **Matching arcs**: For a pair of subcells $\square$ and $\square'$ in $X_\boxplus$, if M contains an edge $(b,a) \in B_\square \times A_{\square'}$ then we add an arc $(B_\square, A_{\square'})$ to $E_\boxplus$. For each pair of subcells $\square, \square'$ of $\boxplus$ such that any child cell of $\boxplus$ contains at most one of them, we store the matching edges from $B_\square$ to $A_{\square'}$, denoted as *$M_{\square,\square'}$* $:= M \cap (B_\square \times A_{\square'})$, in a priority queue so that we can update $M_{\square,\square'}$ and retrieve the longest edge in $M_{\square,\square'}$ in $O(\log n)$ time. We note that each edge of M appears in exactly one $M_{\square,\square'}$, so the total size of $\sum |M_{\square,\square'}|$, summed over all subcell pairs, is equal to $|M|$, which is at most n.

- **Non-matching arcs**: For a pair of subcells $\square, \square' \in X_\boxplus$ such that $A_\square \neq \varnothing, B_{\square'} \neq \varnothing$, we add the arc $(A_\square, B_{\square'})$ to $E_\boxplus$. If there is no child cell of $\boxplus$ that contains both $\square$ and $\square'$, we refer to $(A_\square, B_{\square'})$ as a *bridge* arc, otherwise it is an *internal* arc.

We note that if $\boxplus \in \mathscr{C}_0$, then $E_\boxplus = \varnothing$.

We now define a weight function $w_\boxplus$ on the arcs $E_\boxplus$ as described below. Using this weight function, we compute a minimum-weight path *$\pi_\boxplus(X,Y)$* between all pairs of nodes X, Y in $G_\boxplus$. For every pair $A_\square \in \mathscr{A}_\boxplus, B_{\square'} \in \mathscr{B}_\boxplus$, we also compute the *expansion* of $\pi_\boxplus(X,Y)$ to a simple (possibly empty) alternating path *$\Phi_\boxplus(A_\square, B_{\square'})$* in $\vec{G}_M$ whose first and last edges are matching edges, as described below. Let b_0 (resp. a_0) be the first (resp. last) vertex of $\Phi_\boxplus(A_\square, B_{\square'})$. For a given pair $(a,b) \in A_\square \times B_{\square'}$, we construct an alternating path from a to b by adding the (non-matching) edges (a, b_0) and (a_0, b) at each end; if $\Phi_\boxplus(A_\square, B_{\square'}) = \varnothing$ then the path is simply (a,b). We denote this path as $(a, \Phi_\boxplus(A_\square, B_{\square'}), b)$ and refer to $\Phi_\boxplus(A_\square, B_{\square'})$ as the *flex-tip* expansion of $\pi_\boxplus(A_\square, B_{\square'})$. The construction guarantees the following condition, which we refer to as *expansion inequality*:

$$\alpha_{\underline{\varepsilon}}\Big((a, \Phi_\boxplus(A_\square, B_{\square'}), b)\Big) \le w_\boxplus(\pi_\boxplus(A_\square, B_{\square'})). \tag{3}$$

$\Phi_\boxplus(A_\square, B_{\square'})$ is stored implicitly using a data structure and can be retrieved in time proportional to its size.

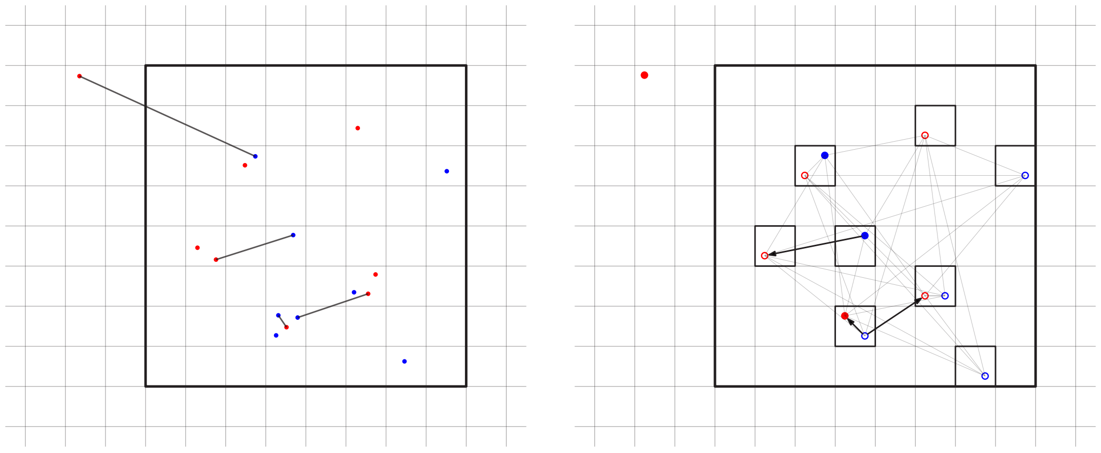


Figure 3.3. **Left**: The points and matching edges intersecting a cell. The grid shows the subcells of the level. **Right**: The clusters in each subcell and the compressed graph built in the cell. Each subcell has at most 2 clusters, one for each of A and B. Saturated clusters are solid, unsaturated clusters are hollow. Matching arcs are in bold, non-matching in gray. For visibility, the arrowheads of non-matching arcs are not drawn.

While computing minimum weight paths $\pi_{\boxplus}(\cdot,\cdot)$ or their expansions $\Phi_{\boxplus}(\cdot,\cdot)$, we may discover a reducing cycle Γ in $\vec{G}_M$ (e.g., if the compressed graph $G_{\boxplus}$ contains a negative cycle) in which case we update M by cancelling Γ and rebuilding the data structure $\mathscr{D}_{\boxplus}$ and its descendants. We say cell $\boxplus$ is *stable* if (1) the children of $\boxplus$ are stable, and (2) no reducing cycle was detected while computing $\pi_{\boxplus}(\cdot,\cdot)$ or $\Phi_{\boxplus}(\cdot,\cdot)$ for all pairs of nodes in $\mathscr{A}_{\boxplus} \times \mathscr{B}_{\boxplus}$. Assuming all children cells of $\boxplus$ are stable, then we define *arc weights $w_{\boxplus}$* $: E_{\boxplus} \to \mathbb{R}$ recursively as follows:

- **Matching arcs**: For a matching arc $(B_{\square}, A_{\square'}) \in E_{\boxplus}$, we define $w_{\boxplus}(B_{\square}, A_{\square'})$ to be

$$w_{\boxplus}(B_{\square}, A_{\square'}) := -\left(\max_{(b,a)\in M_{\square,\square'}} \|b-a\|\right) + \delta\ell_i; \tag{4}$$

  the maximum is taken over all matching edges between $B_{\square}$ and $A_{\square'}$. If there is a child cell $\boxplus'$ of $\boxplus$ that contains both $\square$ and $\square'$, then $w_{\boxplus}(B_{\square}, A_{\square'})$ can be defined recursively as follows: Observe that

$$\bar{\cancel{c}}(B_{\square}, A_{\square'}) = \min_{(b,a)\in M_{\square,\square'}} \bar{\cancel{c}}(b,a) = \min_{\substack{\Delta\in \mathrm{Ch}(\square)\\ \Delta'\in \mathrm{Ch}(\square')}} \bar{\cancel{c}}(B_{\Delta}, A_{\Delta'}). \tag{5}$$

  Otherwise $\bar{\cancel{c}}(B_{\square}, A_{\square'})$ can be obtained from the priority queue that stores $M_{\square,\square'}$. We set

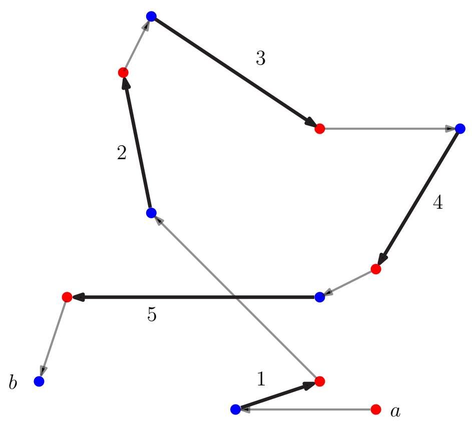
$$w_{\boxplus}(B_{\square}, A_{\square'}) = \bar{\cancel{c}}(B_{\square}, A_{\square'}) + \delta\ell_i. \tag{6}$$

- **Non-matching arcs**: Let $(A_{\square}, B_{\square'}) \in E_{\boxplus}$ be a non-matching arc. If $(A_{\square}, B_{\square'})$ is a bridge arc, i.e., there is no child of $\boxplus$ that contains both $\square$ and $\square'$, then we set

$$w_{\boxplus}(A_{\square}, B_{\square'}) := \|\mathrm{cntr}_{\square} - \mathrm{cntr}_{\square'}\| + \delta\ell_i, \tag{7}$$

  where $\mathrm{cntr}_{\square}$ (resp. $\mathrm{cntr}_{\square'}$) is the center of $\square$ (resp. $\square'$). If $(A_{\square}, B_{\square'})$ is an internal arc, i.e., there is a child that contains both $\square$ and $\square'$, we set $w_{\boxplus}(A_{\square}, B_{\square'})$ to

$$w_{\boxplus}(A_{\square}, B_{\square'}) := \min_{\substack{\Delta\in \mathrm{Ch}(\square), \Delta'\in \mathrm{Ch}(\square')\\ \boxplus'\in \mathrm{Ch}(\boxplus): \Delta,\Delta'\in\boxplus'}} \alpha_{\underline{\varepsilon}}\Big(\big(a_{\Delta}, \Phi_{\boxplus'}(A_{\Delta}, B_{\Delta'}), b_{\Delta'}\big)\Big) + \delta\ell_i \tag{8}$$



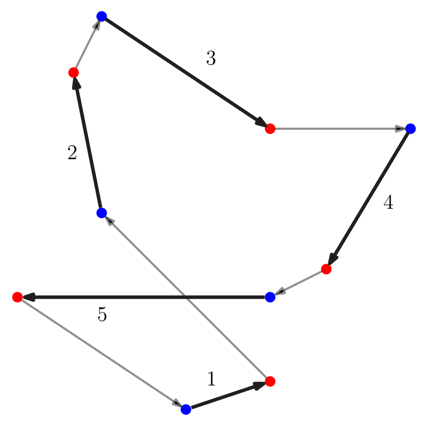


Figure 3.4. Left: Flex-tip representation of an augmenting path, with tips fixed to a and b. Bold edges are matching edges, labels indicate the sequence. **Right**: Flex-tip representation of an alternating cycle on the same sequence of matching edges.

where $(a_\triangle, b_{\triangle'}) \in A_\triangle \times B_{\triangle'}$ is an arbitrary pair that is used to compute the weight.

Given the weight function $w_\boxplus$, if $\boxplus$ is not stable (but all its children are), then let $\Gamma_\boxplus$ be a reducing cycle in $\vec{G}_M$ that acts as a witness of its instability. If $\boxplus$ is stable, the data structure maintains a minimum-weight path $\pi_\boxplus(A_\square, B_{\square'})$ for every pair $A_\square \in \mathscr{A}_\boxplus$ and $B_{\square'} \in \mathscr{B}_\boxplus$. If both $A_\square, B_{\square'}$ are unsaturated, then we call $\pi_\boxplus(A_\square, B_{\square'})$ an *augmenting path* in $G_\boxplus$. Recall that for each pair $A_\square, B_{\square'}$, we also compute an expansion $\Phi_\boxplus(A_\square, B_{\square'})$ of $\pi_\boxplus(A_\square, B_{\square'})$ to an alternating path in $\vec{G}_M$. If $\pi_\boxplus(A_\square, B_{\square'})$ is an augmenting path in $G_\boxplus$, we choose a pair of free points $(a_\square, b_{\square'}) \in A_\square \times B_{\square'}$ and set $\Pi_{\square,\square'} = (a_\square, \Phi_\boxplus(A_\square, B_{\square'}), b_\square)$; $\Pi_{\square,\square'}$ is an augmenting path in $\vec{G}_M$. Let

$$(\square, \square')^*_\boxplus := \arg\min_{\square,\square'} \alpha_{\underline{\varepsilon}}(\Pi_{\square,\square'})$$

where the minimum is taken over all pairs $A_\square, B_{\square'}$ such that both are unsaturated. If $G_\boxplus$ does not have any augmenting path, the pair $(\square, \square')^*_\boxplus$ is undefined. Set *OptPairs* $:= \left\{(\square, \square')^*_\boxplus \mid \boxplus \in \mathscr{C}\right\}$. We store *OptPairs* in a priority queue with $\alpha_{\underline{\varepsilon}}(\Pi_{(\square,\square')^*_\boxplus})$ as the key of $(\square, \square')^*_\boxplus$.

We conclude by remarking that the information stored at $\boxplus$ depends on $M \cap ((A \cap \boxplus) \times (B \cap \boxplus))$. So whenever the matching edges change, we have to update the information stored at the corresponding $\boxplus$. The REPAIR($\boxplus$) procedure, described in Section 3.4, updates the data structure at $\boxplus$. Initially $M = \varnothing$, and the data structure can be built by calling REPAIR at all cells of $\mathscr{C}$ in a bottom-up manner.

3.3 Flex-tip expansion of compressed paths

Let $\langle a_0, b_1, a_1, \dots, a_{k-1}, b_k\rangle$ be a path in $\vec{G}_M$ from $a_0 \in A$ to $b_k \in B$. The path can be specified by the sequence $\langle (b_1, a_1), \dots, (b_{k-1}, a_{k-1})\rangle$ of matching edges plus the endpoints a_0 and b_k. Conversely, since $(a, b) \in \vec{G}_M$ if $(b, a) \notin M$ for any pair $(a, b) \in A \times B$, any sequence $\Phi = \langle e_1, e_2, \dots, e_k\rangle$ of matching edges $\langle a, b_1, a_1, \dots, b_k, a_k, b\rangle$ where $e_i = (b_i, a_i)$. The sequence Φ also defines a unique cycle $\langle b_1, a_1, \dots, b_k, a_k, b_1\rangle$. In our application, we will keep the tips flexible and choose them from some fixed subsets of A and B.

For a path $\pi_\boxplus(A_\square, B_{\square'})$, $\Phi_\boxplus(A_\square, B_{\square'})$ is a *flex-tip expansion (FTE)* to a simple path in $\vec{G}_M$, represented as a sequence of matching edges whose both endpoints lie in $\boxplus$ as described above. For any pair $(a, b) \in A_\square \times B_{\square'}$, $(a, \Phi_\boxplus(A_\square, B_{\square'}), b)$ gives a simple path in $\vec{G}_M$ from $a \in A_\square$ to $b \in B_{\square'}$, and the expansion inequality (3) implies that $\alpha_{\underline{\varepsilon}}\big((a, \Phi_\boxplus(A_\square, B_{\square'}), b)\big) \le w_\boxplus(\pi_\boxplus(A_\square, B_{\square'}))$. Recall that for an internal arc $\gamma \in G_\boxplus$, $w_\boxplus(\gamma)$ is $\alpha_{\underline{\varepsilon}}((a_\triangle, \Phi_{\boxplus'}(A_\triangle, B_{\triangle'}), b_{\triangle'}))$ for some child cell $\boxplus'$ of $\boxplus$ and children clusters $A_\triangle$, $B_{\triangle'}$ of $A_{\boxplus'}$, $B_{\boxplus'}$, respectively. We define $\Phi(\gamma)$ — FTE of γ — to be $\Phi_{\boxplus'}(A_\triangle, B_{\triangle'})$, again represented as a sequence of matching edges. We refer to a subpath defined by a contiguous subsequence of matching edges in $\Phi(\gamma)$ as a *pathlet* of γ, which can be represented by specifying its first and last matching edges. (See Section 4.1.)

It would be too expensive to maintain flex-tip expansions explicitly for every pair $\square, \square' \in X_{\boxplus}$. Instead, we maintain them implicitly. Roughly speaking, $\Phi_{\boxplus}(A_{\square}, B_{\square'})$ is implicitly represented as a sequence $\varphi_1 \circ \varphi_2 \circ \cdots \circ \varphi_t$, where each φ_i is either a matching edge lying in $\boxplus$ or a pathlet of an internal arc γ_i of $G_{\boxplus}$. The description complexity of this implicit representation is $O(|V_{\boxplus}|)$. It is computed and updated using a data structure described in Section 4, that supports the following four high-level operations.

- ConstructFTE(π). Given a compressed path π of $G_{\boxplus}$, return an implicit representation of its FTE that satisfies the expansion inequality (3). This operation may abort, and instead return an implicit representation of a simple reducing cycle.

- SimpleReducingCycle(Γ). Given a negative cycle Γ of $G_{\boxplus}$, return an implicit representation of a simple reducing cycle. (The existence of such a cycle follows from Lemma 3.3 below.)

- Report(Φ). Given an implicit representation of an FTE Φ, return the actual path. The procedure takes $|\Phi| \cdot (\varepsilon^{-1} \log n)^{O(d)}$ time, where $|\Phi|$ is the output size.

- Cost(Φ). Given an implicit representation of an FTE Φ, return $\alpha_{\underline{\varepsilon}}(\Phi)$ and the first and the last endpoint of Φ.

Here, we give a brief, high-level description of the procedure. Suppose we have computed FTEs for all children cells $\boxplus'$ of $\boxplus$. Then we construct the flex-tip expansions $\Phi_{\boxplus}(A_{\square}, B_{\square'})$ as follows: Let $\pi_{\boxplus}(A_{\square}, B_{\square'}) = \langle A_{\square} = A_{\square_0}, B_{\square_1}, A_{\square_2}, \ldots, A_{\square_k}, B_{\square_{k+1}} = B_{\square'} \rangle$; $k \le |V_{\boxplus}| = (\varepsilon^{-1} \log n)^{O(d)}$. For each $1 \le j \le k$, $(B_{\square_j}, A_{\square_{j+1}})$ is a matching arc of $G_{\boxplus}$, so let g_j be the longest matching edge between $B_{\square_j}$ and $A_{\square_{j+1}}$. For $0 \le j \le k$, let $\eta_j := (A_{\square_j}, B_{\square_{j+1}})$ be the non-matching arc of $G_{\boxplus}$. Set

$$\tilde{\Pi} := \Phi(\eta_0) \circ g_1 \circ \Phi(\eta_2) \circ \cdots \circ g_k \circ \Phi(\eta_k). \tag{9}$$

If η_j is a bridge arc, then $\Phi(\eta_j) = \varnothing$, and if η_j is an internal arc then $\Phi(\eta_j)$ is the flex-tip expansion of η_j. We refer to $\tilde{\Pi}$ as the *intermediate expansion* of $\pi_{\boxplus}(A_{\square}, B_{\square'})$. Each $\Phi(\eta_j)$ is a simple path in $\vec{G}_M$ that lies inside $\boxplus$. It is tempting to set $\Phi_{\boxplus}(A_{\square}, B_{\square'})$ to $\tilde{\Pi}$ but $\tilde{\Pi}$ may not be simple as $\Phi(\eta_j)$ and $\Phi(\eta_{j'})$ for $j' \ne j$ may share a matching edge, so we repeatedly splice cycles out of $\tilde{\Pi}$. If the procedure finds a reducing cycle Γ at any stage, it returns (an implicit representation of) a simple reducing subcycle $\hat{\Gamma} \subseteq \Gamma$. The procedure either returns an (implicit) simple reducing cycle $\hat{\Gamma} \subseteq \tilde{\Pi}$, or a simple subpath $\hat{\Pi} \subseteq \tilde{\Pi}$, where $\alpha_{\underline{\varepsilon}}(\hat{\Pi}) \le \alpha_{\underline{\varepsilon}}(\tilde{\Pi})$. Actually, it returns an implicit representation of $\hat{\Pi}$ or $\hat{\Gamma}$ as described above. The actual path/cycle is retrieved using the Report procedure.

Using the Lemmas in Section 4, we conclude the following:

Lemma 3.1. *After having computed minimum weight paths for all pairs $(A_{\square}, B_{\square'})$ between subcells $\square, \square'$ of $\boxplus$, implicit representations of their flex-tip expansions can be computed in $(\varepsilon^{-1} \log n)^{O(d)}$ time, provided no reducing cycle is detected.*

3.4 FindPath, Augment, and Repair procedures

We describe the operations FindPath, Augment, and Repair performed on the data structure $\mathscr{D}_{\boxplus}$ built on each cell $\boxplus$. The first two procedures have been defined in Section 2. Repair($\boxplus$) builds the data structure at $\boxplus$, and is a subroutine for Augment. During the process, if it detects a reducing cycle Γ, it returns a simple reducing cycle $\hat{\Gamma} \subseteq \Gamma$, and otherwise *null*. We now describe each procedure in detail. For an alternating path Π in $\vec{G}_M$, we call a cell $\boxplus \in \mathscr{C}$ *affected* by Π if $\boxplus$ contains a vertex of Π (that is, a point in $\mathbb{R}^d$). Let $\mathscr{C}_{\Pi} \subseteq \mathscr{C}$ denote the set of cells affected by Π.

FINDPATH(): It performs a DELETEMIN operation on the priority queue storing the candidate subcell pairs *OptPairs* and retrieves a pair $(\square, \square')^*_{\boxplus}$, such that $\Pi_{(\square,\square')^*_{\boxplus}} = (a_\square, \Phi_{\boxplus}(A_\square, B_{\square'}), b_{\square'})$ is an augmenting path, i.e., $a_\square, b_{\square'}$ are free vertices. Given an implicit representation φ of $\Phi_{\boxplus}(A_\square, B_{\square'})$, we call REPORT($\varphi$) to retrieve $\Phi_{\boxplus}(A_\square, B_{\square'})$ and return Π. The total time spent is $|\Pi| \cdot (\varepsilon^{-1}\log n)^{O(d)}$.

AUGMENT(M, Π): It first sets $M \leftarrow M \oplus \Pi$. We store $\mathscr{C}_\Pi$, the set of cells affected by Π, in a priority queue $\mathscr{Q}_\Pi$ with the level of the cell as its key. At each step, we retrieve a cell $\boxplus$ from $\mathscr{Q}_\Pi$ of the lowest level and update the data structure $\mathscr{D}_{\boxplus}$ by calling REPAIR($\boxplus$). If the call to REPAIR returns a reducing simple cycle Γ, we set $M = M \oplus \Gamma$ and insert all cells in $\mathscr{C}_\Gamma$ into $\mathscr{Q}_\Pi$ and continue. This process continues until $\mathscr{Q}_\Pi$ becomes empty. The procedure then returns the matching M. Figure 3.5 gives the pseudo-code of AUGMENT.

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 $\left(\sum_i |\Gamma_i| + |\Pi|\right) \cdot (\varepsilon^{-1}\log n)^{O(d)}$ time.

```
AUGMENT($M, \Pi$):
  $\Xi \leftarrow \emptyset$
  $M \leftarrow M \oplus \Pi$
  for all $\boxplus \in \mathscr{C}_\Pi$:
      INSERT($\boxplus, \Xi$)
  while $\Xi \neq \emptyset$:
      $\boxplus \leftarrow$ DELETEMIN($\Xi$)
      $\Gamma \leftarrow$ REPAIR($\boxplus$)
      if $\Gamma \neq null$:
          $M \leftarrow M \oplus \Gamma$
          for all $\boxplus' \in \mathscr{C}_\Gamma$:
              INSERT($\boxplus', \Xi$)
  return $M$
```

Figure 3.5. Implementation of AUGMENT using REPAIR.

REPAIR($\boxplus$): Recall that 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($\boxplus$). REPAIR at $\boxplus$ 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. We mark $A_\square$ (resp. $B_\square$) saturated when it does not have any free vertex; once saturated, it remains saturated for the rest of the algorithm.

ii. (**Assigning arc weights**) Let $\square, \square'$ be a pair of subcells of $\boxplus$.

- If no child cell of $\boxplus$ contains both $\square$ and $\square'$, then we update $M_{\square,\square'}$, i.e., delete the edges that are no longer in M and insert newly created edges of M whose endpoints lie in $\square$ and $\square'$. (Recall that $\square$ and $\square'$ appear in multiple cells, but we update $M_{\square,\square'}$ for the one associated with $\boxplus$.) If $M_{\square,\square'} \neq \emptyset$, we add $(B_\square, A_{\square'})$ to $E_{\boxplus}$ and set $w_{\boxplus}(B_\square, A_{\square'})$ using (4). Since $M_{\square,\square'}$ is stored in priority queue $\mathscr{Q}_{\square,\square'}$, the cost of the longest edge can be computed in $O(1)$ time.
- If some child cell contains both $\square$ and $\square'$, we use the recursive expression in (6), since all children are stable, to determine whether $(B_\square, A_{\square'}) \in E_{\boxplus}$ and to compute $w_{\boxplus}(B_\square, A_{\square'})$. Finally, we add the non-matching arc $(A_\square, B_{\square'})$ to $E_{\boxplus}$ and set $w_{\boxplus}(A_\square, B_{\square'})$ using (7) and (8). We call the COST procedure to compute $\alpha_{\bar{\varepsilon}}\left((a_\triangle, \Phi_{\boxplus'}(A_\triangle, B_{\triangle'}), b_\triangle)\right)$, which we need in (8).

iii. (**APSP computation**) We compute the minimum-weight 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 FTEs, as described above, which returns the implicit representation of a simple reducing cycle Γ in $\vec{G}_M$. In this case, we abort the update of $\boxplus$ and return REPORT(Γ), which retrieves the actual simple cycle $\hat{\Gamma}$. Otherwise this step terminates after computing minimum-weight paths between all pairs of nodes in $G_{\boxplus}$.

iv. (**Computing expansions**) For every pair of subcells $\square, \square'$, we compute the implicit representation of the FTEs $\Phi_{\boxplus}(A_{\square}, B_{\square'})$ of $\pi_{\boxplus}(A_{\square}, B_{\square'})$ by calling CONSTRUCTFTE($\pi_{\boxplus}(A_{\square}, B_{\square'})$). The procedure may abort and return a simple reducing cycle Γ, in which case we abort the update of $\boxplus$ and return Γ. If we succeed in computing expansions for all pairs, we mark $\boxplus$ as stable.

v. (**Augmenting paths**) For each pair of subcells $\square, \square'$ of $\boxplus$ such that both $A_{\square}$ and $B_{\square'}$ are unsaturated, we choose a pair of free vertices $a \in A_{\square}$ and $b \in B_{\square'}$ and set $\Pi_{\square,\square'} = (a, \Phi_{\boxplus}(\square, \square'), b)$. Otherwise, $\Pi_{\square,\square'}$ is undefined and $\alpha_{\underline{\varepsilon}}(\Pi_{\square,\square'}) = \infty$. We compute $(\square, \square')^*_{\boxplus} = \arg\min_{\square,\square'} \alpha_{\underline{\varepsilon}}(\Pi_{\square,\square'})$. If *OptPairs* already contains a subcell pair for $\boxplus$, we replace it with the new $(\square, \square')^*_{\boxplus}$; otherwise we insert $(\square, \square')^*_{\boxplus}$ into *OptPairs*.

vi. (**Auxiliary data structure**) Finally, we build the data structure, which is used to compute the FTEs at ancestor cells of $\boxplus$, and return *null*.

The time spent in Step (i) and in computing $M_{\square,\square'}$ in Step (ii) is charged to changes in the current matching. Each matching edge that is deleted or inserted is charged $O(\log n)$ times — $O(1)$ times per level. If Step (iii) or (iv) calls REPORT($\hat{\Gamma}$), it takes $|\hat{\Gamma}| \cdot O(\log n)$ time, which we charge to the cycle returned. Hence, the total charge to the matching edges and reducing cycles over all calls of REPAIR during a single execution of AUGMENT is $\left(\sum_i |\Gamma_i| + |\Pi|\right) \cdot O(\log n)$, which we charge to AUGMENT. The rest of the REPAIR procedure takes $(\varepsilon^{-1} \log n)^{O(d)}$ time. Putting everything together, we obtain the following:

Lemma 3.2. 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* $\left(|\Pi| + \sum_i |\Gamma_i|\right) \cdot (\varepsilon^{-1} \log n)^{O(d)}$ *time, where* $\{\Gamma_i\}$ *are the reducing cycles canceled by the procedure.*

3.5 Proof of correctness

We prove the correctness of FINDPATH and AUGMENT and the invariant CI. There are two main lemmas in this section, Lemma 3.3 and Lemma 3.7, that relate the net costs of residual paths and cycles to the weights of compressed paths. We first present Lemma 3.3, then two technical lemmas used for proving Lemma 3.7, and finally Lemma 3.7 itself. Lemma 3.3 bounds the $\underline{\varepsilon}$-adjusted cost of the expansion of a minimum-weight path in a compressed graph. Lemma 3.7 proves a relationship between the $\bar{\varepsilon}$-adjusted cost of a path Π in $\vec{G}_M$ and the corresponding "compressed" path in the lowest cell $\boxplus$ that contains Π. We use them together to prove the correctness of FINDPATH (Lemma 3.9) and to prove the invariant invariant CI (Corollary 3.11).

Lemma 3.3 (Expansion Inequality). *Let* $\boxplus$ *be a cell, let* $\square, \square'$ *be two subcells in* $\boxplus$*, and let* $\Phi_{\boxplus}(A_{\square}, B_{\square'})$ *be the flex-tip expansion of* $\pi_{\boxplus}(A_{\square}, B_{\square'})$*. For any pair* $(a, b) \in A_{\square} \times B_{\square'}$*, let* $\Pi_{ab} = (a, \Phi_{\boxplus}(A_{\square}, B_{\square'}), b)$ *be the alternating path in* $\vec{G}_M$ *from* a *to* b *induced by* $\Phi_{\boxplus}(A_{\square}, B_{\square'})$*. Then,*

$$\alpha_{\underline{\varepsilon}}(\Pi_{ab}) \leq w_{\boxplus}(\pi_{\boxplus}(A_{\square}, B_{\square'})) \qquad \textit{for all } (a, b) \in A_{\square} \times B_{\square'}.$$

Proof: Let $\pi_{\boxplus}(A_\square, B_{\square'}) = \left\langle A_\square = A_{\square_0}, B_{\square_1}, A_{\square_2}, \ldots, A_{\square_k}, B_{\square_{k+1}} = B_{\square'} \right\rangle$, and let $\Pi = \Phi(\eta_0) \circ g_1 \circ \Phi(\eta_2) \circ \cdots \circ g_{k-1} \circ \Phi(\eta_k)$ be its (possibly self-intersecting) flex-tip representation, as from Section 3.3; recall that Π may be a self-intersecting residual path. Recall the notation: $\eta_j = (A_{\square_j}, B_{\square_{j+1}})$, $g_j = (b_j, a_{j+1})$ is the longest matching edge in $M \cap (B_{\square_j} \times A_{\square_{j+1}})$, and $\Phi(\eta_j) = \varnothing$ if η_j is a bridge arc and $\Phi(\eta_j)$ is the flex-tip expansion of η_j if it is internal (with tips fixed to a_j, b_{j+1}).

Let $\tilde{\Pi}_{ab} = (a, \Pi, b)$, which is a possibly self-intersecting path in $\vec{G}_M$. By construction, $\alpha_{\underline{\varepsilon}}(\Pi_{ab}) \le \alpha_{\underline{\varepsilon}}(\tilde{\Pi}_{ab})$, so it suffices to prove that $\alpha_{\underline{\varepsilon}}(\tilde{\Pi}_{ab}) \le w_{\boxplus}(\pi_{\boxplus}(A_\square, B_{\square'}))$. Recall that $\delta = c_4 \underline{\varepsilon}$. We now choose $c_4 \ge 2c_0\sqrt{d}$.

i. For each g_j, $\|g_j\| < \operatorname{diam}(\boxplus) \le \sqrt{d}\ell_i$ so

$$\begin{aligned}
\alpha_{\underline{\varepsilon}}(g_j) &= -\|g_j\| + c_0\underline{\varepsilon}\|g_j\| \\
&< -\|g_j\| + c_0\underline{\varepsilon}\sqrt{d}\ell_i \\
&\le -\|g_j\| + \frac{\delta}{2}\ell_i \qquad (c_4 \ge 2\sqrt{d}c_0, \delta = c_4\underline{\varepsilon}) \\
&\le w_{\boxplus}(B_{\square_j}, A_{\square_{j+1}}).
\end{aligned}$$

ii. For each η_j that is a non-matching bridge arc, $\Phi(\eta_j) = \varnothing$. Let cntr_j be the center of $\square_j$ for all j,

$$\begin{aligned}
\alpha_{\underline{\varepsilon}}((a_j, \Phi(\eta_j), b_{j+1})) &= \|a_j - b_{j+1}\| + c_0\underline{\varepsilon} \cdot \|a_j - b_{j+1}\| \\
&\le \|\mathrm{cntr}_j - \mathrm{cntr}_{j+1}\| + 2 \cdot \frac{\delta}{4}\ell_i + c_0\underline{\varepsilon}\sqrt{d}\ell_i \\
&\le \|\mathrm{cntr}_j - \mathrm{cntr}_{j+1}\| + \frac{\delta}{2}\ell_i + \frac{\delta}{2}\ell_i \\
&\le \|\mathrm{cntr}_j - \mathrm{cntr}_{j+1}\| + \delta\ell_i \\
&= w_{\boxplus}(A_{\square_j}, B_{\square_{j+1}}).
\end{aligned}$$

iii. For each η_j that is a non-matching internal arc, $\Phi(\eta_j)$ is the flex-tip expansion of a minimum-weight path computed at a child cell $\boxplus'$ of $\boxplus$ between subcells $(\Delta, \Delta') \in \mathrm{Ch}(\square) \times \mathrm{Ch}(\square')$. Let a_Δ and $b_{\Delta'}$ be the tips that were added to $\Phi_{\boxplus'}(A_\Delta, B_{\Delta'})$ to compute $w_{\boxplus}(\eta_j)$. In Π, we change the (non-matching) tips of this expansion to a_j and b_{j+1} (from a_Δ and $b_{\Delta'}$), and connecting to the endpoints of the matching edges g_{j-1} and g_{j+1}. Since $a_j \in \square_j$ and $b_{j+1} \in \square_{j+1}$, this perturbation increases the length of the first and last edge in $\Phi(\eta_j)$ by at most $\operatorname{diam}(\square_j) = \operatorname{diam}(\square_{j+1}) \le \frac{\delta}{4}\ell_i$ each, thus

$$\begin{aligned}
\alpha_{\underline{\varepsilon}}((a_j, \Phi(\eta_j), b_{j+1})) &\le \alpha_{\underline{\varepsilon}}((a_\Delta, \Phi(\eta_j), b_{\Delta'})) + \operatorname{diam}(\square_j) + \operatorname{diam}(\square_{j+1}) \\
&\le \alpha_{\underline{\varepsilon}}(a_\Delta, \Phi_{\boxplus'}(A_\Delta, B_{\Delta'}), b_{\Delta'}) + \frac{\delta}{2}\ell_i \\
&\le w_{\boxplus}(A_{\square_j}, B_{\square_{j+1}}).
\end{aligned}$$

Summing over each component in Π,

$$\begin{aligned}
\alpha_{\underline{\varepsilon}}(\hat{\Pi}_{ab}) &= \sum_{\substack{j=0 \\ j \text{ even}}}^{k} \alpha_{\underline{\varepsilon}}((a_j, \Phi(\eta_j), b_{j+1})) + \sum_{\substack{j=1 \\ j \text{ odd}}}^{k-1} \alpha_{\underline{\varepsilon}}(g_j) \\
&\le \sum_{\substack{j=0 \\ j \text{ even}}}^{k} w_{\boxplus}(A_{\square_j}, B_{\square_{j+1}}) + \sum_{\substack{j=1 \\ j \text{ odd}}}^{k-1} w_{\boxplus}(B_{\square_j}, A_{\square_{j+1}}) \\
&= w_{\boxplus}(\pi_{\boxplus}(A_\square, B_{\square'})).
\end{aligned}$$

□

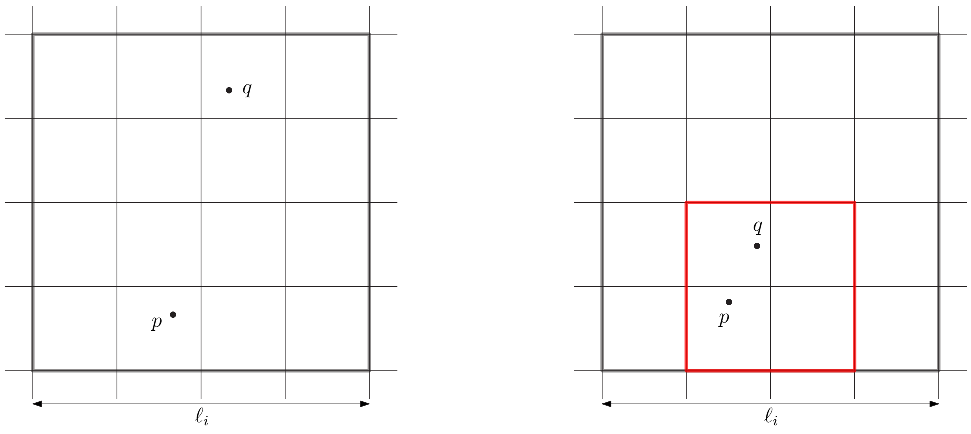


Figure 3.6. Left: Proof of Lemma 3.5 upper bound. The level-i cell in bold. Grid lines are those of $\mathfrak{G}_{i-2}$. **Right**: Proof of Lemma 3.5 lower bound. If p and q are too close in all dimensions, a lower-level cell (red) contains them both.

Corollary 3.4. *If C is a negative cycle in $G_\boxplus$ for some $\boxplus \in \mathscr{C}$, then $\alpha_{\underline{\varepsilon}}(\Phi_\boxplus(C)) < 0$.*

The next two lemmas are technical lemmas.

Lemma 3.5. *Let (p,q) be a pair of points in $A \cup B \subset \mathbb{R}^d$. Let i be the lowest level at which there exists a cell $\boxplus \in \mathscr{C}_i$ that contains both p and q. Then*

i. $\ell_{i-2} \le \|p-q\|_\infty \le \ell_i$ and

ii. $\ell_{i-2} \le \|p-q\|_2 \le \sqrt{d}\ell_i$.

Proof: Part (ii) follows from (i), so we prove (i). The upper bound follows from the fact that the side-length of $\boxplus$ is ℓ_i. Let $p = (p_1, \dots, p_d)$ and $q = (q_1, \dots, q_d)$. Without loss of generality, assume that $q_j \ge p_j$ for all $j = 1, \dots, d$. Assume for contradiction that $\|p-q\|_\infty < \ell_{i-2}$, then $q_j - p_j < \ell_{i-2}$ for all $1 \le j \le d$. For $1 \le j \le d$, let $b_j = 1$ if $(p_j \mod \ell_{i-1}) \ge \ell_{i-2}$ and 0 otherwise. We claim that the (unique) cell $\boxplus'$ in the copy of $\mathfrak{G}_{i-1}$ translated by $\ell_{i-2}(b_1, \dots, b_d)$, i.e., in $\mathfrak{G}_{i-1} + \ell_{i-2}(b_1, \dots, b_d)$, that contains[3] p also contains q. Indeed, suppose $\boxplus' = [0, \ell_{i-1}]^d + (a_1 + b_1\ell_{i-2}, \dots, a_d + b_d\ell_{i-2})$ for some $a_1, \dots, a_d \in \ell_{i-1}\mathbb{Z}$. By construction, $p_j - (a_j + b_j\ell_{i-2}) \le \ell_{i-2}$. Therefore

$$q_j - (a_j + b_j\ell_{i-2}) = q_j - p_j + p_j - (a_j + b_j\ell_{i-2}) < \ell_{i-2} + \ell_{i-2} < \ell_{i-1},$$

or $q_j \le \ell_{i-1} + a_j + b_j\ell_{i-2}$, which implies $q \in [0, \ell_{i-1}]^d + (a_1 + b_1\ell_{i-2}, \dots, a_d + b_d\ell_{i-2}) = \boxplus'$. This contradicts the claim that i is the lowest level for which a cell contains both p and q. Hence, $\|p-q\|_\infty \ge \ell_{i-2}$. $\square$

Lemma 3.6. *Let $\boxplus$ be a level-i cell and $\square, \square'$ be two subcells of $\boxplus$; and let $\boxplus'$ be a level-i' ancestor cell of $\boxplus$ for $i' > i$, and Δ (resp. Δ') a subcell of $\boxplus'$ that is the ancestor subcells of $\square$ (resp. $\square'$). Then,*

$$w_{\boxplus'}(\pi_{\boxplus'}(A_\Delta, B_{\Delta'})) \le w_\boxplus(\pi_\boxplus(A_\square, B_{\square'})) + 2\delta\ell_{i'}.$$

[3]Here we assume that p does not lie on the boundary of the grid. If p lies on the grid boundary then $\boxplus'$ is the cell that contains $p^+ = (p_1 + \varepsilon_0, \dots, p_d + \varepsilon_0)$ where $\varepsilon_0 > 0$ is some infinitesimally small value.

Proof: Let $\boxplus = \boxplus_0, \boxplus_1, \ldots, \boxplus_t = \boxplus'$ be the sequence of parent-child cells starting from $\boxplus$ up to $\boxplus'$. Define $\square_j$ (resp. $\square'_j$) to be the ancestor of $\square$ (resp. $\square'$) in $\boxplus_j$. Let a_j (resp. b_j) be the arbitrary points selected as tips for $\Phi_{\boxplus_j}(A_{\square_j}, B_{\square'_j})$ to set $w_{\boxplus_j}(A_{\square_j}, B_{\square'_j})$, see (8).

For $j \geq 1$, by the internal edge weight constructions (6) and (8),

$$\begin{aligned} w_{\boxplus_j}(\pi_{\boxplus_j}(A_{\square_j}, B_{\square'_j})) &\leq w_{\boxplus_j}(A_{\square_j}, B_{\square'_j}) \\ &\leq \alpha_{\underline{\varepsilon}}((a_j, \Phi_{\boxplus_j}(A_{\square_j}, B_{\square'_j}), b_j)) + \delta\ell_j \\ &\leq w_{\boxplus_{j-1}}(A_{\square_{j-1}}, B_{\square'_{j-1}}) + \delta\ell_j, \end{aligned} \tag{10}$$

where the last inequality follows from Lemma 3.3. Using (10) recursively, we obtain

$$\begin{aligned} w_{\boxplus'}(\pi_{\boxplus'}(A_\Delta, B_{\Delta'})) = w_{\boxplus_t}(\pi_{\boxplus_t}(A_{\square_t}, B_{\square'_t})) &\leq w_{\boxplus_{t-1}}(\pi_{\boxplus_{t-1}}(A_{\square_{t-1}}, B_{\square'_t})) + \delta\ell_{i'} \\ &\leq w_{\boxplus}(\pi_{\boxplus}(A_\square, B_{\square'})) + \delta \sum_{j=1}^{t} \frac{\ell_{i'}}{2^{t-j}} \\ &\leq w_{\boxplus}(\pi_{\boxplus}(A_\square, B_{\square'})) + 2\delta\ell_{i'}. \end{aligned}$$

$\square$

Lemma 3.7 (Lifting Inequality). *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). Let $\boxplus$ be a cell at the smallest level that contains Π; let the level of $\boxplus$ be i. Let $X(p)$ (resp. $X(q)$) be the cluster in $V_\boxplus$ containing p (resp. q). Then there is a path P from $X(p)$ to $X(q)$ in $G_\boxplus$ such that*

$$w_\boxplus(P) \leq \bar{\phi}(\Pi) + \alpha_{\bar{\varepsilon}}(\Pi).$$

Proof: We will prove the statement $w_\boxplus(P) \leq \bar{\phi}(\Pi) + c_5 \cdot i \cdot \delta\|\Pi\|$ for some constant $c_5 \geq 72$. From there, since $i \leq h \leq c_3 \log n$, $\delta = c_4\underline{\varepsilon}$, and $\underline{\varepsilon} = \frac{\bar{\varepsilon}}{c_2 \log n}$, we obtain

$$\begin{aligned} w_\boxplus(P) &\leq \bar{\phi}(\Pi) + c_5(c_3 \log n)c_4 \left(\frac{\bar{\varepsilon}}{c_2 \log n}\right) \cdot \|\Pi\| \\ &\leq \bar{\phi}(\Pi) + c_0\bar{\varepsilon} \cdot \|\Pi\| = \alpha_{\bar{\varepsilon}}(\Pi) \end{aligned} \tag{11}$$

provided that $c_0 \geq c_2c_3c_4c_5$.

We prove the statement by induction on the levels. Let $\boxplus$ be the smallest cell containing Π; say $\boxplus$ is at level i. 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 $\Pi = \langle p_1 = p, p_2, \ldots, p_t = q\rangle$. For a point p_i, let $X(p_i)$ be the cluster in $V_\boxplus$ containing p_i. By Lemma 3.5, $\|\Pi\| \geq \ell_i/4$. We construct P by traversing Π and using a greedy approach. Suppose we have traversed a prefix of Π, constructed a prefix of $P = \langle X_1 = X(p_1), X_2, \ldots, X_j\rangle$, and we are currently at point p_{u_j}. Let $p_{u_{j+1}}$ be the furthest point of Π (along Π) such that $p_{u_{j+1}}$ is still a B-point and $\Pi[p_{u_j}, p_{u_{j+1}}]$ — the portion of Π between p_{u_j} and $p_{u_{j+1}}$ — lies in a child cell of $\boxplus$, i.e., we traverse Π starting from p_{u_j} and go as far as we can to a B-point while remaining in a single child cell of $\boxplus$. We set $X_{j+1} = X(p_{u_{j+1}})$ and set $P = P \circ X_{j+1}$. If $p_{u_{j+1}}$ is the last point of Π, we stop. Otherwise, we set $u_{j+2} = u_{j+1} + 1$ and $X_{j+2} = X(p_{u_{j+2}})$. Since $(p_{u_{j+1}}, p_{u_{j+2}})$ is a matching edge, (X_{j+1}, X_{j+2}) is a matching arc in $G_\boxplus$. If $p_{u_{j+2}}$ is the last point of Π, we stop. Otherwise, we repeat this process.

Let $P = \langle X_1, X_2, \ldots X_s\rangle$. By construction, P is a path from $X(p)$ to $X(q)$ in $G_\boxplus$. It thus suffices to bound $w_\boxplus(P)$. For $1 \leq j \leq s$, let $\square_j$ be the subcell of $\boxplus$ containing X_j, and let cntr_j be the center of $\square_j$. For $1 \leq j < s$, let $\Pi_j = \Pi[p_{u_j}, p_{u_{j+1}}]$. We now relate $w_\boxplus(X_j, X_{j+1})$ to the cost of Π_j. There are three cases:

- (X_j, X_{j+1}) is a **matching arc**. Then, $X_j = B_{\square_j}$ and $X_{j+1} = A_{\square_{j+1}}$. By construction $p_{u_j} \in B_{\square_j}, p_{u_{j+1}} \in A_{\square_{j+1}}$, $\Pi_j = (p_{u_j}, p_{u_{j+1}})$, and $(p_{u_j}, p_{u_{j+1}}) \in M$. Therefore,

$$\begin{aligned} w_{\boxplus}(X_j, X_{j+1}) &= \min_{(b,a)\in M\cap(B_{\square_j}\times A_{\square_{j+1}})} \bar{¢}(b,a) + \delta\ell_i \\ &\le \bar{¢}(p_{u_j}, p_{u_{j+1}}) + \delta\ell_i = \bar{¢}(\Pi_j) + \delta\ell_i. \end{aligned} \tag{12}$$

- (X_j, X_{j+1}) is a **non-matching bridge arc**. In this case, $X_j = A_{\square_j}$, $X_{j+1} = B_{\square_{j+1}}$, $p_{u_j} \in \square_j, p_{u_{j+1}} \in \square_{j+1}$, and $\Pi_j = (p_{u_j}, p_{u_{j+1}})$. Since $\text{diam}(\square_j) = \text{diam}(\square_{j+1}) \le \frac{\delta\ell_i}{4}$, we have

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

Therefore

$$\begin{aligned} w_{\boxplus}(X_j, X_{j+1}) &= \|\text{cntr}_j - \text{cntr}_{j+1}\| + \delta\ell_i \\ &\le \|p_{u_j} - p_{u_{j+1}}\| + \frac{\delta\ell_i}{2} + \delta\ell_i = \bar{¢}(\Pi_j) + \frac{3}{2}\delta\ell_i. \end{aligned} \tag{13}$$

- (X_j, X_{j+1}) is a **non-matching internal arc**. As in the previous case, $X_j = A_{\square_j}$, $X_{j+1} = B_{\square_{j+1}}$, $p_{u_j} \in A_{\square_j}, p_{u_{j+1}} \in B_{\square_{j+1}}$. By construction, there is a child cell $\boxplus'$ of $\boxplus$ that contains Π_j. Let Δ_j (resp. Δ_{j+1}) be the child subcell of $\square_j$ (resp. $\square_{j+1}$) that contains p_{u_j} (resp. $p_{u_{j+1}}$). According to (8), there were arbitrary points $a_j \in A_{\square_j}$, $b_{j+1} \in B_{\square_{j+1}}$ chosen such that

$$\begin{aligned} w_{\boxplus}(X_j, X_{j+1}) &= \min_{\substack{\Delta\in\text{Ch}(\square_j), \Delta'\in\text{Ch}(\square_{j+1}) \\ \boxplus"\in\text{Ch}(\boxplus): \Delta,\Delta'\in\boxplus"}} \alpha_{\underline{\varepsilon}}\left((a_j, \Phi_{\boxplus"}(A_{\Delta}, B_{\Delta'}), b_{j+1})\right) + \delta\ell_i \\ &\le \alpha_{\underline{\varepsilon}}((a_j, \Phi_{\boxplus'}(A_{\Delta_j}, B_{\Delta_{j+1}}), b_{j+1})) + \delta\ell_i \\ &\le w_{\boxplus'}(\pi_{\boxplus'}(A_{\Delta_j}, B_{\Delta_{j+1}}) + \delta\ell_i, \end{aligned} \tag{14}$$

where the last inequality follows from Lemma 3.3.

Let $\hat{\boxplus}$ be the smallest descendant of $\boxplus'$ that contains Π_j, and let $\hat{i} \le i-1$ be the level of $\hat{\boxplus}$ (note that $\hat{\boxplus}$ may be $\boxplus'$ itself). Let $\hat{\Delta}_j$ (resp. $\hat{\Delta}_{j+1}$) be the subcell of $\hat{\boxplus}$ containing p_{u_j} (resp. $p_{u_{j+1}}$). Since $\pi_{\hat{\boxplus}}(A_{\hat{\Delta}_j}, B_{\hat{\Delta}_{j+1}})$ is the minimum-weight path from $A_{\hat{\Delta}_j}$ to $B_{\hat{\Delta}_{j+1}}$ in $G_{\hat{\boxplus}}$, we can derive the following from induction hypothesis:

$$w_{\hat{\boxplus}}(\pi_{\hat{\boxplus}}(A_{\hat{\Delta}_j}, B_{\hat{\Delta}_{j+1}})) \le \bar{¢}(\Pi_j) + c_5\hat{i}\delta\|\Pi_j\|.$$

Then by using Lemma 3.6 and then induction hypothesis, we obtain

$$\begin{aligned} w_{\boxplus'}(\pi_{\boxplus'}(A_{\Delta_j}, B_{\Delta_{j+1}})) &\le w_{\hat{\boxplus}}(\pi_{\hat{\boxplus}}(A_{\hat{\Delta}_j}, B_{\hat{\Delta}_{j+1}})) + 2\delta\ell_{i-1} \\ &\le \bar{¢}(\Pi_j) + c_5\hat{i}\delta\|\Pi_j\| + 2\delta\ell_{i-1} \\ &\le \bar{¢}(\Pi_j) + c_5(i-1)\delta\|\Pi_j\| + \delta\ell_i. \end{aligned} \tag{15}$$

Substituting (15) into (14) we obtain

$$w_{\boxplus}(X_j, X_{j+1}) \le \bar{¢}(\Pi_j) + c_5(i-1)\delta\cdot\|\Pi_j\| + 2\delta\ell_i. \tag{16}$$

By combining (12), (13), and (16),

$$\begin{aligned} w_{\boxplus}(P) &= \sum_{j=1}^{s-1} w_{\boxplus}(X_j, X_{j+1}) \\ &\le \sum_{j=1}^{s-1} \left(\bar{¢}(\Pi_j) + c_5(i-1)\delta\|\Pi_j\| + 2\delta\ell_i \right) \\ &= \bar{¢}(\Pi) + c_5(i-1)\delta\|\Pi\| + 2(s-1)\cdot\delta\ell_i. \end{aligned} \tag{17}$$

Suppose (X_j, X_{j+1}) is a non-matching arc in P. If it is not the last non-matching arc, i.e., $j \le s-3$, then by construction there is no child cell of $\boxplus$ that contains $\Pi_j \circ \Pi_{j+1} \circ \Pi_{j+2}$. Therefore by Lemma 3.5.i, $\|\Pi_j \circ \Pi_{j+1} \circ \Pi_{j+2}\| \ge \ell_{i-1}/4 = \ell_i/8$. Π has at least $\left\lfloor \frac{s-1}{2} \right\rfloor$ non-matching arcs. Summing this quantity over all but the last non-matching arc, we obtain

$$\sum_j \|\Pi_j \circ \Pi_{j+1} \circ \Pi_{j+2}\| \ge \left(\left\lfloor \frac{s-1}{2} \right\rfloor - 1 \right) \frac{\ell_i}{8}.$$

On the other hand, each edge of Π is being counted at most twice in the sum. Hence, the sum is at most $2\|\Pi\|$. We therefore conclude that $\left\lfloor \frac{s-1}{2} \right\rfloor \frac{\ell_i}{8} \le 2\|\Pi\|$, or

$$(s-2)\ell_i \le 32\|\Pi\|. \tag{18}$$

Plugging (18) into (17) and using the fact that $\|\Pi\| \ge \ell_i/4$,

$$\begin{aligned} w_{\boxplus}(P) &\le \bar{¢}(\Pi) + c_5(i-1)\delta\|\Pi\| + 2(s-1)\cdot\delta\ell_i \\ &\le \bar{¢}(\Pi) + c_5(i-1)\delta\|\Pi\| + 64\delta\|\Pi\| + 2\delta\ell_i \\ &\le \bar{¢}(\Pi) + c_5(i-1)\delta\|\Pi\| + 72\delta\|\Pi\| \\ &\le \bar{¢}(\Pi) + c_5 i\cdot\delta\|\Pi\|, \end{aligned}$$

provided that $c_5 \ge 72$. This completes the proof of the lemma. $\square$

By combining the Lifting and Expansion Inequalities, we can now prove algorithm correctness. We need one technical lemma arguing the height h is sufficient.

Lemma 3.8. *Let Π be either a reducing cycle, or the residual augmenting path (in $\vec{G}_M$) with the minimum $\bar{\varepsilon}$-adjusted cost α^*. Then there exists a level $i \in \{0,\dots,h\}$ and $\boxplus \in \mathscr{C}_i$ such that Π lies completely in $\boxplus$.*

Proof: For either option of Π from the lemma statement (reducing cycle, low-adjusted-cost augmenting path), Lemma 2.3 bounds the Euclidean length of the polygonal curve: $\|\Pi\| \le \frac{27\sqrt{d}n}{\varepsilon}$. For $1 \le j \le d$, let $p_j^- = \min_{p\in\Pi} x_j(p)$ and $p_j^+ = \max_{p\in\Pi} x_j(p)$. Then, $p_j^+ - p_j^- \le \|\Pi\|$. Let $p^- = (p_1^-,\dots,p_d^-)$ and $p^+ = (p_1^+,\dots,p_d^+)$; $\|p^+ - p^-\|_\infty \le \|\Pi\|$, so there is a cell $\boxplus \in \mathscr{C}_i$ for $i = 2 + \left\lceil \log\left(\frac{27\sqrt{d}n}{\varepsilon}\right) \right\rceil$ that contains both p^- and p^+. Let $Box := \prod_{j=1}^d [p_j^-, p_j^+]$ be the axis-aligned hypercube determined by p^- and p^+. Then, $\Pi \subseteq Box \subseteq \boxplus$. Assuming $c_3 > 0$ is chosen sufficiently large and $\varepsilon > 1/n$ so that $h = c_3\lceil \log n \rceil \ge 2 + \left\lceil \log\left(\frac{27\sqrt{d}n}{\varepsilon}\right) \right\rceil$, the lemma holds. $\square$

Lemma 3.9. FINDPATH *returns an augmenting path Π such that $\alpha_{\underline{\varepsilon}}(\Pi) \le \alpha^*_{\bar{\varepsilon},M}$.*

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., $\alpha_{\bar{\varepsilon}}(\Pi^*) = \alpha^*_{\bar{\varepsilon},M}$. Let $a^* \in A$ (resp. $b^* \in B$) be the initial (resp. final) endpoint of Π^*. By Lemma 3.8, there is a cell in $\mathscr{C}$ that contains Π^*. Let $\boxplus$ be the smallest cell in $\mathscr{C}$ that contains Π^* (if there is more than one, choose an arbitrary one). Let $\square$ (resp. $\square'$) be the subcell of $\boxplus$ that contains a (resp. b). By Lifting Lemma 3.7,

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

Since both $A_{\square}$ and $B_{\square'}$ are unsaturated, REPAIR procedure chose a representative path $\Pi_{\square,\square'} = (a, \Phi_{\boxplus}(A_{\square}, B_{\square'}), b)$ for some $a \in A_{\square}, b \in B_{\square'}$. By Expansion Lemma 3.3,

$$\alpha_{\underline{\varepsilon}}(\Pi_{\square,\square'}) \le w_{\boxplus}(\pi_{\boxplus}(A_{\square}, B_{\square'})). \tag{19}$$

Let $(\square,\square')^*_{\boxplus}$ be the pair chosen by REPAIR($\boxplus$) in Step (v), then

$$\alpha_{\underline{\varepsilon}}(\Pi_{(\square,\square')^*_{\boxplus}}) \le \alpha_{\underline{\varepsilon}}(\Pi_{\square,\square'}) \le \alpha^*.$$

Since $(\square,\square')^*_{\boxplus} \in \mathit{OptPairs}$, FINDPATH returns an augmenting path Π with $\alpha_{\underline{\varepsilon}}(\Pi) \le \alpha_{\underline{\varepsilon}}(\Pi_{(\square,\square')^*_{\boxplus}}) \le \alpha^*$. □

Lemma 3.10. *Let M be an intermediate matching. If $\vec{G}_M$ contains an alternating cycle Γ with $\alpha_{\bar{\varepsilon}}(\Gamma) < 0$ then* AUGMENT *returns a reducing cycle.*

Proof: Suppose $\vec{G}_M$ contains a cycle Γ with $\alpha_{\bar{\varepsilon}}(\Gamma) < 0$. Then, as in the proof of Lemma 3.9, we can argue that the compressed graph $G_{\boxplus}$ at the smallest cell $\boxplus \in \mathscr{C}$ containing Γ contains a negative cycle C. Then, $\alpha_{\underline{\varepsilon}}(\Phi_{\boxplus}(C)) < 0$, i.e., $\Phi_{\boxplus}(C)$ is a reducing cycle, which will be returned by REPAIR. □

Corollary 3.11. *The invariant CI holds at the beginning of each iteration.*

4 Computing and Maintaining Flex-Tip Expansions (FTEs)

In this section we describe the algorithms and data structure for computing the flex-tip expansions (FTEs) of compressed paths at all cells. We first describe a high-level representation of FTEs, then we describe two high-level procedures that construct FTEs. Next, we describe the data structure to maintain FTEs. Finally, we describe the REPORT and COST procedures as well as the basic operations on the data structure needed by the high-level procedures. Recall that cells in $\mathscr{C}$ are processed in a bottom up manner. We focus on computing FTEs at a cell $\boxplus$, assuming that FTEs at all children cells of $\boxplus$ have been computed and that minimum-weight compressed paths between all pairs of subcells of $\boxplus$ have been computed.

Recall that an FTE Φ is represented as a sequence $\langle (b_1, a_1), (b_2, a_2), \ldots, (b_t, a_t) \rangle$ of matching edges, which models the alternating path $\langle b_1, a_1, b_2, a_2, \ldots, a_t \rangle$ in $\vec{G}_M$. If this alternating path is not simple (i.e., it contains a cycle) then a matching edge appears more than once in the path as well as in its representing sequence. Throughout this section we will view Φ as a sequence of matching edges. The only exception is when we need to compute the adjusted cost of a subpath of Φ in which case we treat it as a path in $\vec{G}_M$. We define $s := |V_{\boxplus}| = (\varepsilon^{-1} \log n)^{O(d)}$ as the maximum number of clusters in a cell.

4.1 Pathlets and composition of an FTE

Given an FTE $\Phi = \langle e_1, \ldots, e_k \rangle$ and two indices $1 \le i \le j \le k$, we define the *splice* operators:

$$\begin{aligned} e_i \blacktriangleright \Phi \blacktriangleleft e_j &:= \langle e_i, e_{i+1}, \ldots, e_{j-1}, e_j \rangle && \text{(inclusive)} \\ e_i \triangleright \Phi \triangleleft e_j &:= \langle e_{i+1}, e_{i+2}, \ldots, e_{j-2}, e_{j-1} \rangle && \text{(exclusive)} \end{aligned}$$

We define a *pathlet* ϕ of Φ to be $e_i \blacktriangleright \Phi \blacktriangleleft e_j$ for some $1 \le i \le j \le k$. Obviously, Φ is a pathlet of itself. We write one-sided splices to represent prefixes and suffixes, e.g., $e_i \blacktriangleright \Phi = \langle e_i e_{i+1}, \dots, e_k \rangle$. We mix the inclusive and exclusive splices when convenient, e.g., $e_i \triangleright \Phi \blacktriangleleft e_j = \langle e_{i+1}, \dots, e_{j-1}, e_j \rangle$. 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 both ϕ and ϕ' *originate from* Φ. To simplify notation, we consider single matching edges to be (trivial) pathlets.

Recall that for an internal arc $\gamma = (A_\square, B_{\square'})$ at $\boxplus$, the FTE of γ, $\Phi(\gamma)$, is $\Phi_{\boxplus'}(A_\triangle, B_{\triangle'})$ for some child cell $\boxplus'$ of $\boxplus$ and children clusters $A_\triangle, B_{\triangle'}$ of $A_\square, B_{\square'}$. We have $\Phi(\gamma)$ at our disposal when we compute FTEs at $\boxplus$. Of particular interest will be pathlets of $\Phi(\gamma)$, which we will often refer to as pathlets of γ.

For a pair of subcells $\square, \square'$ of $\boxplus$, $\Phi_\boxplus(A_\square, B_{\square'})$ is represented as a sequence of at most s pathlets $\phi_1 \circ \phi_2 \circ \dots \circ \phi_t$ for $t \le s$, where ϕ_i is either a matching edge corresponding to a matching arc of $\pi_\boxplus(A_\square, B_{\square'})$ or a pathlet of the FTE of an internal arc of $\pi_\boxplus(A_\square, B_{\square'})$. We note that not all matching/internal arcs of $\pi_\boxplus(A_\square, B_{\square'})$ may have a corresponding pathlet in $\Phi_\boxplus(A_\square, B_{\square'})$. A pathlet ϕ of $\Phi_\boxplus(A_\square, B_{\square'})$ delimited by edges e^- and e^+ is represented in a similar manner as follows. Suppose $e^- \in \phi_i$ and $e^+ \in \phi_j$, then ϕ is represented as $(e^- \blacktriangleright \phi_i) \circ \phi_{i+1} \circ \dots \circ \phi_{j-1} \circ (\phi_j \blacktriangleleft e^+)$. FTEs and pathlets are maintained using a data structure, which consists of a collection of trees and Boolean look-up tables, described 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, as described in Section 4.3.

- Intersects(ϕ_1, ϕ_2): Given two pathlets ϕ_1 and ϕ_2 that originate from FTEs 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 FTEs where $\phi_1 \cap \phi_2 \neq \emptyset$, return *references* to four edges[4]: 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, if $\phi = \langle e_1, e_2, \dots, e_k \rangle$, return a reference to $e_{\lceil k/2 \rceil}$.

- PathCost$(\Pi = \phi_1 \circ \dots \circ \phi_k)$: Given a possibly non-simple path $\Pi \in \vec{G}_M$ as a sequence pathlets $\phi_1, \cdots \phi_k$, return $\alpha_{\underline{\varepsilon}}(\Pi)$.

- CycleCost$(\Gamma = \phi_1 \circ \dots \circ \phi_k)$: Given a possibly non-simple cycle $\Gamma \in \vec{G}_M$ as a sequence of pathlets $\phi_1, \cdots \phi_k$, return $\alpha_{\underline{\varepsilon}}(\Gamma)$.

- PathletReport(ϕ): Given a pathlet ϕ, return the alternating path represented by ϕ.

Before describing the data structure and these operations (Sections 4.3 and 4.4), we describe the two high-level procedures that compute FTEs, using these operations. We will use $t(n)$ to denote the maximum time taken by the above procedures except PathletReport. We will see below in Section 4.4 that $t(n) = (\varepsilon^{-1} \log n)^{O(d)}$ and PathletReport takes $O(t(n) + k \log n)$ time where k is the length of the path returned by the procedure.

4.2 High-level simplification procedures

We now describe the procedures ConstructFTE and SimpleReducingSubcycle, defined in Section 3, that are used to construct FTEs.

[4]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.

ConstructFTE. Given a compressed path $\pi = \langle \eta_0, g_1, \eta_2, \ldots, \eta_k \rangle$ in $G_{\boxplus}$ where each η_i is an internal arc of $G_{\boxplus}$ and g_i is the longest matching edge corresponding to a matching arc of $G_{\boxplus}$, we write the intermediate expansion $\tilde{\Pi}$ of π, defined in (9), as

$$\tilde{\Pi} = \Phi(\eta_0) \circ g_1 \circ \Phi(\eta_2) \circ \cdots \circ \Phi(\eta_k) = \phi_0 \circ \phi_1 \circ \phi_2 \circ \cdots \circ \phi_k \tag{20}$$

where each ϕ_i is either $\Phi(\eta_i)$ or g_i. We process the ϕ_i's in sequence and grow a pathlet sequence $\Xi = \langle \phi'_0, \ldots, \phi'_t \rangle$ where $\phi'_0 \circ \cdots \circ \phi'_t$ is simple.

Initially, $\Xi = \varnothing$ and the first pathlet we process is ϕ_0. To process $\phi_i \subseteq \tilde{\Pi}$, 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 $\text{Intersects}(\phi_i, \phi'_j)$ 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 $\phi'_{t+1} = \phi_i$, append ϕ'_{t+1} to Ξ, and continue on to ϕ_{i+1}. If there is an intersection against ϕ_j, we invoke $\text{LastCommonEdge}(\phi_i, \phi'_j)$ to find e_1, the *last* edge in ϕ_i that intersects 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.[5]

Next, check $\alpha_{\underline{\varepsilon}}(C)$ using $\text{CycleCost}(C)$. If C is reducing, we abort the procedure and return the simple reducing cycle $\hat{C}$ returned by $\text{SimpleReducingSubcycle}(C)$. 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 $\Xi = \langle \phi'_1, \ldots, \phi'_j, \phi'_{j+1} \rangle$. Note that this may remove existing elements from Ξ. Then, we continue on to ϕ_{i+1}. If all elements of $\tilde{\Pi}$ are processed without aborting, then we return $\Phi = \phi'_1 \circ \phi'_2 \circ \cdots \circ \phi'_t$, where $\Xi = \langle \phi'_0, \ldots, \phi'_t \rangle$.

SimpleReducingSubcycle. This procedure is either called by Repair to expand a negative compressed cycle φ in $G_{\boxplus}$ or by ConstructFTE. In the former case, we first construct an intermediate expansion of φ of the form (20) as above. So we assume that the reducing cycle is represented as a pathlet sequence $C = \langle \phi_1, \ldots, \phi_k \rangle$. Like ConstructFTE, 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. There are base cases when C is composed of less than three pathlets, so assume first that $k \geq 3$.

Initially $\Xi = \varnothing$, and the first element we process is ϕ_1. Suppose we are processing $\phi_i \subseteq C$. For each $\phi'_j \in \Xi$ in reverse order, we query $\text{Intersects}(\phi_i, \phi'_j)$. 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 $\phi'_{t+1} = \phi_i$, 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 $\text{LastCommonEdge}(\phi_i, \phi'_j)$ to acquire e_1, the *last* edge of ϕ_i in the intersection. There are two subcycles about e_1:

$$\begin{aligned} C_0 &:= (e_1 \blacktriangleright \phi'_j) \circ \phi'_{j+1} \circ \cdots \circ \phi'_{i-1} \circ (\phi_i \triangleleft e_1) \\ C_I &:= \phi'_1 \circ \cdots \circ (\phi'_j \triangleleft e_1) \circ (e_1 \blacktriangleright \phi_i) \circ \phi_{i+1} \circ \cdots \circ \phi_t \end{aligned}$$

Neither C_0 nor C_I may be simple, but:

- In C_0, the only potentially intersecting pathlets are $e_1 \blacktriangleright \phi'_j$ and $\phi_i \triangleleft e_1$. This is a base-case we call *only-two-intersecting pathlets*, see below.
- In C_I, $(\phi'_j \triangleleft e_1) \cap (e_1 \blacktriangleright \phi_i) = \varnothing$, so the number of pairs of intersecting component pathlets in C_I is strictly less than in C.

[5]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.

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 CycleCost(C_0). 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 a single pathlet, then it is simple since the originating FTE must also be simple. Next, if at most two of the component pathlets of C are intersecting, i.e., $C = \phi_1 \circ \cdots \circ \phi_k$ and we know only ϕ_1 and ϕ_k intersect, we use the following binary search algorithm since the the above procedure will not make progress quickly enough.

Given $C = \phi_1 \circ \cdots \circ \phi_k$ where only ϕ_1 and ϕ_k intersect, the *only-two-intersecting* pathlets base-case is handled as follows. Let $\phi_1 = e_L \blacktriangleright \Phi_1 \triangleleft e^+$ and $\phi_k = e^- \blacktriangleright \Phi_k \triangleleft e_R$ for two edges e_L, e_R (initially, $e_L = e_R$). At each step, we query Intersects($(e_L \blacktriangleright \Phi_1 \triangleleft e^+), (e^- \blacktriangleright \Phi_k \triangleleft e_R)$). If there is no intersection then C is simple, so we return C. If there is an intersection, then we invoke Median($e_L \blacktriangleright \Phi_1 \triangleleft e^+$) to acquire f, the median edge of $e_L \blacktriangleright \Phi_1 \triangleleft e^+$, and query Intersects($(e_L \blacktriangleright \Phi_1 \triangleleft f), (e^- \blacktriangleright \Phi_k \triangleleft e_R)$). If there is no intersection up to the median, then we shrink the left pathlet to $f \blacktriangleright \Phi_1 \triangleleft e^+$ by setting $e_L = f$, and attempt again with the new pathlet. Otherwise, if $(e_L \blacktriangleright \Phi_1 \triangleleft f) \cap (e^- \blacktriangleright \Phi_k \triangleleft e_R) \neq \varnothing$, we invoke LastCommonEdge($(e_L \blacktriangleright \Phi_1 \triangleleft f), (e^- \blacktriangleright \Phi_k \triangleleft e_R)$) to acquire e_1, the *last* edge of $e_L \blacktriangleright \Phi_1 \triangleleft f$ that appears in the intersection. There are two subcycles about e_1:

$$\begin{aligned}\widetilde{C}_0 &:= (e_L \blacktriangleright \Phi_1 \triangleleft e_1) \circ (e_1 \blacktriangleright \Phi_k \triangleleft e_R)\\ \widetilde{C}_I &:= (e_1 \blacktriangleright \Phi_1 \triangleleft e^+) \circ \phi_2 \circ \cdots \circ \phi_{k-1} \circ (e^- \blacktriangleright \Phi_k \triangleleft e_1)\end{aligned}$$

In $\widetilde{C}_0$, $|e_L \blacktriangleright \Phi_1 \triangleleft e_1| < |\phi_1|/2$. In $\widetilde{C}_I$, $e_1 \blacktriangleright \Phi_1 \triangleleft e^+$ has no intersection edges before f (e_1 is not included in $e^- \blacktriangleright \Phi_k \triangleleft e_1$). Check $\alpha_{\underline{\varepsilon}}(\widetilde{C}_0)$ using CycleCost($\widetilde{C}_0$). 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 $e_L = e_1$ and setting $e_R = f$. 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.

Analysis. The following lemma, whose proof is straightforward, summarizes this subsection.

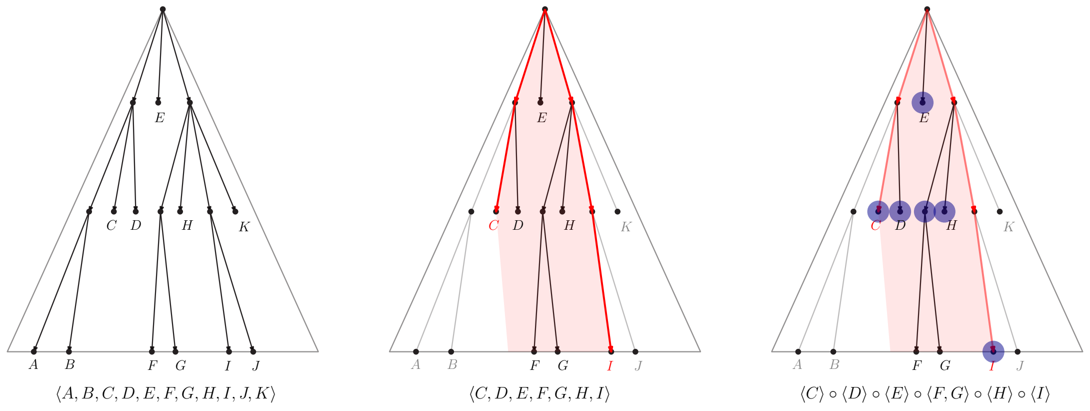
Lemma 4.1. *Assume $t(n)$ is the maximum time taken by the operations on the FTE data structure besides* Report. *Then:*

(i) *Given a compressed path P in $G_{\boxplus}$,* ConstructFTE *either returns an FTE Φ of P or a simple reducing cycle in $O(s^4 \cdot t(n))$ time.*

(ii) SimpleReducingSubcycle *returns a simple reducing cycle in $O(s^4 \cdot t(n))$ time.*

4.3 Compact representation of pathlets

We use trees to represent FTEs and pathlets, which we describe at a conceptual level first. An FTE $\Phi = \Phi_{\boxplus}(A_{\square}, B_{\square'})$ will be a tree $T(\Phi)$ whose leaves are its matching edges, in sequence from left to right, that we call an *FTE-tree*. To search for 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 a root-leaf path that ends at the leaf representing e. Then, the pathlet $\phi = e^- \blacktriangleright \Phi \blacktriangleleft e^+$ for $e^-, e^+ \in \Phi$ is the subtree of $T(\Phi)$ lying between and including the two spines to e^- and e^+. We call this subtree a *pathlet-subtree* and denote it as $T(\Phi, \phi)$. 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^-, e^+ then all children of u that appear in $T(\Phi)$ appear in $T(\Phi, \phi)$ as well; 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



E
C
D
H
K
A
B
F
G
I
J
$\langle A,B,C,D,E,F,G,H,I,J,K\rangle$
$\langle C,D,E,F,G,H,I\rangle$
$\langle C\rangle \circ \langle D\rangle \circ \langle E\rangle \circ \langle F,G\rangle \circ \langle H\rangle \circ \langle I\rangle$

Figure 4.1. **Left**: An FTE-tree representing a sequence $\langle A,B,C,D,E,F,G,H,I,J,K\rangle$. **Center**: Pathlet-subtree of the same FTE-tree, representing subsequence $\langle C,D,E,F,G,H,I\rangle$. The spines are marked in red. **Right**: Canonical nodes of the same pathlet-subtree, marked in blue.

represented by simply storing the spines of e^- and e^+, which takes $O(h) = O(\log n)$ space. A matching edge e is represented as a single-node tree denoted by $T(e)$.

FTE-trees are defined recursively. Recall that ConstructFTE creates the FTE for $\Phi = \Phi_{\boxplus}(A_\square, B_{\square'})$, that has the form $\Phi_{\boxplus}(A_\square, B_{\square'}) = \phi_1 \circ \phi_2 \circ \cdots \circ \phi_t$, where each ϕ_j is a trivial pathlet consisting of a matching edge or a pathlet of $\Phi(\eta_j)$ for an internal arc η_j of $G_{\boxplus}$. $T(\Phi)$ consists of a root node plus t subtrees $T_1, \ldots, T_t$ from left to right, i.e., 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 $T_j = T(\Phi(\eta_j), \phi_j)$. We identify the j-th child of the root of $T(\Phi)$ which is an internal node of $T(\Phi)$, with η_j. If ϕ_j is a trivial pathlet of a single matching edge g_j, then T_j is a single-node tree $T(g_j)$, which is a leaf of $T(\Phi)$.

Canonical pathlets and canonical nodes. Let u be an internal node of $T(\Phi)$. Recursively, the subtree of $T(\Phi)$ rooted at u, denoted T_u, is a pathlet-subtree $T(\Phi(\eta), \phi)$ (which in turn is a subtree of $T(\Phi(\eta))$, where η is an internal arc at a descendant cell of $\boxplus$ and ϕ is a pathlet of $\Phi(\eta)$ consisting of the matching edges stored at the leaves of T_u. We identify T_u with $T(\Phi(\eta), \phi)$ and use ϕ_u to denote the pathlet ϕ of $\Phi(\eta)$. We call ϕ_u a *canonical pathlet* of $\Phi(\eta)$. If u is a leaf node then ϕ_u is the matching edge stored at u and we regard ϕ_u as a (trivial) canonical pathlet. An FTE Φ of length k has $\Theta(k^2)$ possible pathlets, but as we will see the number of canonical pathlets of Φ is much smaller.

For a pathlet ϕ of an FTE 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 $N(T(\Phi, \phi)) = \langle N_1, N_2, \ldots, N_k\rangle$. One has $|N(T(\Phi, \phi))| = O(sh)$. Note that each canonical node N_i has a canonical pathlet ϕ_{N_i} associated with it and $\phi = \phi_{N_1} \circ \cdots \circ \phi_{N_k}$. Alternatively, any pathlet can be represented as the concatenation of $O(sh)$ canonical pathlets.

We associate the following auxiliary information with each node u of $T(\Phi)$ (here we view the canonical pathlet ϕ_u as a path in $\vec{G}_M$ and not just a sequence of matching edges):

(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) $\alpha_{\underline{\varepsilon}}(\phi)$, the $\underline{\varepsilon}$-adjusted cost of ϕ.

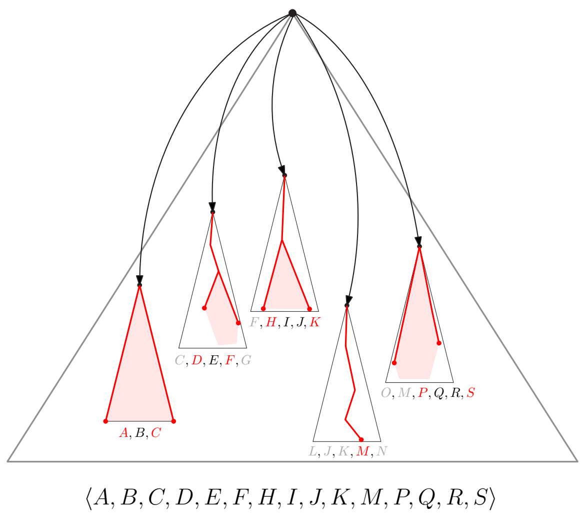


Figure 4.2. An FTE-tree of level i is constructed recursively from the pathlet-subtrees of level-$(i-1)$ FTE-trees.

Compact form. It is too expensive to explicitly represent the entire pathlet-subtree for every canonical pathlet we construct. Instead, we represent pathlet-subtrees implicitly using the spines; i.e., for a pathlet $\phi = e^- \blacktriangleright \Phi' \blacktriangleleft e^+$, we represent $T(\Phi', \phi)$ by a pointer to $T(\Phi')$ plus the (explicitly represented) two spines to e^- and e^+, as described in the beginning of the subsection. The depth of an FTE-tree is at most h, so a spine can be stored in $O(h)$ space. Finally, if an FTE is computed by ConstructFTE as $\Phi = \phi_1 \circ \cdots \circ \phi_t$, then $T(\Phi)$ is stored as the root of $T(\Phi)$ plus the sequence of pathlet subtrees $\phi_1, \ldots, \phi_t$ in their compact forms. We call the root as well as the spine nodes of these pathlet-subtrees *exposed* nodes of $T(\Phi)$. For each exposed node of $T(\Phi)$, we store the auxiliary information described above (i.e., endpoints, adjusted cost, length) with respect to $T(\Phi)$. Since there are up to s children, the space used by this compact representation of an FTE-tree is $O(sh)$, and this compact representation is returned as the output of ConstructFTE. The following lemma suggests how the complete $T(\Phi)$ can be accessed from its compact form.

Lemma 4.2. *Let u be a node of $T(\Phi)$ such that its parent v is an exposed node of $T(\Phi)$, and let Φ_u (resp. Φ_v) be the FTE from which the canonical pathlet ϕ_u (resp. ϕ_v) at u (resp. v) originates. Then one of the children FTE-trees of the root of $T(\Phi_v)$ is identified with Φ_u and the pathlet-subtree rooted at this child is the same as the pathlet-subtree $T(\Phi_u, \phi_u)$ rooted at u in $T(\Phi)$.*

Cells, level, and rank. Let Φ be the FTE of a compressed path in a cell $\boxplus$ of level i. We assign the *cell* and *level* of Φ to be $\boxplus$ and i, respectively. For a pathlet ϕ originating from Φ, we assign the cell and level of ϕ to be those of Φ. Recall that FTE $\Phi(\gamma)$ of an internal arc γ at $\boxplus$ is the FTE of a compressed path at a child cell of $\boxplus$, so $\Phi(\gamma)$ inherits cell and level assigned to that FTE. For an internal 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 $\text{level}(u)$ to be the level of ϕ_u. If u is a leaf node, i.e., ϕ_u is a matching edge, then we assign $\boxplus(u)$, $\text{level}(u)$ as follows: Let v be the parent of u in $T(\Phi)$. If a child cell Δ of $\boxplus(v)$ contains ϕ_u then we set $\boxplus(u) = \Delta$ and $\text{level}(u) = \text{level}(v) - 1$, otherwise we set $\boxplus(u) = \boxplus(v)$ and $\text{level}(u) = \text{level}(v)$.

Roughly speaking, the *rank* of a canonical pathlet ϕ originating from Φ, denoted $\text{rank}(\phi)$, is the smallest level of an FTE whose FTE-tree has ϕ appear in as a canonical pathlet. Formally, $\text{rank}(\phi)$ is the smallest value i for which there is an FTE Φ' of level i such that $T(\Phi, \phi)$ is the subtree of $T(\Phi')$ rooted at an internal node u of $T(\Phi')$.

The following lemma helps bound the number of canonical pathlets.

Lemma 4.3. *Let ϕ_u be a canonical pathlet associated with a node u of an FTE-tree $T9\Phi)$. Then there exists another FTE Φ' with $\text{level}(\Phi') \le \text{level}(\Phi)$ and a node u' in $T(\Phi')$ such that $\phi_u = \phi_{u'}$ and u' is an exposed node of $T(\Phi')$.*

Lemma 4.4. *The following statements hold:*

(i) There are $O(ns^3h^2)$ canonical pathlets.

(ii) For any cell $\boxplus \in \mathscr{C}$, there are $O(s^3h)$ canonical pathlets ϕ such that $\boxplus(\phi) = \boxplus$.

Proof: First we prove (i). By Lemma 4.3, any canonical pathlet can be identified with an exposed node of an FTE-tree. An FTE-tree has $O(sh)$ exposed nodes, there are $O(s^2)$ FTEs at a cell, and each level has $O(n)$ non-empty cells, so (i) follows.

Next we prove (ii). Let ϕ be a canonical pathlet with $\boxplus(\phi) = \boxplus$. Then there is an FTE Φ at an ancestor cell of $\boxplus$ such that ϕ is associated with an exposed node of $T(\Phi)$. Each spine of $T(\Phi)$ has at most one node u where $\boxplus(u) = \boxplus$, and $T(\Phi)$ has $O(s)$ spines. Since $\boxplus$ has $O(h)$ ancestor cells and each cell has $O(s^2)$ FTEs, the bound follows. □

Intersection tables. For an FTE Φ, we dynamically maintain $\mathscr{P}(\Phi)$, the set of canonical pathlets that originate from Φ. $\mathscr{P}(\Phi)$ is updated as we process ancestor cells of the cell of Φ. After processing the ancestors at level i, we have all canonical pathlets of Φ of rank at most i. Let η_1, η_2 be two internal or matching arcs in sibling cells. Here, we assume that if η is a matching arc then $\Phi(\eta)$ is the longest matching edge corresponding to η. We maintain an *intersection table* $\beta_{\eta_1,\eta_2} : \mathscr{P}(\Phi(\eta_1)) \times \mathscr{P}(\Phi(\eta_2)) \to \{0,1\}$ 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 FTEs 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 FTEs 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 the intersection query INTERSECTION we present later in this section.

Since $|\mathscr{P}(\Phi(\eta))| = O(s^3h)$ and the total number of canonical pathlets is $O(ns^3h^2)$, 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 FTEs 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

The high-level idea of INTERSECTS is to reduce the query into a set of intersection table look-ups, i.e., comparisons between same-level canonical pathlets. LASTCOMMONEDGE finds the extreme intersection edge by using INTERSECTS to guide a pruned search.

INTERSECTS. Given two pathlets ϕ_1,ϕ_2 originating from FTEs Φ_1,Φ_2 of lower levels, return 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 $T(\Phi_2,\phi_2)$. The query can be phrased in terms of canonical nodes of the two pathlet-subtrees: Does there exist a pair of canonical nodes $N_1 \in N(T(\Phi_1,\phi_1))$ and $N_2 \in N(T(\Phi_2,\phi_2))$ such that $\phi_{N_1} \cap \phi_{N_2} \neq \emptyset$? We present a procedure for testing the intersection status of two non-exposed nodes, then answer the original query by testing all pairs of canonical nodes in $N(T(\Phi_1,\phi_1)) \times N(T(\Phi_2,\phi_2))$.

Suppose we are testing $N_1 \in N(T(\Phi_1, \phi_1))$ and $N_2 \in N(T(\Phi_2, \phi_2))$. If $\boxplus(N_1) \cap \boxplus(N_2) = \varnothing$, then $\phi_{N_1} \cap \phi_{N_2} = \varnothing$ so we return no. Assume that $\boxplus(N_1) \cap \boxplus(N_2) \neq \varnothing$. If $\text{level}(N_1) = \text{level}(N_2)$, then we are able to directly compare ϕ_{N_1}, ϕ_{N_2} through the intersection table: Let $\Phi(\eta_1)$ (resp. $\Phi(\eta_2)$) be the FTE from which N_1 (resp. N_2) originates, then the table β_{η_1,η_2} exists and stores a bit $\beta_{\eta_1,\eta_2}(N_1, N_2)$ since both are canonical pathlets of lower rank.

Otherwise, assume without loss of generality that $\text{level}(N_1) > \text{level}(N_2)$. If N_1 is a leaf, i.e., a matching edge e, then let $\boxplus = \boxplus(N_1)$. Either e is also the longest matching edge for a matching arc of some child $\boxplus' \in \text{Ch}(\boxplus)$, or e appears in no descendants of $\boxplus$. In the latter case, it is impossible for e to appear in N_2 so we return no. In the former case, let N_1 be a matching arc between subcells $\square, \square'$, then there exists a matching arc N' with edge e for a pair of children Δ, Δ' where Δ (resp. Δ') is a child of $\square$ (resp. $\square'$). We recursively test N' against N_2; this repeats until the two nodes are same-level. On the other hand, if N_1 is a pathlet of an internal arc, then we recursively test N_2 against the children of N_1 but prune — we only recurse on nodes $N' \in \text{Ch}(N_1)$ such that $\boxplus(N') \cap \boxplus(N_2) \neq \varnothing$.

Lemma 4.6. *For a fixed pair of canonical nodes N_1 and N_2,* Intersects *takes $O(s^4h^2)$ time.*

Proof: If $\text{level}(N_1) = \text{level}(N_2)$, then the lemma obviously holds because the procedure performs one table look up. Without loss of generality, assume that $\text{level}(N_1) > \text{level}(N_2)$, and let T_1 be the subtree of $T(\Phi_1)$ rooted at N_1. If the procedure visits a child of a node N' in T_1 then $\text{level}(N') > \text{level}(N_2)$ and $\boxplus(N') \cap \boxplus(N_2) \neq \varnothing$. For any fixed level $i \geq \text{level}(N_2)$, there are $O(1)$ cells Δ such that $\Delta \cap \boxplus(N_2) \neq \varnothing$. By Lemma 4.4 (ii), there are $O(s^3h)$ canonical pathlets φ with $\boxplus(\varphi) = \Delta$. Hence, there are $O(s^3h)$ nodes of T_1 at level i whose children the procedure visits. Since the height of T_1 is at most h and the maximum degree of a node in T_1 is s, the lemma follows. □

Summing this bound over all pairs of canonical nodes, we obtain the following:

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

LastCommonEdge. Like in Intersects, the input pathlets ϕ_1 and ϕ_2 are given as pathlet-subtrees $T(\Phi_1, \phi_1)$ and $T(\Phi_2, \phi_2)$. Recall that we wish to find the *last* edge of ϕ_1 that appears in the intersection as e_1. We first find the last canonical node of $T(\Phi_1, \phi_1)$ that intersects any canonical node of $T(\Phi_2, \phi_2)$, by testing all pairs of canonical nodes with Intersects. Let $N_1 \in N(T(\Phi_1, \phi_1)$ be this node, and let $N_{21}, N_{22}, \ldots, N_{2t} \in N(T(\Phi_2, \phi_2))$ be the canonical nodes of $T(\Phi_2, \phi_2)$ where $\phi_{N_1} \cap \phi_{N_{2j}} \neq \varnothing$. If N_1 is a leaf node with matching edge e, then we return $e_1 = e$ as the spine to N_1 in Φ_1.

When N_1 is not a leaf, we recursively descend into an extreme child of N_1. Let $N^* \in \text{Ch}(N_1)$ be the last child of N_1 that intersects any of $N_{21}, N_{22}, \ldots, N_{2t}$, which we find by invoking Intersects (for a pair of canonical nodes) between all pairs in $\text{Ch}(N_1) \times \{N_{21}, N_{22}, \ldots, N_{2t}\}$. We prune the set $\{N_{21}, N_{22}, \ldots, N_{2t}\}$ to contain only nodes that intersect N^*, and recurse with $N_1 = N^*$ and the pruned subset of $\{N_{21}, N_{22}, \ldots, N_{2t}\}$.

Afterwards, to compute e_2 (the copy of e_1 in ϕ_2), we run the procedure for finding e_1 with first argument ϕ_2 and second argument $e_1 \blacktriangleright \phi_1 \blacktriangleleft e_1$, i.e., to find the last edge of ϕ_2 that intersects e_1. After computing both 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 FTE 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* $O(sh)$*.*

PATHCOST and CYCLECOST. To compute the adjusted cost of a pathlet ϕ originating from Φ, represented as the spines of $T(\Phi, \phi)$, we simply add up the adjusted costs stored at each canonical node of $T(\Phi, \phi)$, plus the adjusted costs of non-matching edges between adjacent canonical pathlets. The total time spent is $O(sh)$. A similar process computes the cost of a pathlet sequence or cycle Γ.

Lemma 4.10. *Given a path or a cycle as sequence of at most* s *pathlets,* PATHCOST *and* CYCLECOST *run in time* $O(s^2h)$*.*

PATHLETREPORT. To report the alternating path represented by a pathlet ϕ originating from an FTE Φ, we recursively PATHLETREPORT each canonical node of $T(\Phi, \phi)$, concatenate the results in sequence, and add the non-matching edges between two canonical pathlets. The base case is when the input pathlet is a leaf, in which case we simply return the matching edge associated with the leaf. Given a path/cycle as a sequence of at most s pathlets, REPORT works by calling PATHLETREPORT for every pathlet in the sequence and adding the non-matching edge between two consecutive pathlets. If the input is a cycle, then REPORT also adds a non-matching edge from the last matching edge to the first one.

Lemma 4.11. *(i)* PATHLETREPORT *runs in* $O((s+k)h)$ *time where* k *is the output size. (ii) Given a path or a cycle as sequence of at most* s *pathlets,* REPORT *run in time* $O((s^2+k)h)$ *where* k *is the output size.*

Combining Lemmas 4.8–4.10, we obtain the following:

Corollary 4.12. CONSTRUCTFTE *and* SIMPLEREDUCINGSUBCYCLE *take* $O(s^{10}h^4)$ *time.*

Updating intersection tables. After we have computed FTEs of all affected cells at a level i, we update the intersection tables to add the entries for canonical pathlets of rank i. Let Φ be an FTE 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 FTE Φ_{η_1}. Let $\Delta_1 = \boxplus(\Phi_{\eta_1})$. 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). It can be verified that the entries of all intersection-table look-ups performed by the INTERSECTS procedure already have been computed. Since there are $O(1)$ sibling cells of Δ_1, each of them has $O(s^2)$ matching/internal arcs, and there are $O(s^3h)$ canonical pathlets originating from each such arc, the total number of entries computed for φ_1 is $O(s^5h)$. Φ has $O(sh)$ canonical pathlets, and $\boxplus$ has $O(s^2)$ FTEs, so the total number entries computed for $\boxplus$ is $O(s^8h^2)$. Using Corollary 4.7, we obtain the following:

Lemma 4.13. *The total time spent in updating the intersection tables because of the new canonical pathlets generated in the computation of FTEs at a cell is* $O(s^{14}h^6)$*.*

Putting everything together, we obtain the following:

Lemma 4.14. *For any cell* $\boxplus$*, the total time spent in computing FTEs and updating intersection tables is* $(\varepsilon^{-1}\log n)^{O(d)}$*.*

Table of Notation

Notation	Description	Page List
M_{opt}	Minimum-weight perfect matching	1
$\vec{G}_M$	Residual graph of G with respect to M	3
$\bar{\text{¢}}_M$	Net cost with respect to ¢ in $\vec{G}_M$	3
w_0	Coarse approximation to cost of optimal matching	4
β	Constant approximation to cost of optimal matching	5
$\alpha_{\theta,M}(e)$	θ-adjusted cost of edge e with respect to M	6
$\bar{\varepsilon}$	Upper ε	6
$\underline{\varepsilon}$	Lower ε	6
$\alpha^*_{\bar{\varepsilon},M}$	Minimum $\bar{\varepsilon}$-adjusted cost among all augmenting paths in $\vec{G}_M$	6
M_{alg}	Perfect matching returned by our algorithm	6
ℓ_i	Side-length of cells at level i	10
$\mathfrak{C}_i$	Collection of cells at level i	10
$\boxplus$	An arbitrary cell	10
$\text{Ch}(\boxplus)$	Set of children cells of $\boxplus$	10
$\text{Sb}(\boxplus)$	Set of sibling cells of $\boxplus$	10
h	Height of the hierarchical covering	11
δ	Penalty; linear to $\underline{\varepsilon}$	11
$\mathscr{C}_i$	Collection of nonempty cells at level i	11
$\mathscr{C}$	Collection of all nonempty cells	11
$G_\boxplus$	Compressed graph at cell $\boxplus$	11
τ	Level offset between cells and subcells	11
$\text{Ch}(\square)$	Children subcells of $\square$	11
$X_\boxplus$	Set of subcells of $\boxplus$	11
$A_\square$	Points of A in subcell $\square$	11
$B_\square$	Points of B in subcell $\square$	11
$\mathscr{A}_\boxplus$	Collection of nonempty $A_\square$s in cell $\boxplus$	11
$\mathscr{B}_\boxplus$	Collection of nonempty $B_\square$s in cell $\boxplus$	11
$V_\boxplus$	Vertices of compressed graph $G_\boxplus$; all clusters of $\boxplus$	11
$E_\boxplus$	Arcs of compressed graph $G_\boxplus$	12
$M_{\square,\square'}$	Matching edges between subcells $\square$ and $\square'$	12
$\pi_\boxplus(X,Y)$	Minimum weight path between X and Y in $G_\boxplus$	12
$\Phi_\boxplus(A_\square,B_{\square'})$	Expansion of $\pi_\boxplus(X,Y)$: a simple alternating path between $A_\square$ and $B_{\square'}$	12
$w_\boxplus$	Arc weights on $G_\boxplus$	13
$\Pi_{\square,\square'}$	Flex-tip expansion of $\pi_\boxplus(A_\square,B_{\square'})$ fixing two tips	14
$(\square,\square')^*_\boxplus$	Subcell pair that realizes minimum $\underline{\varepsilon}$-adjusted cost of $\Pi_{\square,\square'}$ in $\vec{G}_M$ among all pairs of unsaturated $A_\square$ and $B_{\square'}$	14
OptPairs	Set of $(\square,\square')^*_\boxplus$ over all pairs $A_\square, B_{\square'}$ of $\boxplus$	14
g_j	Longest matching edge between $B_{\square_j}$ and $A_{\square_{j+1}}$ in intermediate expansion	15
η_j	non-matching arc in intermediate expansion	15
$\mathscr{C}_\Pi$	Cells affected by Π	15
s	Number of clusters in cell $\boxplus$; $\lvert V_\boxplus \rvert$	23

Notation	Description	Page List
$e_i \blacktriangleright \Phi \blacktriangleleft e_j$	Inclusive splice operators	23
$e_i \triangleright \Phi \triangleleft e_j$	Exclusive splice operators	23
$T(\Phi)$	FTE-tree of Φ	26
$T(\Phi, \phi)$	Pathlet-subtree of ϕ in $T(\Phi)$	26
$T(e)$	Single-node FTE-tree of e	27
$N(T(\Phi, \phi))$	Sequence of canonical nodes of $T(\Phi, \phi)$	27
$\boxplus(u)$	Cell of internal node u of $T(\Phi)$	28
level(u)	Level of internal node u of $T(\Phi)$	28
rank(ϕ)	Smallest level of an FTE whose FTE-tree has ϕ appear in as a canonical pathlet	28
$\mathscr{P}(\Phi)$	Set of canonical pathlets originate from Φ	29
β_{η_1,η_2}	Intersection table for all $(\phi_1, \phi_2) \in \mathscr{P}(\Phi(\eta_1)) \times \mathscr{P}(\Phi(\eta_2))$	29

References

[1] P. Agarwal and K. Varadarajan. A near-linear constant-factor approximation for Euclidean bipartite matching? In *Proceedings of the twentieth annual symposium on Computational geometry*, page 247, 2004.

[2] P. K. Agarwal, H.-C. Chang, and A. Xiao. Efficient Algorithms for Geometric Partial Matching. In G. Barequet and Y. Wang, editors, *35th International Symposium on Computational Geometry (SoCG 2019)*, volume 129 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[3] P. K. Agarwal, A. Efrat, and M. Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM Journal on Computing; Philadelphia*, 29(3):42, 2000.

[4] P. K. Agarwal, K. Fox, D. Panigrahi, K. R. Varadarajan, and A. Xiao. Faster algorithms for the geometric transportation problem. In *Proc. 33rd International Symposium on Computational Geometry*, pages 7:1–7:16, 2017.

[5] P. K. Agarwal and R. Sharathkumar. Approximation algorithms for bipartite matching with metric and geometric costs. In *Proc. ACM Symposium on Theory of Computing*, pages 555–564, 2014.

[6] A. Andoni, A. Nikolov, K. Onak, and G. Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 574–583, New York New York, May 2014. ACM.

[7] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *Proc. 34th International Conference on Machine Learning*, pages 214–223, 2017.

[8] S. Ashur and S. Har-Peled. On Undecided LP, Clustering and Active Learning. In K. Buchin and É. Colin de Verdière, editors, *37th International Symposium on Computational Geometry (SoCG 2021)*, volume 189 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:15, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[9] J. v. d. Brand, Y. T. Lee, Y. P. Liu, T. Saranurak, A. Sidford, Z. Song, and D. Wang. Minimum cost flows, mdps, and ℓ_1-regression in nearly linear time for dense instances. *arXiv e-prints*, 2101:arXiv:2101.05719, Jan. 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 Annual Symposium on Foundations of Computer Science*, pages 919–930, 2020.

[11] P. B. Callahan and S. R. Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In *Proceedings of the fourth annual ACM-SIAM symposium on Discrete algorithms*, SODA '93, pages 291–300, USA, Jan. 1993. Society for Industrial and Applied Mathematics.

[12] T. M. Chan. Approximate nearest neighbor queries revisited. *Discret. Comput. Geom.*, 20(3):359–373, 1998.

[13] T. M. Chan, S. Har-Peled, and M. Jones. On locality-sensitive orderings and their applications. In *10th Innovations in Theoretical Computer Science Conference*, volume 124, pages 21:1–21:17, 2019.

[14] R. Duan and S. Pettie. Approximating maximum weight matching in near-linear time. In *Proc. 51st Annual IEEE Sympos. on Foundat. Comput. Sc.*, pages 673–682, 2010.

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

[16] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18(5):1013–1036, Oct. 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] S. Har-Peled. *Geometric Approximation Algorithms*. American Mathematical Soc., 2011.

[19] J. Hopcroft and R. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[20] P. Indyk. A near linear time constant factor approximation for Euclidean bichromatic matching (cost). In *SODA 2007*, page 4, 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. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2495–2504. Society for Industrial and Applied Mathematics, Jan. 2017.

[22] T. Kathuria, Y. P. Liu, and A. Sidford. Unit capacity maxflow in almost $o(m^4/3)$ time. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 119–130, Nov. 2020.

[23] A. B. Khesin, A. Nikolov, and D. Paramonov. Preconditioning for the geometric transportation problem. In *Proc. 35th Annual Symposium on Computational Geometry*, pages 15:1–15:14, 2019.

[24] N. Lahn and S. Raghvendra. An $\tilde{O}(n^{5/4})$ time ε-approximation algorithm for rms matching in a plane. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, pages 869–888. Society for Industrial and Applied Mathematics, Jan. 2021.

[25] H. Le and S. Solomon. Truly optimal Euclidean spanners. In D. Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science*, pages 1078–1100, 2019.

[26] A. Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *Foundations of Computer Science (FOCS)*, pages 253–262. IEEE, 2013.

[27] S. Micali and V. V. Vazirani. An $O(\sqrt{|v|} \cdot |E|)$ algorithm for finding maximum matching in general graphs. In *FOCS*, pages 17–27, 1980.

[28] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., 1982.

[29] S. Raghvendra and P. K. Agarwal. A near-linear time ε-approximation algorithm for geometric bipartite matching. *Journal of the 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. *International Journal of Computer 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 Annual ACM Symposium on Computational Geometry*, pages 9–16, 2013.

[32] R. Sharathkumar and P. K. Agarwal. Algorithms for transportation problem in geometric settings. In *Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, 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 Transactions on Graphics*, 34(4):66, 2015.

[34] P. M. Vaidya. Geometry helps in matching. *SIAM J. Comput.*, 18:1201–1225, December 1989.

[35] K. R. Varadarajan. A divide-and-conquer algorithm for min-cost perfect matching in the plane. In *Proc. 39th Annual IEEE Symposium on Foundations of Computer Science*, pages 320–331, 1998.

[36] K. R. Varadarajan and P. K. Agarwal. Approximation algorithms for bipartite and non-bipartite matching in the plane. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 17-19 January 1999, Baltimore, Maryland, USA.*, pages 805–814, 1999.