An Improved ε-Approximation Algorithm for Geometric Bipartite Matching

Pankaj K. Agarwal ✉
Duke University

Sharath Raghvendra ✉
Virginia Tech

Pouyan Shirzadian ✉
Virginia Tech

Rachita Sowle ✉
Virginia Tech

───── **Abstract** ─────

For two point sets $A, B \subset \mathbb{R}^d$, with $|A| = |B| = n$ and $d > 1$ a constant, and for a parameter $\varepsilon > 0$, we present a randomized algorithm that, with probability at least $1/2$, computes in $O(n(\varepsilon^{-O(d^3)} \log \log n + \varepsilon^{-O(d)} \log^4 n \log^5 \log n))$ time, an ε-approximate minimum-cost perfect matching under any L_p-metric. All previous algorithms take $n(\varepsilon^{-1} \log n)^{\Omega(d)}$ time. We use a randomly-shifted tree, with a polynomial branching factor and $O(\log \log n)$ height, to define a tree-based distance function that ε-approximates the L_p metric as well as to compute the matching hierarchically. Then, we apply the primal-dual framework on a compressed representation of the residual graph to obtain an efficient implementation of the Hungarian-search and augment operations.

1 Introduction

Let $A, B \subset \mathbb{R}^d$ be two point sets of size n each, where $d > 1$ is a constant, and $\mathsf{d}(\cdot, \cdot)$ a metric. Let $\mathcal{G} = (A \cup B, A \times B)$ be a weighted complete bipartite graph in which the cost of an edge $(a, b) \in A \times B$ is $\mathsf{d}(a, b)$. A *matching* in $\mathcal{G}$ is a set of vertex-disjoint edges in $\mathcal{G}$. A *perfect matching* in $\mathcal{G}$ is a matching of size n. The cost of a matching M, denoted by $\mathfrak{c}(M)$, is $\mathfrak{c}(M) = \sum_{(a,b) \in M} \mathsf{d}(a, b)$. The *minimum-cost perfect matching* in $\mathcal{G}$, denoted by M^*, is a perfect matching in $\mathcal{G}$ of the minimum cost. For any $\varepsilon > 0$, a perfect matching M in $\mathcal{G}$ is called an *ε-approximate matching* if $\mathfrak{c}(M) \leq (1 + \varepsilon)\mathfrak{c}(M^*)$. We consider the case where the cost $\mathsf{d}(a, b)$ is the L_p distance denoted by $\|a - b\|_p$. The optimal transport (OT) distance between two (possibly continuous) distributions can be estimated by taking n samples from both distributions and then computing their minimum-cost perfect matching. The wide applicability of OT in Machine Learning and Computer Vision [17, 20, 4] has motivated the design of fast exact and approximation algorithms that compute a minimum-cost perfect matching. In this paper, for the L_p-norm, we present a new algorithm to computing an ε-approximate matching.

Related work. For an arbitrary weighted bipartite graph with n vertices and m edges, the Kuhn-Munkres algorithm [13] computes a minimum-weight bipartite matching in a weighted bipartite graph in $O(mn + n^2 \log n)$ time. For bipartite graphs with non-negative integer costs bounded by C, Gabow and Tarjan [9] gave an $O(m\sqrt{n} \log(nC))$-time algorithm. Both

the Hungarian and Gabow-Tarjan algorithms are combinatorial algorithms that iteratively find an augmenting path and augment the matching along this path. The augmenting paths are chosen such that the increase in the matching cost after each augmentation is minimized. This cost increase is referred to as the *net-cost* of the path. Alternate approaches such as the electrical flow [21] method and the matrix multiplication based methods [15] can be used to obtain fast matching algorithms. The current best known execution time is $\tilde{O}(m + n^{1.5})$.

When $A \cup B$ is a 2-dimensional point set in the Euclidean space, a Euclidean minimum-weight matching (EMWM) can be computed in $O(n^2 \operatorname{polylog} n)$ time [12], and in $O(n^{3/2} \operatorname{polylog} n)$ time when the points have integer coordinates [19, 18]. For this case, it is easy to compute a $O(\log n)$-approximate matching in expectation using a randomly shifted quad-tree [6, 7]. Agarwal and Varadarajan [1] used the shifted quad-tree to compute an $O(\log 1/\delta)$-approximate solution in $O(n^{1+\delta})$ time. Following this, there were several results that used such a decomposition; see for instance [11, 3, 8]. The current best-known approximation algorithm for computing EMWM is by Raghvendra and Agarwal [16], which computes an ε-approximate matching with high probability in $n(\varepsilon^{-1} \log n)^{O(d)}$ time. In their algorithm, each cell $\square$ of a randomly shifted quad-tree Q is decomposed by a uniform grid into $(\log n/\varepsilon)^{O(d)}$ subcells. The Euclidean distance between any pair of points u, v with $\square$ as their least common ancestor in Q is ε-approximated by the distance between the subcells of $\square$ that contain u and v respectively. Their algorithm uses Q to compute a minimum net-cost augmenting path P with respect to the new distance and augment the matching along this path, both in time $O(|P| \operatorname{poly} \log n)$. They obtain a near-linear execution time by bounding the total length of all augmenting paths by $O(\varepsilon^{-1} n \log n)$. To compute these paths quickly, they compress the residual graph inside $\square$ into a graph of $(\log n/\varepsilon)^{O(d)}$ size and execute Bellman-Ford algorithm on this graph. Lahn and Raghvendra [14] extended this framework to approximate the 2-Wasserstein distance of planar point sets, i.e., an approximate minimum-cost matching when $\mathsf{d}(u, v)$ is $\|u - v\|_2^2$. Unlike Euclidean distance, approximating the squared-Euclidean distance using Q results in a polynomial sized compressed residual graph at each cell. Since using Bellman-Ford algorithm on such a compressed graph can be prohibitively expensive, they introduce a novel primal-dual framework and define *compressed feasibility* on the compressed residual graph. Using this framework, they are able to find an augmenting path as well as augment it along this path in sub-linear time. Consequently, they achieve an $O(n^{5/4} \operatorname{poly}(\log n, 1/\varepsilon))$ time algorithm for the 2-Wasserstein distance between planar point sets. Recently, Agarwal *et al.* [2] have designed a deterministic algorithm that uses multiple quadtrees to compute a $(1 + \varepsilon)$-approximate Euclidean matching in $n(\varepsilon^{-1} \log n)^{O(d)}$ time.

Our result.   The following theorem states our main result.

▶ **Theorem 1.** *Let A, B be two point sets in $\mathbb{R}^d$ of size n each, for a constant $d > 1$, and let $0 < \varepsilon \leq 1$ be a parameter. With probability at least $1/2$, an ε-approximate matching under any L_p-metric can be computed in $O(n(\varepsilon^{-O(d^3)} \log \log n + \varepsilon^{-O(d)} \log^4 n \log^5 \log n))$ time.*

For the sake of simplicity, we describe the algorithm for the Euclidean metric. It can be extended to other L_p-metrics in a straight forward manner. For any two points a and b, we use $\|a - b\|$ to denote the Euclidean distance between them. Using standard techniques [16, 14], we can preprocess the input points in $O(n \log n)$ time so that the point sets A and B satisfy the following conditions: (P1) All input points have integer coordinates bounded by $n^{O(1)}$. (P2) No integer grid point contains points of both A and B. (P3) $\mathsf{c}(M^*) \in \left[\frac{3\sqrt{d}n}{\varepsilon}, \frac{9\sqrt{d}n}{\varepsilon} \right]$. Details of how we preprocess A and B can be found in the Appendix (Section A). Assuming A and

B satisfy (P1)–(P3), we present an algorithm that, with probability $1/2$, computes an $(\varepsilon/2)$-approximate matching in $O(n(\varepsilon^{-O(d^3)} + \varepsilon^{-O(d)} \log^4 n \log^4 \log n))$ time. The preprocessing step adds an additional $\log \log n$ factor to the running time of the algorithm, resulting in the running time mentioned in Theorem 1. In the following, we provide an overview of our approach and its comparison with existing work.

As in [16, 14], we also define a tree based distance $\mathsf{d}_\mathsf{T}(\cdot, \cdot)$ that approximates the Euclidean distance (Section 2). Unlike [16, 14] that use a quad-tree of height $O(\log n)$, we build a tree T of height $O(\log \log n)$ (see Section 2.1). Each cell of T at level i (root is assigned level 0) with a side length of ℓ_i is partitioned using a randomly-shifted grid into children whose side-length $\ell_{i+1} = \ell_i^c$ where $c < 1$ is a constant that depends only on d. Given that the point set have integer coordinates bounded by $n^{O(1)}$ (from (P1)), the height of T is $h = O(\log \log n)$. For any pair of points (u, v) with a cell $\square$ of level i as its least common ancestor, let $\square_u$ and $\square_v$ be the children of $\square$ that contain u and v respectively. As in the case of a randomly shifted-quadtree where we get an $O(h) = O(\log n)$ approximation, one can show that the distance between the centers of $\square_u$ and $\square_v$ is a $O(h) = O(\log \log n)$ approximation of the Euclidean distance (in expectation). We obtain a refined $(1 + \varepsilon)$-approximation of the Euclidean distance by partitioning $\square_u$ and $\square_v$ into finer subcells and then using the distance between the centers of those sub-cells that contain u and v. As in [16, 14], one can divide each cell into $O(h^d)$ many subcells and obtain a $(1+\varepsilon)$-approximation of the Euclidean distance. With $h = \log \log n$, this will result in an execution time of $\Omega(n \log^4 n(\varepsilon^{-1} \log \log n)^{d^3})$. Instead, we partition a cell into subcells more carefully (See the definition of subcells in Section 2.2). Intuitively, we make the number of subcells a function of the height of the cell, i.e., smaller cells have significantly fewer than $\log^{O(d)} \log n$ subcells. As a result, we are able to improve the dependence of our algorithm from $\log^{O(d^3)} \log n$ to $\log^{O(d)} \log n$. Interestingly, we show that the expected distortion is higher for cells that are closer to the leaves. Nonetheless, we are able to bound the expected error of our distance between any two points u and v by $\varepsilon \|u - v\|$ (See Lemma 3).

Similar to [14], our algorithm compactly stores the residual graph (Section 5.2) as well as the dual weights (Section 5.3) and uses this compact representation to efficiently find augmenting paths. The size of the compressed residual graph inside any cell is bounded by the side-length of its child, i.e., smaller cells have a smaller compressed graph (Lemma 14). As a result, finding augmenting paths in smaller cells is significantly faster than that in larger ones. In our analysis, we show that most of the augmenting paths in the algorithm are found in smaller cells which can be computed quickly. In particular, only $O(\frac{n}{\varepsilon \ell_{i+1}})$ augmenting paths are found inside a compressed graph at level i, each of which can be found in $O(\ell_{i+1} \log^2 n)$ time. Combining across all $O(\log \log n)$ levels, we get a near-linear execution time.

Typical matching algorithms that are based on a compressed residual graph modify the dual weights and find an augmenting path with respect to current matching M. The algorithms presented in [16, 14] classify edges into *local* and *non-local* which they use critically in computing a minimum net-cost augmenting path. We remove the need for this classification and make our algorithm and its analysis simpler. Instead of using the classification, our algorithm carefully updates the dual weights, possibly modifies a matching M to another matching M' of the same size and cost, and finds an augmenting path with respect to the new matching M'.

   Hierarchical Partition and the Distance Function

In this section, we present a randomized hierarchical partitioning of space, used to define a new distance function $\mathsf{d}_\mathsf{T}(\cdot, \cdot)$ that approximates the Euclidean metric (in expected sense) as well as to guide the construction of matching (in a hierarchical manner).

2.1   Hierarchical Partitioning

For a value $\ell > 0$, let $\mathsf{G}[\ell]$ be the d-dimensional uniform grid with cell side-length ℓ, i.e., $\mathsf{G}[\ell] = (\ell\mathbb{Z})^d + [0, \ell]^d$. For a point $x \in \mathbb{R}^d$, we use $\mathsf{G}[\ell] + x$ to denote the translate of $\mathsf{G}[\ell]$ by x. For any rectangle R, let $A_\mathsf{R} = A \cap \mathsf{R}$ and $B_\mathsf{R} = B \cap \mathsf{R}$. We say that R is *non-empty* if $A_\mathsf{R} \cup B_\mathsf{R} \neq \emptyset$. Given R and a grid G, let $\mathsf{C}[\mathsf{R}, \mathsf{G}]$ denote the set of non-empty rectangles in the rectangular subdivision of R induced by G. If G is fixed or clear from the context, we use $\mathsf{C}[\mathsf{R}]$ to denote $\mathsf{C}[\mathsf{R}, \mathsf{G}]$. By abusing the notation slightly, we use $\mathsf{C}[\mathsf{R}]$ to denote the subdivision as well as the set of non-empty rectangles in the subdivision. For a non-empty rectangle R, we designate one of the points in $A_\mathsf{R} \cup B_\mathsf{R}$, say r_R, as its *representative*.

Let $\square_0$ be the smallest axis-aligned hypercube that contains $A \cup B$. Let ℓ_0 be its side length. By property (P1), $\ell_0 = n^{O(1)}$. We construct a hierarchical partition and the associated tree T, as follows. Each node in T is associated with a non-empty rectangle which we refer to as a *cell* and we will not distinguish between the two. The *level* of a node (and the corresponding rectangle) is the length of the path in T from the root. The root of T is $\square_0$ and its level is 0. Set $\alpha = 1 - \frac{1}{8d+2}$. For $i > 0$, set $\ell_i = \ell_{i-1}^\alpha$. Let $h > 0$ be the smallest integer such that $\ell_h \leq (\varepsilon^{-1}d)^{\frac{\alpha}{(1-\alpha)^2}}$. Any cell $\square$ in T of level h is designated as a leaf node. By construction, $h = O(\log \log n)$ The choice for the condition of the leaf node will become apparent in Section 2.2. Otherwise, we choose a random point, $\xi_\square \in [0, \ell_{i+1}]^d$ and set $\mathsf{G} = \mathsf{G}[\ell_{i+1}] + \xi_\square$. Let $\mathsf{C}[\square] := \mathsf{C}[\square, \mathsf{G}]$ be the subdivision of $\square$ induced by G. Each rectangle $\square' \in \mathsf{C}[\square]$ is a level $i + 1$ node. We create a child of $\square$ in T for each non-empty rectangle $\square' \in \mathsf{C}[\square]$ and recursively construct the partition and associated sub-tree of $\square'$. For any $0 \leq i \leq h - 1$, let $\Delta[i]$ denote all cells of T of level i.
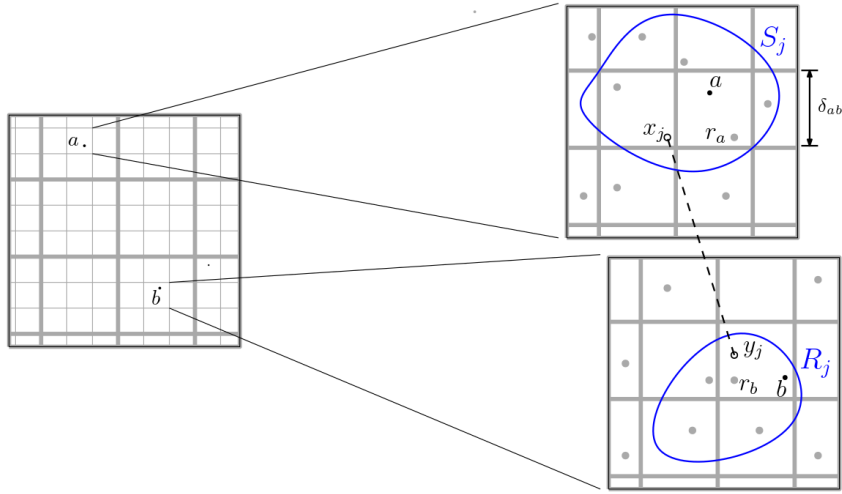
2.2   Euclidean Distance Approximation

For any pair of points $(a, b) \in A \times B$, let $\square$ be the least common ancestor of a and b in T, i.e., the cell with the highest level that contains a and b. Let the level of $\square$ be i. We define the *level* of (a, b), $\mathrm{lev}(a, b)$, to be i and refer to (a, b) as *a level i edge*. Let $\square_a$ and $\square_b$ be the children of $\square$ that contain a and b respectively. We divide $\square_a$ and $\square_b$ into $O(\varepsilon^{-d}(h-i)^{2d})$ *subcells* and show that the distance between any two points in the subcells that contain a and b is a $(1+\varepsilon)$-approximation of $\|a - b\|$. Note that the smaller cells, i.e., those at a higher level, have fewer subcells. Using this and the bound on the side-length of any leaf node of T, we can bound the number of subcells of the children of $\square$ by ℓ_{i+1} (independent of $\log \log n$). In Section 5, we use this fact to compress the residual graph more efficiently.

Subcells: Any cell $\square$ is divided into subcells as follows. For each $0 < i < h$, set

$$\mu_i = \frac{\varepsilon}{c_2\sqrt{d}(h-i)^2}\ell_i, \tag{1}$$

where $c_2 = 24\pi^2$. We set $\mu_0 = \infty$ and $\mu_h = 0$. A *subcell* is formed by combining a subset of children of $\square$ such that the diameter of points in these children is no more than μ_i. Set $\bar{\mu}_i = \left\lfloor \frac{\mu_i}{\sqrt{d}\ell_{i+1}} \right\rfloor \ell_{i+1}$ (By Lemma 2 below, $\left\lfloor \frac{\mu_i}{\sqrt{d}\ell_{i+1}} \right\rfloor$ is a positive integer). For any non-leaf

Figure 1 Euclidean distance approximation: The grey rectangular subdivision shows the children of $\square$ (on left) and the bold grey rectangular subdivision shows the subcells.

and non-root cell of level i in the tree T, let $\mathsf{G}_\square = \mathsf{G}[\bar{\mu}_i] + \xi_\square$, where $\xi_\square$ is the random shift for the grid constructed in $\square$. By construction, the boundary of grid cells in $\mathsf{G}_\square$ are aligned with those of $\mathsf{G}[\ell_{i+1}] + \xi_\square$ (See Figure 1). Therefore, each non-empty child cell of $\square$ lies in exactly one subcell of subdivision $\mathsf{C}[\square, \mathsf{G}_\square]$. Let $\mathsf{S}_\square$ be this set of non-empty subcells, and let $\mathsf{R}_\square$ be the set of representative points of $\mathsf{S}_\square$, i.e., $\mathsf{R}_\square = \{r_\mathsf{R} \mid \mathsf{R} \in \mathsf{S}_\square\}$. It is clear that the diameter of each subcell is at most μ_i. The following lemma relates the side-length of the children of a cell to the diameter of the subcells of that cell.

▶ **Lemma 2.** *For any $0 \le i < h$, $\ell_{i+1} \le \mu_i/\sqrt{d}$.*

WSPD: For any cell $\square$, the number of pairs of children subcells of $\square$ can be prohibitively large. We use a well-separated pair decomposition (WSPD) to compactly store these pairs. For simplicity of the algorithm, similar to [14], we use WSPD to define our Euclidean distance approximation. For a point set X, let $\textsc{Diam}_X$ be the distance between the farthest pair of points in X. For any $\varepsilon > 0$, two point sets X and Y are called ε-*well-separated* if for all $x \in X$ and $y \in Y$, $\max\{\textsc{Diam}_X, \textsc{Diam}_Y\} \le (\varepsilon/12)\|x - y\|$. Given a point set $X \in \mathbb{R}^d$ of size n and a parameter $\varepsilon > 0$, an ε-*WSPD* (or simply WSPD for brevity) of X is a set $\mathcal{W} = \{(R_1, S_1), \cdots, (R_k, S_k)\}$ such that (i) each (R_i, S_i) is ε-well separated, (ii) for each pair of points $(u, v) \in X \times X$, there is a unique pair $(R_i, S_i) \in \mathcal{W}$ such that $(u, v) \in R_i \times S_i$ or $(u, v) \in S_i \times R_i$, and (iii) $k = O(\varepsilon^{-d}n)$. Also, if the spread of X, the ratio of the largest to smallest pairwise distances, is bounded by $n^{O(1)}$, then every point of X participates in $O(\varepsilon^{-d} \log n)$ pairs of $\mathcal{W}$. $\mathcal{W}$ can be constructed in $O(n \log n + \varepsilon^{-d}n)$ time [5, 10]. For any $(R_i, S_i) \in \mathcal{W}$, we choose an arbitrary pair $(x_i, y_i) \in R_i \times S_i$ and make this pair its *representative pair*.

For any non-leaf cell $\square \in T$, let $X_\square = \bigcup_{\square' \in \mathsf{C}[\square]} \mathsf{R}_{\square'}$ be the set of representative points of all non-empty subcells of the children of $\square$. We build an ε-WSPD $\mathcal{W}_\square = \{(R_1, S_1), \ldots, (R_k, S_k)\}$ on $X_\square$. For a leaf cell $\square$, we construct an ε-WSPD $\mathcal{W}_\square$ on $A_\square \cup B_\square$.

Distance function: We are now ready to define the distance function $\mathsf{d}_\mathsf{T} : A \times B \mapsto \mathbb{R}_{\ge 0}$. For any pair of points $(a, b) \in A \times B$ of level i with $\square$ as its least common ancestor, if $i = h$,

we set $\delta_{ab} = 0$ and if $i < h$, we set $\delta_{ab} = \mu_{i+1}$. For $i = h$, we set (R_j, S_j) to be the pair in $\mathcal{W}_\square$ such that $(a, b) \in R_j \times S_j$. For $i < h$, (R_j, S_j) is defined as follows. Let $\square_a$ (resp. $\square_b$) be the child of $\square$ that contains a (resp. b), and let ξ^a_{ab} (resp. ξ^b_{ba}) be the subcell of $\square_a$ (resp. $\square_b$) that contains a (resp. b). Let r_a and r_b be the representatives of ξ^a_{ab} and ξ^b_{ba} respectively. Let (R_j, S_j) be the unique ε-well separated pair of $\mathcal{W}_\square$ such that $(r_a, r_b) \in R_j \times S_j$. We say that (a, b) is *covered* by (R_j, S_j). Now let (x_j, y_j) be the representative pair of (R_j, S_j) (See Figure 1). Then, we define

$$\mathsf{d}_\mathsf{T}(a, b) = (1 + \varepsilon/4)\|x_j - y_j\| + 2\delta_{ab}. \tag{2}$$

Unlike in [16, 14], we create fewer subcells for cells that are closer to the leaves, which results in a larger distortion for the edges within these cells. However, Lemmas 3 and 4 together establish that the expected distortion on any pair of points $(a, b) \in A \times B$ is still proportional to $\varepsilon\|a - b\|$. See Appendix B for the proof.

▶ **Lemma 3.** *For any pair of points $(a, b) \in A \times B$, $\mathbb{E}[\delta_{ab}] \leq \frac{\pi^2}{6c_2}\varepsilon\|a - b\|$.*

Using Lemma 3, we show in Lemma 4 that our distance function, in expectation, approximates the Euclidean distance within a factor of $(1 + \varepsilon)$.

▶ **Lemma 4.** *For any pair of points $(a, b) \in A \times B$, $\mathsf{d}_\mathsf{T}(a, b) \geq \|a - b\|$. Furthermore, $\mathbb{E}[\mathsf{d}_\mathsf{T}(a, b)] \leq (1 + (\frac{5\pi^2}{6c_2} + \frac{11}{24})\varepsilon)\|a - b\|$.*

3   Preliminaries

We begin by presenting notations pertaining to the distance function that will help us describe our algorithm. For any subset $E \subseteq A \times B$ of edges, we use $w(E) = \sum_{(u,v) \in E} \mathsf{d}_\mathsf{T}(u, v)$ to denote the sum of the weights of the edges in E with respect to $\mathsf{d}_\mathsf{T}(\cdot, \cdot)$.

Next, we describe definitions related to matching that will assist us in presenting our algorithm. Let M be any matching in $\mathcal{G}$. A vertex is *free* with respect to M if it has no edges of M incident on it. An *alternating* path with respect to M is a simple path in $\mathcal{G}$ whose edges alternate between those in M and not in M. An *augmenting* path is an alternating path whose endpoints are free. We can *augment* M along an augmenting path P by simply setting $M \leftarrow M \oplus P$. For any matched vertex $u \in A \cup B$, let $m(u)$ denote the vertex to which u is matched in M. For any edge (u, v), we define the *net-cost* $\phi(u, v)$ of (u, v) as follows: $\phi(u, v) = \mathsf{d}_\mathsf{T}(u, v) + \delta_{uv}$ if $(u, v) \notin M$, and $\phi(u, v) = -\mathsf{d}_\mathsf{T}(u, v)$ if $(u, v) \in M$. For a set $E \subseteq A \times B$ of edges, we define its *net-cost* as $\phi(E) = \sum_{(u,v) \in E} \phi(u, v)$.

A *residual graph* $\mathcal{G}_M$ is a directed bipartite graph that has the same set of edges as $\mathcal{G}$ and for any matching (resp. non-matching) edge $(a, b) \in A \times B$, it is directed from a to b (resp. b to a). We refer to the *weight* of (a, b) in $\mathcal{G}_M$ to be its net-cost. It is easy to see that any simple directed path P in $\mathcal{G}_M$ alternates between matching and non-matching edges and therefore, P is an alternating path. For any rectangle R, let M_R be the subset of the edges of M with both endpoints inside R, and let $\mathcal{G}^\mathsf{R}_M$ denote the residual graph on $A_\mathsf{R} \cup B_\mathsf{R}$ with respect to the matching M_R.

Similar to the Hungarian algorithm, our algorithm assigns a dual weight $y(v) \geq 0$ to each vertex $v \in A \cup B$. We say that a matching M and a set of non-negative dual weights $y(\cdot)$ are *feasible* if, for every directed edge (u, v) of $\mathcal{G}_M$, $y(u) - y(v) \leq \phi(u, v)$. The presence of δ_{uv} in the definition of $\phi(u, v)$ makes our feasibility conditions a relaxed form of the feasibility conditions of the Hungarian algorithm. It can be shown that a feasible perfect matching is (in expectation) a $(1 + \varepsilon/2)$-approximation of the minimum-cost Euclidean matching.

▶ **Lemma 5.** *Suppose a perfect matching M along with a set of dual weights $y(\cdot)$ are feasible and let M^* denote an optimal matching with respect to the Euclidean cost $\mathbb{c}(\cdot)$. Then, $\mathbb{E}[\mathbb{c}(M)] \leq \mathbb{E}[w(M)] \leq (1 + \varepsilon/2)\mathbb{c}(M^*)$.*

For any directed edge (u, v) of $\mathcal{G}_M$, we define its *slack* as $s(u, v) = \phi(u, v) - y(u) + y(v)$. Based on the definition of feasibility, it is clear that $s(u, v) \geq 0$. An edge (u, v) of $\mathcal{G}_M$ is called *admissible* if $s(u, v) = 0$. Given a feasible matching M, we can use the definition of slack to relate the weight of any directed path P from u to v in $\mathcal{G}_M$ to the slack on its edges:

$$\phi(P) = \sum_{(u', v') \in P} (y(u') - y(v') + s(u', v')) = y(u) - y(v) + \sum_{(u', v') \in P} s(u', v'). \qquad (3)$$

We present a slow yet simple implementation to find a $(1 + \varepsilon)$-approximate matching, which is basically the Hungarian algorithm. Initialize for every $v \in A \cup B$, $y(v) = 0$ and $M \leftarrow \emptyset$. At each step, we find an augmenting path in the residual graph as follows. We set the edge weights in the residual graph to be their slacks. Next, starting from the free vertices of B, we execute the Dijkstra's shortest-path algorithm (also called the *Hungarian Search*) in this graph. For any $v \in A \cup B$, let κ_v be the shortest path from any free vertex of B to v. The algorithm returns the augmenting path P to a free vertex a of A that minimizes κ_a; i.e, the minimum net-cost augmenting path. We update the dual weight of every vertex v with $\kappa_v < \kappa_a$ by setting $y(v) \leftarrow y(v) - \kappa_v + \kappa_a$, making all edges of P admissible. Finally, we set $M = M \oplus P$. Augmenting the matching along P keeps the matching feasible. In the following lemma, we state two observations of this algorithm.

▶ **Lemma 6.** *During the execution of the algorithm described above,*

(i) *augmenting paths are computed in increasing order of their net-costs; and*

(ii) *if the net-cost of an augmenting path P is less than μ_i, then P does not contain any edge of level lower than i (Recollect that any such edge has a cost of at least μ_i).*

Lemma 6 suggests that we can search for augmenting paths in residual graph inside the cells of $\Delta[i]$ until the net-cost of the augmenting path reaches μ_i.

The implementation described above requires n executions of Dijkstra's algorithm, each taking $\Theta(n^2)$ time. In the next two sections, we use the properties of this algorithm (Lemma 6) to present an efficient implementation of a variant of the above algorithm.

4   Overview of the Algorithm

We present our algorithm assuming the existence of three procedures, namely, BUILD, HUNGARIANSEARCH, and AUGMENT procedures. The details of these procedures is deferred to Section 5.

Our algorithm computes a feasible perfect matching by processing the cells of T in decreasing order of their levels. Initially $i \leftarrow h - 1$ and $M \leftarrow \emptyset$ (no matching is computed at level h). For any cell $\square \in \Delta[i]$, the algorithm executes the following steps:

- If $i < h - 1$, the algorithm calls the BUILD($\square, M$) procedure. This step builds a compact representation of the residual graph (defined in Section 5.2).
- The algorithm does the following iteratively: It calls the HUNGARIANSEARCH($\square, M$) procedure. This procedure returns NULL if there is no augmenting path in $\mathcal{G}_M^\square$ of a net-cost less than μ_i (see (1)). In this case, the algorithm stops processing $\square$. Otherwise, if there is an augmenting path, the procedure updates the dual weights $y(\cdot)$ and may update $M \leftarrow M'$ where M' is another matching of equal size such that $y(\cdot), M'$ is feasible

and $w(M) = w(M')$. Then it returns a minimum net-cost augmenting path P with respect to the updated matching. The algorithm calls $\text{AUGMENT}(\square, P, M)$ to augment M along P.

After all cells in $\Delta[i]$ are processed, if $i = 0$, the algorithm returns the matching M. Otherwise, it sets $i \leftarrow i - 1$ and continues.

Execution time of the procedures: The BUILD, HUNGARIANSEARCH, and AUGMENT procedures presented in Section 5 have the following execution time. For any cell $\square$, let $n_\square = |A_\square \cup B_\square|$. If $\square$ has a level $i < h - 1$, the BUILD procedure takes $n_\square \log^3 n (\varepsilon^{-1} \log \log n)^{O(d)}$ time. Next, we present the running time of HUNGARIANSEARCH and AUGMENT procedures.

For any cell $\square$, if $\square$ is at level $h - 1$ of T, then the HUNGARIANSEARCH and AUGMENT procedures takes $\varepsilon^{-O(d^3)}$ time. Otherwise, let $i = \text{lev}(\square) < h - 1$. For $j > i$, let k_j be the number of level j cells that contains at least one vertex of P. If HUNGARIANSEARCH returns NULL, $k_j = 0$. HUNGARIANSEARCH takes

$$O(\mu_{i+1}\varepsilon^{-O(d)} \log^3 n \log^2 \log n + \sum_{j=i+1}^{h-1} k_j \mu_{j+1} \varepsilon^{-O(d)} \log^3 n \log^3 \log n)$$

time and the total time taken by AUGMENT is $O(\sum_{j=i+1}^{h-1} k_j \mu_{j+1} \varepsilon^{-O(d)} \log^3 n \log^3 \log n)$.

Invariants: In Section 5, we also show that the three procedures maintain the following two invariants while processing cells at level i. At any point, for the matching M, there is a set of dual weights $y(\cdot)$ such that

(J1) $M, y(\cdot)$ is feasible, and,

(J2)   For any vertex $u \in B$, $y(u) \leq \mu_i$. Furthermore, if u is a free vertex of B, its dual weight $y(u) \geq \mu_{i+1}$. If u is a free vertex of A, its dual weight $y(u) = 0$.

Our procedures will not maintain dual weights $y(\cdot)$ explicitly but only guarantee the existence of dual weights that satisfy (J1) and (J2). From (J1), (J2), and (3), we get the following:

▶ **Corollary 7.** *For any $i \geq 0$, while the algorithm is processing level i cells, the net-cost of any augmenting path in $\mathcal{G}_M$ is at least μ_{i+1}.*

4.1   Analysis of the algorithm

Note that at the root cell $\square_0$, $\mu_0 = \infty$ and therefore, the second step of the algorithm terminates only when there are no more augmenting paths in $\mathcal{G}_M$; i.e, M is prefect. Since $M, y(\cdot)$ is also feasible, from Lemma 5, it is (in expectation) a $(1 + \varepsilon)$-approximate matching. Let W be the cost of the matching returned by our algorithm. From (P3) and the fact that $\varepsilon \leq 1$, we get that, with probability at least $1/2$,

$$W = \Theta(n/\varepsilon). \tag{4}$$

Next, using the execution time of the procedures and the invariants they maintain, we bound the execution time of our algorithm.

We introduce a few notation that helps us analyze our algorithm. Recollect that $n_\square = |A_\square \cup B_\square|$ and M^* denotes the optimal matching of $A \cup B$ with respect to the Euclidean costs. Let $\mathbb{P} = \langle P_1, \ldots, P_n \rangle$ be the sequence of augmenting paths computed by the algorithm in the order in which they were computed. Let $M_0 = \emptyset$ and let M_i be the matching after augmenting along the path P_i.

Efficiency analysis: We begin by bounding the time taken by BUILD($\square, M$) across all $O(\log \log n)$ levels of T by

$$\sum_{i=0}^{h-1} \sum_{\square \in \Delta[i]} n_\square \log^3 n_\square (\varepsilon^{-1} \log \log n)^{O(d)} = O(n \log^3 n (\varepsilon^{-1} \log \log n)^{O(d)}). \qquad (5)$$

This equality follows from the fact that $\sum_{\square \in \Delta[i]} n_\square = n$ and $h = O(\log \log n)$. Next, we bound the execution time of HUNGARIANSEARCH and AUGMENT procedures. For any cell $\square$ of level $h - 1$, the HUNGARIANSEARCH and AUGMENT procedures take $(1/\varepsilon)^{O(d^3)}$ time. The total time taken across all level $h - 1$ cells is $n/\varepsilon^{O(d^3)}$. Next, we bound the running time for cell at levels less than $h - 1$.

▶ **Lemma 8.** *For $i < h$, the number of free vertices after processing level i cells is $O\left(\frac{W}{\mu_i}\right)$.*

Therefore, there are $O(\frac{W}{\mu_{i+1}})$ executions of HUNGARIANSEARCH on level i cells. The time taken by all HUNGARIANSEARCH executions that return a NULL is at most

$$\left(\frac{W}{\mu_{i+1}}\right) \times O(\mu_{i+1} \varepsilon^{-O(d)} \log^3 n \log^2 \log n) = O(W \varepsilon^{-O(d)} \log^3 n \log^2 \log n).$$

Otherwise, the HUNGARIANSEARCH procedure returns a minimum net-cost augmenting path P and the AUGMENT procedure augments the matching along P. The time taken by each such execution of HUNGARIANSEARCH and AUGMENT procedure is

$$O(\mu_{i+1} \varepsilon^{-O(d)} \log^3 n \log^2 \log n + \sum_{j=i+1}^{h} k_j \mu_{j+1} \varepsilon^{-O(d)} \log^3 n \log^3 \log n)$$

$$= O(\mu_{i+1} \varepsilon^{-O(d)} \log^3 n \log^4 \log n + \sum_{j=i+1}^{h} (k_j - 1) \mu_{j+1} \varepsilon^{-O(d)} \log^3 n \log^3 \log n). \qquad (6)$$

The equality follows from the fact that $\mu_{i+1} > \mu_{j+1}$. While processing $\Delta[i]$, $O(W/\mu_{i+1})$ augmenting paths are found (see Lemma 8), so the first term in the RHS of (6) is $O(W \varepsilon^{-O(d)} \log^3 n \log^5 \log n)$ over all augmenting paths. Next, we bound the second term over all augmenting paths.

Any augmenting path P has at least $k_j - 1$ edges of level at most $j - 1$. Furthermore, for any $j' \geq j$, every level $j - 1$ edge on P will contribute at most two new cells to $k_{j'}$. Suppose γ_j is the number of level $j - 1$ edges across all augmenting paths. The second term of the RHS of (6), when summed across all augmenting paths, can be written as $O(\sum_{j=1}^{h} \gamma_j \mu_{j+1} \varepsilon^{-O(d)} \log^3 n \log^3 \log n)$. Lemma 9 (See Appendix B for a proof) establish that $\gamma_j = O\left(\frac{W \log n}{\mu_{j+1}}\right)$.

▶ **Lemma 9.** *For any $1 \leq j \leq h - 1$, $\gamma_j = O(W \log n/\mu_{j+1})$.*

Therefore, the second term of the RHS of (6) over all augmenting paths is $O(W \varepsilon^{-O(d)} \log^4 n \log^4 \log n)$ and the total running time is $O(W(\varepsilon^{-O(d^3)} + \varepsilon^{-O(d)} \log^4 n \log^4 \log n))$. Since $W = \Theta(n/\varepsilon)$ with probability at least $1/2$, we get the following:

▶ **Lemma 10.** *Let $A, B \in \mathbb{R}^d$ be two point sets of size n each and a parameter $0 < \varepsilon \leq 1$ that satisfy $(P1) - (P3)$. With probability at least $1/2$, an ε-approximate matching of A, B can be computed in time $O(n(\varepsilon^{-O(d^3)} + \varepsilon^{-O(d)} \log^4 n \log^4 \log n))$.*
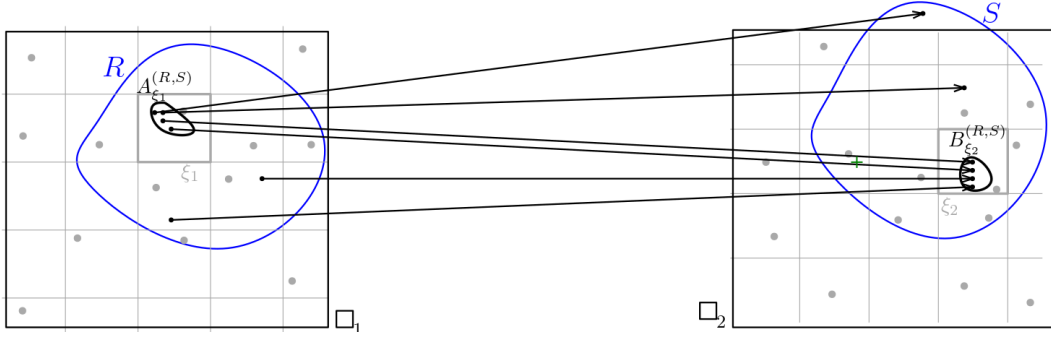
■ **Figure 2** Illustrates two boundary clusters $A_{\xi_1}^{(R,S)}$ and $B_{\xi_2}^{(R,S)}$. $\square_1$ and $\square_2$ are siblings. Grey subdivision represents the subcells and matching edges are shown as solid black.

5   Algorithm Details

In this section, we present the details of the BUILD, HUNGARIANSEARCH and AUGMENT procedures for some level i. For any cell, we cluster the points inside the cell into $O(\mu_{i+1}^{1/4}\varepsilon^{-d}\log n \log\log n)$ clusters (Section 5.1). We use these clusters to compress the residual graph (Section 5.2) and feasibility conditions (Section 5.3). Next, we describe an efficient implementation of the BUILD, HUNGARIANSEARCH, and AUGMENT procedures using the compressed graph and feasibility conditions (Section 5.4). Our compressed graph is different from [16, 14] as it does not recursively expand an augmenting path in this compressed graph to an admissible augmenting path with respect to M. Instead, our algorithm may modify the matching M to another matching M' of the same cost and size and returns an admissible path with respect to M'.

We begin by introducing a few notation that will assist in describing the procedures. For any cell $\square$ and $\xi \in \mathsf{S}_\square$, let $K \subseteq \mathsf{C}[\square]$ be the subset of children of $\square$ that are contained inside ξ. Let $D(\xi) = \bigcup_{\square'' \in K} \mathsf{S}_{\square''}$ be the set of subcells of the children cells of $\square$ that lie inside ξ. For any cell $\square$ with $\mathrm{lev}(\square) < h-1$ and for any $j < \mathrm{lev}(\square)$, let $\square^j$ be the ancestor of $\square$ at level j. Consider any $\square' \in \mathsf{C}[\square]$ and a subcell $\xi \in \mathsf{S}_{\square'}$ of $\square'$. Then, it can be shown that each level-j edge $(a,b) \in A_{\square^j} \times B_{\square^j}$ with one endpoint in ξ is covered by at least one WSPD pair from a subset $\mathcal{W}_\xi^j \subseteq \mathcal{W}_{\square^j}$, with $|\mathcal{W}_\xi^j| = O(\varepsilon^{-1}\log n)$.

▶ **Lemma 11.** *For any cell $\square$ with $\mathrm{lev}(\square) < h-1$, for any $j < \mathrm{lev}(\square)$, and for any subcell $\xi \in \mathsf{S}_\square$, $|\mathcal{W}_\xi^j| = O(\varepsilon^{-1}\log n)$. Furthermore, for any non-empty subcell $\xi' \in D(\xi)$, $\mathcal{W}_\xi^j = \mathcal{W}_{\xi'}^j$.*

5.1   Clustering points

For any cell $\square$ of level $k \in [i, h-1)$, we partition $A_\square \cup B_\square$ into a set of clusters denoted by $V_\square$. For a subcell ξ of a child $\square'$ of $\square$ (i.e, $\xi \in \mathsf{S}_{\square'}$), we partition $A_\xi \cup B_\xi$ into three types of clusters. We create one *free cluster* A_ξ^F (resp. B_ξ^F) for all free points of A_ξ (resp. B_ξ) and one *internal cluster* A_ξ^I (resp. B_ξ^I) for all points $a \in A_\xi$ (resp. $b \in B_\xi$) such that $m(a) \in \square'$ (resp. $m(b) \in \square'$). Additionally, we create *boundary clusters* as follows: For any level $j \in [i, k]$, recall that $\mathcal{W}_\xi^j$ is the set of WSPD pairs that cover all level-j edges with at least one endpoint in ξ. For every pair $(R, S) \in \mathcal{W}_\xi^j$, we create one cluster $A_\xi^{(R,S)}$ (resp. $B_\xi^{(R,S)}$) of A (resp. B) that contains all points $a \in A_\xi$ (resp. $b \in B_\xi$) whose matching edge $(a, m(a))$ (resp. $(m(b), b)$) is a level-j edge that is covered by the well-separated pair (R, S) (See Figure 2). For each level j, there are $O(\varepsilon^{-d}\log n)$ WSPD pairs in $\mathcal{W}_\xi^j$ and there are

$O(\log \log n)$ levels. Therefore, there are $O(\varepsilon^{-d} \log n \log \log n)$ many clusters of this type.

For any cell $\square$ of level $k \in [i, h-1)$, and for any child $\square' \in \mathsf{C}[\square]$, every subcell $\xi \in \mathsf{S}_{\square'}$ is formed by combining the children of $\square'$. For any subcell $\xi \in \mathsf{S}_\square$, the free and internal clusters of ξ are formed as in Equation (7).

$$A_\xi^F = \bigcup_{\xi' \in D(\xi)} A_{\xi'}^F, \qquad\qquad\qquad B_\xi^F = \bigcup_{\xi' \in D(\xi)} B_{\xi'}^F. \qquad\qquad (7)$$

$$A_\xi^I = \bigcup_{\xi' \in D(\xi)} \{(A_{\xi'}^I \cup \bigcup_{(R,S) \in \mathcal{W}_{\xi'}^{k+1}} A_{\xi'}^{(R,S)})\}, \quad B_\xi^I = \bigcup_{\xi' \in D(\xi)} \{(B_{\xi'}^I \cup \bigcup_{(R,S) \in \mathcal{W}_{\xi'}^{k+1}} B_{\xi'}^{(R,S)})\}.$$

For any $i \le j \le k$ and any $(R,S) \in \mathcal{W}_\xi^j$, the boundary cluster of ξ corresponding to (R,S) is formed as follows.

$$A_\xi^{(R,S)} = \bigcup_{\xi' \in D(\xi)} A_{\xi'}^{(R,S)}, \qquad\qquad\qquad B_\xi^{(R,S)} = \bigcup_{\xi' \in D(\xi)} B_{\xi'}^{(R,S)}. \qquad (8)$$

Based on these relations, we extend the definition of $D(\cdot)$ for any cluster X to denote the clusters that combine to form X as $D(X)$. We refer to each cluster $X' \in D(X)$ as a *child-cluster* of X.

If $\square$ is a level $h-1$ node, for every child $\square' \in \mathsf{C}[\square]$, we add $A_{\square'} \cup B_{\square'}$ to $V_\square$ and appropriately classify them as free, internal, or boundary cluster depending on whether they are free, matched inside $\square'$, or outside $\square'$.

We present an upper bound on the number of subcells of the children of any cell in the following lemma. See Appendix B for a proof.

▶ **Lemma 12.** *For any cell $\square$ at level $i < h-1$, $\sum_{\square' \in \mathsf{C}[\square]} |\mathsf{S}_{\square'}| = O(\mu_{i+1}^{1/4})$.*

For each cell $\square$ at level $i < h-1$ and each child $\square' \in \mathsf{C}[\square]$, any subcell $\xi \in \mathsf{S}_{\square'}$ contributes $O(\varepsilon^{-d} \log n \log \log n)$ clusters to $V_\square$. Therefore, by Lemma 12,

▶ **Corollary 13.** *For any cell $\square$ at level $i < h-1$, $|V_\square| = O(\mu_{i+1}^{1/4} \varepsilon^{-d} \log n \log \log n)$.*

5.2   Compressed residual graph

At every non-leaf node $\square$ in the tree T, we create a *compressed residual graph* $\mathcal{AG}_\square$ of $\mathcal{G}_M^\square$ with $V_\square$ being its vertex set. For any non-leaf node $\square$, the vertex set of $\mathcal{AG}_\square$ consists of one vertex for each cluster in $V_\square$. For any cell $\square$ and its child $\square' \in \mathsf{C}[\square]$, we use $V_\square(\square')$ to denote the clusters of $V_\square$ that are inside $\square'$. Next, we define $E_\square$, the set of edges of $\mathcal{AG}_\square$, and a weight $\Phi(X, Y)$ for every edge $(X, Y) \in E_\square$.

If $\text{lev}(\square) = h-1$, we simply set the edges of $\mathcal{AG}_\square$ to be the edges of $\mathcal{G}_M^\square$ and use its net-cost as the weight, i.e., for any edge (u, v) in $\mathcal{G}_M^\square$, $\Phi(u, v) = \phi(u, v)$. If $\text{lev}(\square) < h-1$, then we define *internal* and *bridge* edges between vertices of $V_\square$ as follows:

Bridge edges: For any two children $\square_1 \ne \square_2 \in \mathsf{C}[\square]$, let $X_1 \in V_\square(\square_1)$ and $X_2 \in V_\square(\square_2)$ be two clusters in those children, such that X_1 (resp. X_2) is a cluster of type A (resp. B). If there is at least one non-matching edge $(b, a) \in X_2 \times X_1$, we add a directed edge from X_2 to X_1 and assign it a weight equal to $\Phi(X_2, X_1) = \phi(b, a)$. We refer to this edge as a *non-matching* arc. If there is a matching edge $(a, b) \in X_1 \times X_2$, we add an edge from X_1 to X_2 and set its cost to be $\Phi(X_1, X_2) = \phi(a, b)$. We call this edge from X_1 to X_2 a *matching arc*.

We classify clusters as entry and exit clusters, and define internal edges from an entry to an exit cluster: The free cluster B_ξ^F, the internal cluster A_ξ^I, and every boundary cluster

$B_\xi^{(R,S)}$, for $i \leq j \leq k$ and $(R,S) \in \mathcal{W}_\xi^j$, is designated as an *entry cluster*. The free cluster A_ξ^F, the internal cluster B_ξ^I, and every boundary cluster $A_\xi^{(R,S)}$, for $i \leq j \leq k$ and $(R,S) \in \mathcal{W}_\xi^j$, is designated as an *exit cluster*. For any cell $\square$ and its child $\square' \in \mathsf{C}[\square]$, we use $V_\square^\downarrow(\square')$ and $V_\square^\uparrow(\square')$ to denote the entry and exit clusters of $V_\square(\square')$.

We classify entry and exit clusters in this way for the following reason. Consider any augmenting path P. For any cell $\square'$, consider any edge (u,v) of P such that (u,v) is in $\mathcal{G}_M^{\square'}$ but the edge preceding (u,v) (resp. succeeding (u,v)) is not. Then u has to be a point in an entry (resp. exit) cluster.

Internal edges: Let $\square'$ be any child of $\square$. For any pair of clusters $(X_1, X_2) \in V_\square^\downarrow(\square') \times V_\square^\uparrow(\square')$, we create an *internal edge* (X_1, X_2) in $\mathcal{AG}_\square$. Let $E_\square(\square')$ denote the set of these edges. For any $(X_1', X_2') \in V(\square') \times V(\square')$, define $P_{\square'}(X_1', X_2')$ to be the minimum-weight path between X_1' and X_2' in $\mathcal{AG}_{\square'}$. Define $P(X_1, X_2) = \arg\min_{X_1' \in D(X_1), X_2' \in D(X_2)} \Phi(P_{\square'}(X_1', X_2'))$. We set $\Phi(X_1, X_2)$ to be the weight of the path $P(X_1, X_2)$ in $\mathcal{AG}_{\square'}$.

For consistency, if $\square$ is a cell of level $h - 1$, any edge that lies completely inside a child of $\square$ becomes an internal edge and edges that go between two points in two different children is referred to as a bridge edge.

We abuse notation and refer to any directed path P between two free clusters in the compressed residual graph $\mathcal{AG}_\square$ as an *augmenting path*. For efficiency reasons, we only store the internal edges of $E_\square$. To compute the bridge edges and their costs efficiently, we construct an ε-WSPD as described in Section 2. In the following lemma, we bound the total size of all compressed residual graphs across all cells. See Appendix B for a proof.

▶ **Lemma 14.** *The total size of all compressed residual graphs across all cells of T is $O(n \log n (\varepsilon^{-1} \log \log n)^{O(d)})$.*

▶ **Lemma 15.** *For any cell $\square$ and for any augmenting path P from u to v in $\mathcal{G}_M^\square$, there is an augmenting path P' in $\mathcal{AG}_\square$ that goes from the cluster of u to the cluster of v and $\Phi(P') \leq \phi(P)$.*

5.3   Compressed Feasibility

We use the compressed residual graph to compute an augmenting path. To assist us with the computation of this path, we describe a set of dual weights of $V_\square$ and a set of feasibility conditions for the edges of the compressed graph. Let $\square$ be a cell of level i. For every $X \in V_\square$, we assign a dual weight $y(X)$. We say that a matching and dual weights are *compressed feasible* with respect to $\mathcal{AG}_\square$ if, for any directed edge $(X, Y) \in E_\square$,

$$y(X) - y(Y) \leq \Phi(X, Y). \tag{9}$$

Next, we define slack on any compressed edge (X, Y) to be $s(X, Y) = \Phi(X, Y) - y(X) + y(Y)$. Note that $s(X, Y) \geq 0$ for a compressed feasible matching. We say that an edge (X, Y) is *admissible* if $s(X, Y) = 0$. Similar to (3), one can express the weight of any path P in $\mathcal{AG}_\square$ from X to Y using the slacks on its edges as follows.

$$\Phi(P) = \sum_{(X', Y') \in P} (y(X') - y(Y') + s(X', Y')) = y(X) - y(Y) + \sum_{(X', Y') \in P} s(X', Y'). \tag{10}$$

After any execution of Build, HungarianSearch, or Augment procedures at $\square$ within our algorithm, in addition to (J1) and (J2), our algorithm also satisfies a third invariant. Let $\square'$ be either $\square$ or any descendant of $\square$ in T.

(J3)   There exists a set of dual weights $y(\cdot)$ on $V_{\square'}$ that satisfy compressed feasibility conditions for edges in $\mathcal{AG}_{\square'}$. In addition, for any $X \in V_{\square'}$, if X is a free cluster of A, then $y(X) = 0$ and if X is a free cluster of B, then $y(X) = \max_{X' \in V_{\square'}} y(X')$. Furthermore, for any cluster $X \in V_{\square'}$,

(a)   if X is an internal entry (resp. exit) cluster, $y(X) \leq \min_{X' \in D(X)} y(X')$ (resp. $y(X) \geq \max_{X' \in D(X)} y(X')$), and

(b)   if X is a free or a boundary cluster then for every cluster $X' \in D(X)$, $y(X') = y(X)$.

5.4   Details of the Procedures

Assume that the algorithm has executed until level $i + 1$ and (J1)–(J3) hold at the end. We describe the implementation of BUILD, HUNGARIANSEARCH and AUGMENT procedures and show that (J1)–(J3) continue to hold after their executions.

5.4.1   Build procedure

For any cell $\square$ of level i and for every $\square' \in \mathsf{C}[\square]$, we have a set of compressed feasible dual weights on $V_{\square'}$. The BUILD procedure creates a cluster X at $\square$ by simply combining the clusters $D(X)$ of its children and set its dual weight $y(X)$ to $\min_{X' \in D(X)} y(X')$ (resp. $\max_{X' \in D(X)} y(X')$) provided X is an entry (resp. exit) cluster.

In order to compute the weight of any internal edge $(X, Y) \in V_{\square}^{\downarrow}(\square') \times V_{\square}^{\uparrow}(\square')$, we observe that from (10), the minimum-weight path $P(X, Y)$ is also the path with the smallest total slack between any two clusters $X', Y' \in D(X) \times D(Y)$ in $\mathcal{AG}_{\square'}$. For every entry cluster $X \in V_{\square}^{\downarrow}(\square')$, this can be found in a straight-forward way using an execution of Dijkstra's shortest-path algorithm. More specifically, we add a source s to $\mathcal{AG}_{\square'}$, connect s to all $X' \in D(X)$ with an edge of weight of $y(X')$, and replace the weight of every other edge in $\mathcal{AG}_{\square'}$ with its slack. Then, we execute Dijkstra's algorithm in $\mathcal{AG}_{\square'}$ from s to find the distance of each cluster Y' from s, denoted by $\kappa_{Y'}$. For any exit cluster $Y \in V_{\square}^{\uparrow}(\square')$, we set the weight of the edge (X, Y) in $\mathcal{AG}_{\square}$, to be $\Phi(X, Y) = \min_{Y' \in D(Y)}\{\kappa_{Y'} - y(Y')\}$.

The BUILD procedure does not affect the invariants (J1) and (J2). The following lemma states that the invariant (J3) holds after the execution of BUILD procedure.

▶ **Lemma 16.** *The dual weights assigned to the clusters of $V_{\square}$ by the BUILD procedure satisfy (J3).*

Efficiency of the Build procedure: To compute the internal edges incident on the entry cluster X, instead of using an $O(|V_{\square'}|^2)$ time Dijkstra's algorithm, as in [14], we use the WSPDs in a straight-forward way to compute the shortest path in $O(|E_{\square'}| \log |E_{\square'}|)$ time. Given that the number of entry clusters in each cell is $\log n (\varepsilon^{-1} \log \log n)^{O(d)}$ and since $|E_{\square'}| = O(\varepsilon^{-O(d)} n_{\square'} \log n \log^{O(d)} \log n))$ (from Lemma 14), the total running time of the BUILD procedure is

$$O\left(\sum_{\square' \in \mathsf{C}[\square]} \sum_{X \in V_{\square}^{\downarrow}(\square')} \varepsilon^{-O(d)} n_{\square'} \log^2 n \log^{O(d)} \log n \right) = O(n_{\square} \log^3 n_{\square} (\varepsilon^{-1} \log \log n)^{O(d)}).$$

5.4.2   HungarianSearch procedure

Let $\square$ be a cell of level i. The HUNGARIANSEARCH procedure on $\square$ consists of two parts. In the first part, the algorithm modifies the dual weights of $V_{\square}$ and make the edges on the

minimum-weight augmenting path in $\mathcal{AG}_\square$ admissible (Search step). Then, the algorithm updates the dual weights of the clusters and points and may modify the matching M to another feasible matching M' with $w(M') = w(M)$. Then, it returns an admissible augmenting path in $\mathcal{G}_{M'}^\square$ with respect to M' (Update step). This path is also the minimum net-cost augmenting path with respect to M'.

Search Step: From (10), the path from a free cluster F' to a free cluster F with the smallest total slack is also the minimum-weight path between F' and F in $\mathcal{AG}_\square$. To compute this path, we replace the cost of every edge in $\mathcal{AG}_\square$ with its slack and then execute a Dijkstra's algorithm that starts at the free clusters of B. Let P_X be the shortest path from a free cluster of B to any cluster X and κ_X be its cost. Let F be a free cluster of A that has the smallest shortest path. If there are no free clusters of A in $\mathcal{AG}_\square$, then the HUNGARIANSEARCH returns NULL. Let the path P_F start at some free cluster F' of B. P_F is the minimum-weight augmenting path in $\mathcal{AG}_\square$ and $y(F) = 0$ (from (J3)). Therefore, from (10), the augmenting path P_F has a weight of $\Phi(P_F) = y(F') + \kappa_F$. If $\Phi(P_F) \leq \mu_i$, let $U \subseteq V_\square$ be the subset of clusters whose shortest-path distances from s is less than κ_F. We update the dual weights of any cluster $X \in U$ by setting $y(X) \leftarrow y(X) - \kappa_X + \kappa_F$. If $\Phi(P_F) > \mu_i$, the algorithm sets $\kappa_F = \mu_i - y(F')$, updates the dual weights as described above and then returns NULL. The dual updates to the clusters of $V_\square$ in the search step ensures that the dual weights of free clusters of B do not exceed μ_i.

For any cell $\square_1$ and its child $\square_2 \in \mathsf{C}[\square_1]$, we say that the dual weights of $V_{\square_1}$ *dominates* the dual weights of $V_{\square_2}$ if for each exit cluster $X \in V_{\square_1}^\uparrow(\square_2)$, $y(X) \geq \max_{X' \in D(X)} y(X')$. During the search step, the dual weight of any cluster $X \in V_\square$ is non-decreasing. Therefore, after the search step, for each child $\square' \in \mathsf{C}[\square]$, the dual weights of $V_\square$ dominates the dual weights of $V_{\square'}$. Furthermore, the updated dual weights are compressed feasible and the edges of P_F is admissible.

▶ **Lemma 17.** *For any cell $\square$, after the execution of the search step of the HUNGARIANSEARCH procedure, the updated dual weights of $V_\square$ are compressed feasible and every edge on the minimum-weight path computed by the search step is admissible. Furthermore, for any child $\square' \in \mathsf{C}[\square]$, the dual weights of $V_\square$ dominate the dual weights of $V_{\square'}$.*

After the search step, the updated dual weights of $V_\square$ remain compressed feasible and the edges of P_F are admissible. In the Update Step, we expand the path P_F into an admissible augmenting path in the residual graph. We describe a procedure called SYNC that assists in expanding this path. In particular, consider any admissible edge (X, Y) on P_F where (X, Y) is an internal edge for some child $\square' \in \mathsf{C}[\square]$. The SYNC procedure updates the dual weight of $V_{\square'}$ so that the path $P(X, Y)$ becomes admissible.

Sync Procedure: The SYNC procedure takes any cell $\square_1$ and its child $\square_2 \in \mathsf{C}[\square_1]$ along with a set of compressed feasible dual weights of $V_{\square_1}$ and $V_{\square_2}$ such that the dual weights of $V_{\square_1}$ dominates the dual weights of $V_{\square_2}$. This procedure then generates a set of compressed feasible dual weights of $V_{\square_2}$ such that the dual weights of $V_{\square_1}$ and $V_{\square_2}$ satisfy conditions (a) and (b) of (J3). Furthermore, if any internal edge $(X, Y) \in E_{\square_1}(\square_2)$ is admissible, then the path $P(X, Y)$ is also admissible with respect to the dual weights of $V_{\square_2}$. The description of the SYNC procedure is provided in Appendix C. Let $\square_1$ be a cell of level j of T. The execution of the SYNC procedure on $\square_2$ requires an execution of Dijkstra's algorithm on $\mathcal{AG}_{\square_2}$ and takes a total of $O(\mu_{j+1}\varepsilon^{-O(d)} \log^3 n \log^2 \log n)$ time. The following lemma states the important properties of the SYNC procedure.

▶ **Lemma 18.** *For any cell $\square_1$ and any child $\square_2 \in \mathsf{C}[\square_1]$, suppose the set of dual weights of $V_{\square_1}$ and $V_{\square_2}$ are compressed feasible and the set of dual weight of $V_{\square_1}(\square_2)$ dominates the set of dual weights of $V_{\square_2}$. After applying the SYNC procedure on $\square_2$, we have that:*

(i)   *The updated dual weights on $\mathcal{AG}_{\square_2}$ are compressed feasible,*

(ii) *For any cluster $X \in V_{\square_1}(\square_2)$, the dual weight of X and clusters in $D(X)$ satisfies the conditions (a) and (b) of (J3), and,*

(iii) *If $\square_2$ is not a leaf cell, for any $\square_3 \in \mathsf{C}[\square_2]$, the dual weights of $V_{\square_2}(\square_3)$ dominates the set of dual weights of $V_{\square_3}$.*

Using Lemma 18 part (iii), it is clear that one can recursively apply the SYNC procedure on all descendants of a cell $\square$. The following lemma shows that after the search step, if we apply the SYNC procedure on all descendants of every cell $\square \in \Delta[i]$, the dual weights assigned to all clusters and points satisfy (J1)–(J3).

▶ **Lemma 19.** *After executing the search step of HUNGARIANSEARCH on $\mathcal{AG}_\square$, for every $\square \in \Delta[i]$, suppose we apply SYNC to $\square$ and all its descendants. The resulting up-to-date dual weights satisfy (J1)–(J3).*

The following lemma helps in converting the minimum-weight path obtained by the search step into an admissible augmenting path.

▶ **Lemma 20.** *For any admissible internal edge $(X, Y) \in E_\square(\square')$, let $X' \in D(X)$ and $Y' \in D(Y)$ be the clusters containing the first and the last vertex of some $P(X, Y)$. Then, after calling the SYNC procedure on $\square'$, the path $P(X, Y)$ is admissible, $y(X') = y(X)$, and $y(Y') = y(Y)$.*

Using the SYNC procedure, the Update step converts the augmenting path P_F returned by the Search step to an admissible augmenting path $\tilde{P}_F$ in the residual graph. In this process, the Update step might change the matching M to another matching M' with the same weight and size. We describe the Update step as a recursive procedure that initially takes P_F as the input.

Update step: For any cell $\square'$, the Update step takes any admissible path $P = \langle X = X_1, X_2, \ldots, X_m = Y \rangle$ in $\mathcal{AG}_{\square'}$ as input and returns an admissible alternating path from a point p to p' in $\mathcal{G}_M^{\square'}$ with the property that $y(p) = y(X)$ and $y(p') = y(Y)$.

If $\square'$ is a cell of level $h - 1$, then P is also an admissible path in $\mathcal{G}_M^{\square'}$ and the procedure returns this path. Otherwise, let $k = \mathrm{lev}(\square') < h - 1$ be the level of $\square'$. Let $\mathcal{I}$ denote the set of all internal edges on the path P. Note that $\mathcal{I}$ is a set of vertex-disjoint edges. Let $\mathcal{B}$ be the set of all clusters X_t on P that do not participate in any edge of $\mathcal{I}$. It is easy to see that $\mathcal{B}$ is a set of boundary clusters.

For any internal edge $(X_j, X_{j+1}) \in \mathcal{I}$, let $\square_j \in \mathsf{C}[\square']$ such that $(X_j, X_{j+1}) \in E_{\square'}(\square_j)$. We execute the SYNC procedure on $\square_j$. Since (X_j, X_{j+1}) is an admissible edge, the path $P(X_j, X_{j+1})$ is admissible with $y(X'_j) = y(X_j)$ and $y(X'_{j+1}) = y(X_{j+1})$ (From Lemma 20). We recursively apply the Update step on $P(X_j, X_{j+1})$.

Assume that for each internal edge $(X_j, X_{j+1}) \in \mathcal{I}$, this recursive call has returned an admissible path Π_j in the residual graph from a point $p_j \in X_j$ to a point $p_{j+1} \in X_{j+1}$ with $y(p_j) = y(X_j)$ and $y(p_{j+1}) = y(X_{j+1})$. We *select* the point p_j for the cluster X_j and the point p_{j+1} for the cluster X_{j+1}. For any boundary cluster $X_t \in \mathcal{B}$, we select an arbitrary point p_t. Let Z be the set of all ancestors of p_t in T of level greater than k. First, we iteratively apply SYNC on every cell in Z in increasing order of their level. After all executions of the SYNC procedure, from Lemma 18 (ii), $y(p_t) = y(X_t)$. Thus, from every cluster X_j on P, we have selected one point p_j with $y(p_j) = y(X_j)$.

Next, we construct the admissible alternating path $\tilde{P}$ corresponding to the path P as follows. For every internal edge $(X_j, X_{j+1}) \in \mathcal{I}$, we replace (X_j, X_{j+1}) with the path Π_j.
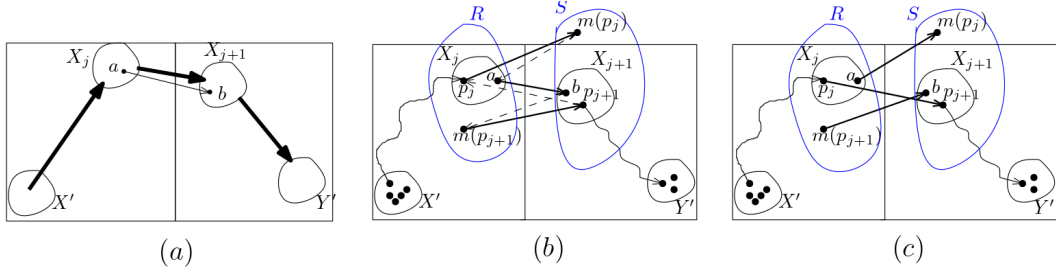
Figure 3 (a) Compact minimum-weight path P in $\mathcal{AG}_\square$. (b) and (c) show how the matching M is modified to a matching M' to obtain an augmenting path.

For every bridge edge (X_j, X_{j+1}), if (X_j, X_{j+1}) is a non-matching arc, we simply add an edge from p_j to p_{j+1} to the path. If (X_j, X_{j+1}) is a matching arc, then both X_j and X_{j+1} are boundary clusters. While there may not be a matching edge between p_j and p_{j+1}, we know that there is some matching edge (a, b) such that $a \in X_j$ and $b \in X_{j+1}$. Let Z (resp. Z') be the set of all ancestors of a (resp. b) of level greater than k. We apply the SYNC procedure on the cells in Z (resp. Z') in increasing order of their level. Then, we modify our matching M to M' as follows: Add matching edges $(p_j, p_{j+1}), (a, m(p_j))$, and $(b, m(p_{j+1}))$ to the matching and remove the edges $(a, b), (p_j, m(p_j))$, and $(p_{j+1}, m(p_{j+1}))$ from the matching (See Figure 3). The new matching continues to be feasible since the dual weights of a (resp. b) and p_j (resp. p_{j+1}) are identical. This is because X_j (resp. X_{j+1}) is a boundary cluster and therefore, from Lemma 18 (ii), the application of the SYNC procedure on the ancestors of p_j (resp. p_{j+1}) and a (resp. b) will make $y(a) = y(p_j)$ (resp. $y(b) = y(p_{j+1})$). Note that, for every internal edge (X_j, X_{j+1}), the path Π_j consists of only admissible edges. Furthermore, for every bridge edge (X_j, X_{j+1}), the edge (p_j, p_{j+1}) added to the path is admissible. This follows from the fact that $y(p_j) = y(X_j)$, $y(p_{j+1}) = y(X_{j+1})$, and (X_j, X_{j+1}) is admissible. Finally, the dual weight of the first (resp. last) point p_1 (resp. p_m) is equal to $y(X)$ (resp. $y(Y)$) as desired. This completes the description of the Update step. Let $\tilde{P}_F$ be the admissible augmenting path in $\mathcal{G}_\square$ returned by the Update step with P_F as input.

▶ **Lemma 21.** *The matching M can be modified to another matching M' so that, $w(M) = w(M')$, $M', y(\cdot)$ is feasible and the compressed residual graph at each node remains unchanged. Furthermore, there is an admissible augmenting path in the residual graph $\mathcal{G}_{M'}$.*

Recollect that, we only require the existence of a set of dual weights that satisfy the conditions in (J1)–(J3). For efficiency reasons, the Update step does not maintain the up-to-date dual weights explicitly. Instead, it computes the up-to-date dual weights for all cells $\square'$ whose $V_{\square'}$ or $M_{\square'}$ may change after augmenting M along $\tilde{P}_F$. For every other cell $\square''$, from Lemma 22 and Lemma 19, we can always retrieve the up-to-date dual weights for these cells satisfying (J1)–(J3) by recursively applying SYNC on $\square''$ and all its descendants.

▶ **Lemma 22.** *During the execution of our algorithm, consider a sequence of consecutive applications of the SYNC procedure on a cell $\square$. If $M_\square$ and the clusters in $V_\square$ remain unchanged, then, this sequence of SYNC executions can be replaced with the last one while producing the same set of dual weights of $V_\square$.*

Efficiency of the HungarianSearch procedure: The search step requires execution of a single Dijkstra's algorithm on $\mathcal{AG}_\square$ which takes $O(\mu_{i+1}\varepsilon^{-O(d)} \log^3 n \log^2 \log n)$ time. Applying

SYNC procedure for a cell $\square'$ of level j requires an execution of Dijkstra's algorithm on $\mathcal{AG}_{\square'}$, which takes $O(\mu_{j+1}\varepsilon^{-O(d)}\log^3 n\log^2\log n)$ time. Recall that for any level j of T and an augmenting path $\tilde{P}_F$ on the residual graph, k_j denotes the number of level j cells containing at least one point of $\tilde{P}_F$. In the Update step, For each level $j > i$, we execute the SYNC procedure on the cell of level j containing at least one point of $\tilde{P}_F$. Furthermore, during the Update step, for each bridge matching arc (X_j, X_{j+1}) the algorithm may also apply the SYNC procedure on an additional $O(\log\log n)$ cells. This is done before the algorithm modifies the matching. These executions of the SYNC procedure can be charged to the $O(\log\log n)$ SYNC procedures executed for the ancestors of points $p_j \in X_j$ and $p_{j+1} \in X_{j+1}$. Therefore, there are at most $O(k_j \log\log n)$ executions of the SYNC procedure during the execution of the Update step. The execution time of HUNGARIANSEARCH procedure, therefore, is $O(\mu_{i+1}\varepsilon^{-O(d)}\log^3 n\log^2\log n + \sum_{j=i+1}^{h-1} k_j\mu_{j+1}\varepsilon^{-O(d)}\log^3 n\log^3\log n)$.

5.4.3 Augment procedure

Given an augmenting path $P = \langle b_0, a_0, b_1, \cdots, b_k, a_k \rangle$ with respect to the matching M, the AUGMENT procedure will simply update $M \leftarrow M \oplus P$. After augmentation, any edge (a_i, b_i) is a matching edge and any edge (b_{i+1}, a_i) is a non-matching edge (Note that the direction of the edges are reversed after the augmentation). For a new matching edge (a_i, b_i) after an augmentation, suppose $\square_i$ is the least common ancestor of a_i and b_i. Let $X, Y \in V_{\square_i}$ be the pair of boundary clusters such that $(a_i, b_i) \in X \times Y$. If there exists another matching edge $(a_i', b_i') \in (X \times Y) \setminus P$, then a_i and b_i inherit their dual weights, i.e., $y(a_i) \leftarrow y(a_i')$ and $y(b_i) \leftarrow y(b_i')$. Otherwise, their dual weight remains unchanged. This completes the description of the AUGMENT procedure. For every new matching edge (a_i, b_i), the procedure may inherit the dual weights from another matching edge (a_i', b_i') that did not participate in P. Since (J2) holds prior to augmentation, $y(a_i')$ and $y(b_i')$ satisfy (J2). Therefore, post augmentation, $y(a_i)$ and $y(b_i)$ also satisfy (J2).

The following lemma shows that the dual weights of the points after the AUGMENT procedure remains feasible and (J1) holds.

▶ **Lemma 23.** *After augmenting the matching and updating the dual weights by the* AUGMENT *procedure, the dual weights of the points are feasible with respect to the new matching.*

The vertex and the edge sets of the compressed residual graph change after augmentation. The AUGMENT procedure creates the new clusters at all ancestors of every point in P in a straight-forward way. In a bottom-up fashion, for any cluster $X \in V_\square$, if X is an exit (resp. entry) cluster, it assigns $\max_{X' \in D(X)} y(X')$ (resp. $\min_{X' \in D(X)} y(X')$) as $y(X)$. For each edge on P, the procedure will update the bridge edges in $\mathcal{AG}_\square$ in a straight-forward way. The following lemma shows that the updated dual weights are compressed feasible with respect to $\mathcal{AG}_\square$.

▶ **Lemma 24.** *After augmenting the matching, the new set of clusters and their dual weights* $y(\cdot)$ *satisfy (J3).*

Next, for any $\square' \in \mathsf{C}[\square]$, in order to update the weights on the internal edges $E_\square(\square')$ in $\mathcal{AG}_\square$, we apply the BUILD procedure. From Corollary 13, if $\square$ is a cell of level i, then $|V_\square| = O(\mu_{i+1}^{1/4}\varepsilon^{-d}\log n\log\log n)$. Since there at most $O(|V_\square|)$ entry clusters in $V_\square$ and the BUILD procedure executes that many Dijkstra's algorithm to construct the internal edges, the total time taken is bounded by $O(\mu_{i+1}\varepsilon^{-O(d)}\log^3 n\log^3\log n)$.

The AUGMENT procedure executes BUILD procedure on all ancestors of any vertex of P in a bottom-up fashion. Therefore, the total time taken is $O(\sum_{j=1}^{i}(k_j\mu_{i+1}\varepsilon^{-O(d)}\log^3 n\log^3\log n))$.

—— **References** ——

1   Pankaj Agarwal and Kasturi 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   Pankaj K Agarwal, Hsien-Chih Chang, Sharath Raghvendra, and Allen Xiao. Deterministic, near-linear ε-approximation algorithm for geometric bipartite matching. *arXiv preprint arXiv:2204.03875*, 2022.

3   A. Andoni, K. D. Ba, P. Indyk, and D. P. Woodruff. Efficient sketches for earth-mover distance, with applications. *In Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 324–330, 2009.

4   Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proc. 34th International Conference on Machine Learning*, pages 214–223, 2017.

5   Paul B. Callahan and S. Rao Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM*, 42(1):67–90, January 1995.

6   Moses Charikar. Similarity estimation techniques from rounding algorithms. In *Proc. 34th Annu. ACM Sympos. Theory Comput.*, pages 380–388, 2002.

7   Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *In Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 448–455, 2003.

8   Kyle Fox and Jiashuai 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.

9   H. N. Gabow and R.E. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18:1013–1036, 1989.

10   Sariel Har-Peled. *Geometric approximation algorithms*. Number 173. American Mathematical Soc., 2011.

11   Piotr Indyk. A near linear time constant factor approximation for Euclidean bichromatic matching (cost). In *SODA 2007*, page 4, 2007.

12   Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. *In Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2495–2504, 2017.

13   Harold Kuhn. Variants of the hungarian method for assignment problems. *Naval Research Logistics*, 3(4):253–258, 1956.

14   Nathaniel Lahn and Sharath Raghvendra. An $O(n^{5/4})$ time ε-approximation algorithm for RMS matching in a plane. *In Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms*, pages 869–888, 2021.

15   Marcin Mucha and Piotr Sankowski. Maximum matchings via gaussian elimination. *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 248–255, 2004.

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

17   Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.

18   R. Sharathkumar. A sub-quadratic algorithm for bipartite matching of planar points with bounded integer coordinates. In *29th International Symposium on Computational Geometry*, pages 9–16, 2013. `doi:10.1145/2462356.2480283`.

19   R. Sharathkumar and P. K. Agarwal. Algorithms for transportation problem in geometric settings. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 306–317, 2012.

20 Justin Solomon, Fernando De Goes, Gabriel Peyré, Marco Cuturi, Adrian Butscher, Andy Nguyen, Tao Du, and Leonidas Guibas. Convolutional wasserstein distances: Efficient optimal transportation on geometric domains. *ACM Transactions on Graphics*, 34(4):66, 2015.

21 Jan van den Brand, Danupon Nanongkai Yin-Tat Lee, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. *IEEE 61st Annual Symposium on Foundations of Computer Science*, pages 919–930, 2020.

A   Preprocessing Step

Similar to some of the earlier algorithms [16, 14], we perform the following preprocessing step that makes the input "well-conditioned" at a slight increase in the cost of the optimal matching. Using a quad-tree based greedy algorithm [6], we compute a $c_1 \log n$-approximate matching M_0 of A and B, in $O(n \log n)$ time, for some constant $c_1 \geq 0$ [7]. Let $w_0 = \textit{c}(M_0)$. Then $\frac{w_0}{c_1 \log n} \leq \textit{c}(M^*) \leq w_0$.

For any integer $i \in [0, \log(c_1 \log n)]$, define $\beta_i = w_0/2^i$; there is an i such that $\beta_i \leq \textit{c}(M^*) \leq \beta_{i-1}$. Set $t(n) = c_0 n(\varepsilon^{-O(d^3)} + \log^3 n(\varepsilon^{-1} \log \log n)^{O(d)} + \varepsilon^{-O(d)} \log^4 n \log \log^4 n)$ for some sufficiently large constant c_0. We run the algorithm described in Section 4 for at most $t(n)$ steps 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 10 ensures that if $\beta_i \leq \textit{c}(M^*) \leq \beta_{i-1}$ then with probability at least $1/2$, the algorithm returns an ε-approximate matching within $t(n)$ time. Now forward, we assume that we have computed a value $\beta > 0$ such that $\textit{c}(M^*) \leq \beta \leq 2\textit{c}(M^*)$. As in [16, 14], by scaling $A \cup B$ and snapping points to an integer grid, we can assume A and B satisfy the following conditions: (P1) All input points have integer coordinates bounded by $n^{O(1)}$. (P2) No integer grid point contains points of both A and B. (P3) $\textit{c}(M^*) \in \left[\frac{3\sqrt{d}n}{\varepsilon}, \frac{9\sqrt{d}n}{\varepsilon}\right]$. We compute an $(\varepsilon/2)$-approximate matching of A and B satisfying $(P1) - (P3)$ in $t(n)$ time.

B   Missing Proofs

Proof of Lemma 3:   If the edge (a, b) intersects the boundary of a cell at level $i + 1$, then, $\mathrm{lev}(a,b) \leq i$. Therefore, $\Pr[\mathrm{lev}(a,b) = i] \leq \frac{\sqrt{d}\|a-b\|}{\ell_{i+1}}$. As a result,

$$\mathbb{E}[\delta_{ab}] \leq \sum_{i=0}^{h-1} \Pr[\mathrm{lev}(a,b) = i].\mu_{i+1} \leq \sum_{i=0}^{h-1} \frac{\sqrt{d}\|a-b\|}{\ell_{i+1}} \cdot \frac{\varepsilon \ell_{i+1}}{c_2\sqrt{d}(h-i)^2} = \frac{\varepsilon}{c_2}\|a-b\| \sum_{i=0}^{h-1} \frac{1}{(h-i)^2}.$$

$\sum_{i=1}^{\infty} \frac{1}{i^2} = \zeta(2) = \frac{\pi^2}{6}$, where $\zeta(\cdot)$ is the *Riemann zeta function*. Therefore, $\mathbb{E}[\delta_{ab}] \leq \frac{\pi^2}{6c_2}\varepsilon\|a-b\|$.

Proof of Lemma 9:   To prove this lemma, first, we bound the total additive error across all augmenting paths computed during the execution of our algorithm.

▶ **Lemma 25.** $\sum_{P_i \in \mathbb{P}} \sum_{(u,v) \in P_i \cap M_i} \delta_{uv} = O(W \log n)$.

Every level $j-1$ edge in γ_j appears as a matching or a non-matching edge. Furthermore, any matching edge will first appear as a non-matching edge in an augmenting path and carry an additive error of μ_{j+1}. Therefore, we charge at most two level $j-1$ edges in γ_j to the additive error of any non-matching edge. So, the total additive error on all non-matching edges of γ_j is at least $\gamma_j \mu_{j+1}/2$. Combining with Lemma 25, we get $\gamma_j = O(W \log n/\mu_{j+1})$.

Proof of Lemma 12:   By construction, the number of subcells of a cell $\square'$ is bounded by the number of children of $\square'$; i.e, $|\mathsf{S}_{\square'}| \leq |\mathsf{C}[\square']|$. Furthermore, for any cell $\square'$ at level j, $|\mathsf{C}[\square']| \leq (\frac{\ell_j}{\ell_{j+1}})^d = \ell_j^{d(1-\alpha)}$. As a result,

$$\sum_{\square' \in \mathsf{C}[\square]} |\mathsf{S}_{\square'}| \leq \sum_{\square' \in \mathsf{C}[\square]} |\mathsf{C}[\square']| \leq \ell_i^{d(1-\alpha)} \ell_{i+1}^{d(1-\alpha)} \leq \ell_i^{\frac{2d}{8d+2}}. \tag{11}$$

From Lemma 2, $\mu_{i+1} \geq \ell_{i+2}$. Furthermore, from the construction of the tree and since $d \geq 2$, $\ell_{i+2} = \ell_i^{(\frac{8d+1}{8d+2})^2} \geq \ell_i^{\frac{8d}{8d+2}}$. Plugging this in (11), we get $\sum_{\square' \in \mathsf{C}[\square]} |\mathsf{S}_{\square'}| = O(\mu_{i+1}^{1/4})$.

Proof of Lemma 14:   By construction, for any cell $\square$ and any child $\square' \in \mathsf{C}[\square]$ and for each pair of clusters $(X, Y) \in V_\square^\downarrow(\square') \times V_\square^\uparrow(\square')$, there exists at most one internal edge in $E_\square$. Therefore, each cluster $X \in V_\square(\square')$ has degree at most $|V_\square(\square')| = O(|\mathsf{S}_{\square'}|\varepsilon^{-d} \log n \log \log n) = O(\varepsilon^{-O(d)} \log n \log^{O(d)} \log n)$, where the last equality is resulted since $|\mathsf{S}_{\square'}| = O(\varepsilon^{-d} \log^{2d} \log n)$. Summing over all clusters in $V_\square$,

$$|E_\square| = O(\varepsilon^{-O(d)} |V_\square| \log n \log^{O(d)} \log n).$$

Summing across all cells in T, we get

$$\sum_{\square \in T} |E_\square| = \sum_{\square \in T} |V_\square| \times O(\varepsilon^{-O(d)} \log n \log^{O(d)} \log n). \tag{12}$$

Since each point participates in a single cluster per level and $O(\log \log n)$ clusters overall, we get $\sum_{\square \in T} |V_\square|$ by $O(n \log \log n)$. Plugging this in (12), we can conclude that the total size of the compressed residual graph across all cells is $O(n \log n (\varepsilon^{-1} \log \log n)^{O(d)})$.

C   Details of the Sync procedure

For any entry cluster $X \in V_\square^\downarrow$ and any cluster $X' \in D(X)$, the dual weight $y(X')$ needs to be no less than the updated $y(X)$ (In the case of free or boundary clusters, it should be equal). We define $\text{inc}(X')$ to be the value by which $y(X')$ should be increased, i.e., $\text{inc}(X') = y(X) - y(X')$. Note that $\text{inc}(X')$ can be negative. Let $\rho_X = \max_{X' \in D(X)} \text{inc}(X')$ and let $\rho = \max\{0, \max_{X \in V_\square^\downarrow(\square')} \rho_X\}$. The value ρ corresponds to the largest increase in dual weights we desire across all child-clusters in each entry cluster. So, for any cluster $X \in V_\square^\downarrow(\square')$ and $X' \in D(X)$, $-\infty < \text{inc}(X') \leq \rho$.

Let $\mathcal{AG}'$ be an augmented compressed residual network that is created by adding a vertex s to $\mathcal{AG}_{\square'}$ and connecting s to every $X' \in D(X)$ for any entry cluster $X \in V_\square^\downarrow(\square')$. We set the weight of (s, X') to be $\rho - \text{inc}(X')$. Since $\text{inc}(X') \leq \rho$, the weight on the edge will be non-negative. For every other edge (U, V), we set its weight to be the slack $s(U, V)$. We then execute Dijkstra's algorithm on $\mathcal{AG}'$ from the source s. For any cluster V, let κ_V denote the weight of the shortest-path distance from s to V. The dual updates are done in an identical fashion to the HUNGARIANSEARCH. Let U denote the set of all clusters $V \in V_{\square'}$ with $\kappa_V < \rho$. For any $V \in U$, we update the dual weight $y(V) \leftarrow y(V) - \kappa_V + \rho$.

After updating these dual weights, for every boundary and free exit cluster $X \in V_\square^\uparrow(\square')$ and any $X' \in D(X)$, we set $y(X') \leftarrow y(X)$. This step will not decrease the dual weight of any cluster. This completes the description of SYNC procedure.