Algorithms for the Transportation Problem in Geometric Settings *

R. Sharathkumar Pankaj K. Agarwal

Department of Computer Science
Duke University
{sharath, pankaj}@cs.duke.edu

Abstract

For $A, B \subset \mathbb{R}^d$, $|A| + |B| = n$, let $a \in A$ have a demand $d_a \in \mathbb{Z}^+$ and $b \in B$ have a supply $s_b \in \mathbb{Z}^+$, $\sum_{a\in A} d_a = \sum_{b\in B} s_b = U$ and let $\mathtt{d}(\cdot,\cdot)$ be a distance function. Suppose the diameter of $A \cup B$ is Δ under $\mathtt{d}(\cdot,\cdot)$, and $\varepsilon > 0$ is a parameter. We present an algorithm that in $O((n\sqrt{U}\log^2 n + U\log U)\Phi(n)\log(\Delta U/\varepsilon))$ time computes a solution to the transportation problem on A, B which is within an additive error ε from the optimal solution. Here $\Phi(n)$ is the query and update time of a dynamic weighted nearest neighbor data structure under distance function $\mathtt{d}(\cdot,\cdot)$. Note that the $(1/\varepsilon)$ appears only in the log term. As among various consequences we obtain,

- For $A, B \subset \mathbb{R}^d$ and for the case where $\mathtt{d}(\cdot,\cdot)$ is a metric, an ε-approximation algorithm for the transportation problem in $O((n\sqrt{U}\log^2 n + U\log U)\Phi(n)\log(U/\varepsilon))$ time.
- For $A, B \subset [\Delta]^d$ and the L_1 and L_∞ distance, exact algorithm for computing an optimal bipartite matching of A, B that runs in $O(n^{3/2}\log^{d+O(1)} n \log\Delta)$ time.
- For $A, B \subset [\Delta]^2$ and RMS distance, exact algorithm for computing an optimal bipartite matching of A, B that runs in $O(n^{3/2+\delta}\log\Delta)$ time, for an arbitrarily small constant $\delta > 0$.

For point sets, $A, B \subset [\Delta]^d$, for the L_p norm and for $0 < \alpha, \beta < 1$, we present a randomized dynamic data structure that maintains a partial solution to the transportation problem under insertions and deletions of points in which at least $(1-\alpha)U$ of the demands are satisfied and whose cost is within $(1+\beta)$ of that of the optimal (complete) solution to the transportation problem with high probability. The insertion, deletion and update times are $O(\text{poly}(\log(n\Delta)/\alpha\beta))$, provided $U = n^{O(1)}$.

1 Introduction

Let A be a set of points in $\mathbb{R}^d$, each $a \in A$ with a positive integral demand d_a and let B be a set of points in $\mathbb{R}^d$, each $b \in B$ with a positive integral supply s_b such that $\sum_{a\in A} d_a = \sum_{b\in B} s_b = U$; set $|A| + |B| = n$. Let $\mathtt{d}(\cdot,\cdot)$ be a distance function. We use $\mathcal{G}(A, B)$ to denote the complete bipartite graph $(A \cup B, A \times B)$[1] with every edge (a, b) having a cost $\mathtt{d}(a, b)$. We call a function $\sigma : A \times B \to \mathbb{Z}_{\geq 0}$ an *assignment* if $\sum_{b\in B}\sigma(a, b) = d_a$ and $\sum_{a\in A}\sigma(a, b) = s_b$. The cost of σ is $\mathtt{w}(\sigma) = \sum_{a\in A, b\in B}\sigma(a, b)\mathtt{d}(a, b)$. The *Hitchcock-Koopmans transportation* problem, or simply the transportation problem for brevity, is to compute an assignment σ_{OPT} for A, B of minimum possible cost. An assignment σ is called *ε-approximate* if $\mathtt{w}(\sigma) \leq (1+\varepsilon)\mathtt{w}(\sigma_{\text{OPT}})$, and *$\varepsilon$-close* if $\mathtt{w}(\sigma) \leq \mathtt{w}(\sigma_{\text{OPT}}) + \varepsilon$. The *bipartite matching* problem is a special case of the transportation problem in which $d_a = s_b = 1$ for all $a \in A, b \in B$. In this case an assignment is called the *perfect matching*. Similarly optimal, ε-approximate and ε-close matchings are optimal, ε-approximate and ε-close assignments in this special case. We study the problem of computing optimal, ε-close and ε-approximate assignments to the transportation problem.

Previous Work. Optimal assignments on weighted bipartite graphs with n vertices can be computed using the Hungarian algorithm in $O(n^3)$ time [11]. For unweighted bipartite graphs with n vertices and m edges, a maximum-cardinality matching can be computed in time $O(m\sqrt{n})$ [8]. If edge costs are positive integers bounded by $n^{O(1)}$, Gabow and Tarjan [7] show that an optimal matching can be computed in time $O(n\sqrt{m}\log n)$ and an optimal assignment for the transportation problem in time $O((\min(\sqrt{U}, n)m + U\log U)\log n)$. If $A, B \subset \mathbb{R}^2$ and $\mathtt{d}(\cdot,\cdot)$ is the L_2 norm, Vaidya [14] shows that the optimal matching on $\mathcal{G}(A, B)$ can be computed in time $O(n^{2.5})$. Atkinson and Vaidya [4] extend this approach to an $O(n^{2.5}\log U)$ algorithm for the transportation problem. Agarwal *et al.* [1] improve the running time of the algorithms for matching and transportation problems to $O(n^{2+\delta})$ and $O(n^{2+\delta}\log U)$, where $\delta > 0$ is an arbitrarily small constant, respectively. For

*This work is supported by NSF under grants CNS-05-40347, IIS-07-13498, CCF-09-40671, and CCF-1012254, by ARO grants W911NF-07-1-0376 and W911NF-08-1-0452, by an NIH grant 1P50-GM-08183-01, and by a grant from the U.S.–Israel Binational Science Foundation.

[1] For notational simplicity, we assume that $A \cap B = \emptyset$. Our algorithm works even otherwise.

$A, B \subset \mathbb{R}^d$ and the L_1 and L_∞ norms, the running times for the matching and transportation problems can be improved to $n^2 \log^{O(d)} n$ and $n^2 \log^{O(d)} n \log U$, respectively [14, 4].

For arbitrary weighted bipartite graphs, there are no known approximation algorithms that are more efficient than the algorithm in [7]. For point sets $A, B \subset \mathbb{R}^2$ and for the L_2 norm, Agarwal and Varadarajan [15] show that an ε-approximate matching can be computed in $O((n/\varepsilon)^{3/2} \log^5 n)$ time. In [2], Agarwal and Varadarajan present a Monte Carlo algorithm for computing an $O(\log(1/\varepsilon))$-approximate matching in time $O(n^{1+\varepsilon})$. Building on the ideas of Agarwal and Varadarajan, Indyk [9] presents an algorithm that estimates in $O(n \operatorname{polylog}(n))$ time, with probability at least $1/2$, a cost that is at most $O(1)$ times the cost of the optimal matching. It, however, does not return such a matching. See also [12].

For unweighted graphs, Onak and Rubinfield [10] present a fully dynamic data structure that maintains, for a large constant c, a $1/c$-approximation of maximum cardinality matching under insertions and deletions of edges in $O(\log^2 n)$ amortized time. Baswana *et al.* [5] improve the amortized update time to $O(\log n)$ and the approximation ratio to $1/2$. For bipartite graphs induced by a bounded integer grid $[\Delta]^d$, there is a streaming algorithm that maintains, under insertions and deletions, a matching whose expected cost is at most $O(\log \Delta)$ times the cost of the optimal matching. Andoni *et al.* [3] present a streaming algorithm that maintains, in $O(\Delta^\varepsilon)$ space and update time, a cost whose expected value is within $O(1/\varepsilon)$ times the cost of the optimal matching.

Our results. For $A \subseteq \mathbb{R}^d$, $|A| = n$ and with every $a \in A$ having a weight w_a, suppose there is a dynamic nearest-neighbor data structure under $\mathtt{d}(\cdot,\cdot)$ that allows insertion and deletion of points for a query point q returns the weighted nearest neighbor, i.e. a point $p = \operatorname{argmin}_{p' \in A} \mathtt{d}(p', q) - w_{p'}$. Let $\Phi(n)$ be the query and update time of such a data structure. We assume that $\Phi(n) = \Omega(\log n)$. In this paper, we obtain the following results.

- Suppose the diameter of $A \cup B$ is Δ under $\mathtt{d}(\cdot,\cdot)$. For any $\varepsilon > 0$, we present an algorithm that computes an ε-close assignment in time $O((n\sqrt{U} \log^2 n + U \log U)\Phi(n) \log(\Delta U/\varepsilon))$. This algorithm implies the following results:
  - If $A, B \subset \mathbb{R}^d$ and $\mathtt{d}(\cdot,\cdot)$ is a metric, then for $\varepsilon > 0$ an ε-approximate assignment for A, B can be computed in time $O((n\sqrt{U} \log^2 n + U \log U)\Phi(n) \log(n/\varepsilon))$. If $\mathtt{d}(\cdot,\cdot)$ is L_1, L_2 or L_∞ norms and $U \leq n^{2-\delta}$ for some constant $\delta > 0$, then the running time of our algorithm is $o(n^2)$. For the special case of matching, $d = 2$ and $\mathtt{d}(\cdot,\cdot)$ being the L_p norm, the running time of this algorithm is similar to that in [15] as a function of n, but the latter has a factor $1/\varepsilon^{3/2}$ in its running time while ours has $\log(1/\varepsilon)$.
  - If $A, B \subset [\Delta]^d$ are sets of grid points and $\mathtt{d}(\cdot,\cdot)$ is L_1 or L_∞ norm, an optimal matching can be computed in time $O(n^{3/2} \log^{d+O(1)} n \log \Delta)$. As far as we know, this is the first subquadratic algorithm for the matching problem under L_1, L_∞ norms even when points lie on a bounded grid.
  - If $A, B \subset [\Delta]^2$ and $\mathtt{d}(\cdot,\cdot)$ is the RMS[2] distance, the above algorithm computes an optimal matching in time $O(n^{3/2+\delta} \log \Delta)$ for an arbitrarily small constant $\delta > 0$.
- If $A, B \subseteq [\Delta]^d$ and $\mathtt{d}(\cdot,\cdot)$ is L_p norm, we present a randomized dynamic data structure of size $n(\log \Delta \log U/(\alpha\beta))^{O(d)}$ that maintains an (α, β)-*assignment* σ such that $\sum_{b \in B} \sigma(a, b) \leq d_a$, $\sum_{a \in A} \sigma(a, b) \leq s_b$, $\sum_{a \in A, b \in B} \sigma(a, b) \geq (1 - \alpha)U$ and the cost of σ, $\mathtt{w}(\sigma)$, is at most $(1 + \beta)\mathtt{w}(\sigma_{\mathrm{OPT}})$ with high probability. The update time of our data structure is $(\log(n\Delta)/(\alpha\beta))^{O(d)}$, provided $U \leq n^{O(1)}$.

The rest of the paper is organized as follows. In Section 2, we introduce basic definitions and give a brief overview of the Gabow-Tarjan algorithm for computing optimal bipartite matching. In Section 3, we describe our algorithm for transportation problem. In Section 4, we describe dynamic data structure that maintains an (α, β)-assignment.

2 Preliminaries

Given a matching M on $\mathcal{G}(A, B)$, an *alternating path* (or cycle) is a simple path (resp. cycle) whose edges are alternately in and not in M. A *free vertex* is a vertex that has not been matched. An *augmenting path* P is an alternating path between two free vertices. We can *augment* M by one edge along P if we remove $P \cap M$ from M and add $P \setminus M$ to M.

For every vertex v in $\mathcal{G}(A, B)$, let $y(v)$ be its dual weight. A 1-*feasible matching* consists of a matching M and set of dual weights $y(v)$ such that for every edge between $u \in A$ and $v \in B$ we have

[2]Under the RMS distance, the distance between a and b is $||a - b||^2$ where $||a - b||$ is the Euclidean distance between a and b.

Algorithm ScaleMatch(A, B)

1: $\forall u \in A \cup B$, $y(u) \leftarrow 0$,
   $\forall (u,v) \in A \times B$, $\mathtt{d}'(u,v) \leftarrow 0$, $M = \emptyset$
2: **for** $i = 1$ to k **do**
3: $\quad \forall (u,v) \in E$, $\mathtt{d}'(u,v) \leftarrow 2\mathtt{d}'(u,v) + (b_i \text{ of } \mathtt{d}^*(u,v))$
4: $\quad \forall t \in A \cup B$, $y(t) \leftarrow 2y(t) - 1$
5: $\quad (M, y) =$ Match$(A, B, \mathtt{d}', y)$
6: **end for**
7: **return** M

Algorithm Match$(A, B, \mathtt{d}', y)$

1: $M = \emptyset$
2: **repeat**
3: $\quad \mathcal{P} = \text{DFS}(A, B, \mathtt{d}', y)$
4: $\quad M =$Augment$(\mathcal{P})$
5: $\quad \forall P \in \mathcal{P}, \forall p \in P \cap B$, $y(p) \leftarrow y(p) - 1$
6: $\quad y \leftarrow$ HungarianSearch$(A, B, \mathtt{d}', y)$
7: **until** $|M| = n$
8: **return** M, y

Figure 1: Gabow-Tarjan matching algorithm.

$$y(u) + y(v) \leq \mathtt{d}(u,v) + 1, \tag{2.1}$$

$$y(u) + y(v) = \mathtt{d}(u,v) \quad \text{for } (u,v) \in M. \tag{2.2}$$

The above conditions with the $+1$ removed from (2.1) are identical to the dual feasibility conditions of the linear program corresponding to optimal bipartite matching. A 1-*optimal matching* is a perfect matching that is 1-feasible. The following lemma relates 1-optimal matching to the optimal matching.

Lemma 2.1. [7] *For a bipartite graph $\mathcal{G}(A, B)$ with integer edge costs, let M be a 1-optimal matching and M_{OPT} be the optimal matching. Then, $\mathtt{w}(M) \leq \mathtt{w}(M_{\text{OPT}}) + n$.*

For every $a \in A, b \in B$, let $\mathtt{d}^*(a,b) = (n+1)\mathtt{d}(a,b)$. Such a uniform scaling of edge costs does not change the optimal matching on $\mathcal{G}$. Lemma 2.1 implies that a 1-optimal matching of points in A, B with edge weights $\mathtt{d}^*(\cdot,\cdot)$ corresponds to an optimal matching with the original edge weights $\mathtt{d}(\cdot,\cdot)$. Now we briefly describe the scaling algorithm (See Figure 1).

Each iteration of the loop is called a *scale*. For any edge (u,v), let $b_1, b_2 \ldots b_k$ be the binary representation of $\mathtt{d}^*(u,v)$. In the ith iteration, the cost of an edge, $\mathtt{d}'(u,v)$, corresponds to the most significant i bits of $\mathtt{d}^*(u,v)$. An edge $(u,v) \notin M$ is called *admissible* if $y(u) + y(v) = \mathtt{d}'(u,v) + 1$. An *admissible graph* is the union of the set of admissible edges and edges in M.

Algorithm Match takes as input a complete bipartite graph on A, B, with a cost function $\mathtt{d}'(\cdot,\cdot)$, and a set of dual weights $y(v)$ for every point $v \in A \cup B$. It returns a 1-optimal matching. In each step, the algorithm finds a maximal set $\mathcal{P}$ of vertex-disjoint augmenting paths in the admissible graph by doing a depth first search. Then the matching is augmented along every path $P \in \mathcal{P}$ in the admissible graph. The dual weights are adjusted in order to maintain 1-feasibility. Since the resulting admissible graph does not have any augmenting paths, the algorithm does a Hungarian Search by adjusting the duals and finds an augmenting path.

Note at the beginning of each scale, the residual costs of the edges, i.e., $\mathtt{d}'(u,v) - y(u) - y(v) \geq -1$ and the residual cost of the optimal matching is $O(n)$. Using the fact that in each iteration Hungarian search increases the dual weight of every free vertex by at least one, Gabow-Tarjan show that Algorithm Match iterates only $O(\sqrt{n})$ times, and that the total length of all the augmenting paths found it is $O(n \log n)$. Since its each iteration runs in $O(n^2)$ time and it is invoked $O(\log n)$ times by ScaleMatch, the overall running time is $O(n^{2.5} \log n)$.

3 The Transportation Algorithm

For $A, B \subset \mathbb{R}^d$, consider an instance of transportation problem as defined in Section 1. Under the assumption that the diameter of $A \cup B$ is bounded by Δ, we present an algorithm for computing an ε-close assignment. We assume Δ is of the form 2^i for some integer i An instance of the transportation problem can be converted to an instance of the bipartite matching problem as follows. For every point $b \in B$ (resp. $a \in A$), let S_b (S_a) be a multiset of s_b(resp d_a) identical copies of b (resp. a). Let $S_A = \bigcup_{a \in A} S_a$ and $S_B = \bigcup_{b \in B} S_b$. Using the Gabow-Tarjan algorithm, we describe an algorithim for computing an ε-close matching on multisets S_A, S_B in time $O((n\sqrt{U}\log^2 n + U \log U)\Phi(n) \log(U\Delta/\varepsilon))$. Such an ε-close matching on $\mathcal{G}(S_A, S_B)$ is an ε-close assignment on $\mathcal{G}(A, B)$.

Let *δ-scaled graph*, $\mathcal{G}_\delta(S_A, S_B)$, be the graph identical to $\mathcal{G}(S_A, S_B)$, except that the weight of every edge (u,v) in $\mathcal{G}_\delta$ is $\mathtt{d}_\delta(u,v) = \left\lceil \frac{\mathtt{d}(u,v)}{\delta} \right\rceil$. Note that all the edge weights in $\mathcal{G}_\delta$ are integers. Lemma 3.1, whose proof is similar to that of Lemma 2.1, shows that a 1-optimal matching on $\mathcal{G}_\delta$, for $\delta \leq \varepsilon/3U$, is an ε-close matching on $\mathcal{G}$.

Lemma 3.1. *Let M be a 1-optimal matching on $\mathcal{G}_\delta(S_A, S_B)$, where $\delta < \varepsilon/3U$ and let M^* be the opti-*

mal matching on $\mathcal{G}(S_A, S_B)$, then M is ε-close to M^* in $\mathcal{G}(S_A, S_B)$.

In Figure 2, Algorithm ScaleMatch describes the scaling routine of our algorithm. In each iteration of ScaleMatch a 1-optimal matching of $\mathcal{G}_\delta(S_A, S_B)$ is computed using 1-OptimalMatch, which takes as input the multisets S_A and S_B, scaling factor δ and associated dual weights satisfying the 1-feasibility. For efficiency reasons, S_A, S_B and a 1-feasible matching M are represented compactly. We first describe the compact representations of S_A, S_B and a 1-feasbile matching M of S_A, S_B.

Compact representation of 1-feasible matching. A *compact representation* K^* of a multiset $K \subset S_A \cup S_B$ and associated dual weights is the set of weighted pairs (v, y) where $v \in K$ and $y(v) = y$; its weight $w(v, y)$ is the number of copies of $v \in K$ that have an associated dual weight y. We will refer to the pairs in K^* as tuples. A compact representation $\mathcal{M}$ of a 1-feasible matching M is a set of edges on vertex sets S_A^* and S_B^*. Any edge $((a, y), (b, y')) \in \mathcal{M}$ with weight $w((a, y), (b, y')) = k$ if there are exactly $k \geq 1$ edges in M between copies of a with associated dual weight y and copies of b with associated dual weight y'. We can also view $\mathcal{M}$ as a bipartite graph with S_A^* and S_B^* as vertex sets. A tuple $(a, y) \in S_A^*$ is a *deficit* tuple with a deficit of $w'(a, y)$ if there are exactly $w'(a, y) \geq 1$ free copies of a in S_A with associated dual weight y. Similarly $(b, y') \in S_B^*$ is a *surplus* tuple with a surplus of $w'(b, y')$ if there are exactly $w'(b, y')$ free copies of b in S_B with an associated dual weight y'. Let K_1^* and K_2^* be compact representations of multisets K_1 and K_2. We can perform set operations on compact representation of multisets. The compact representation $K^* = K_1^* \cup K_2^*$ of the union of two multisets K_1 and K_2 contains all tuples $(p, y) \in K_1^*, K_2^*$. If (p, y) appears in only one of K_1^* and K_2^*, then its weight is the same as in that set; otherwise, its weight is the sum of its weights in the two sets. A compact representation $K^* = K_1^* - K_2^*$, of set difference of two multisets K_1, K_2 contains tuples $(p, y) \in K_1^*$. For each $(p, y) \in K_2^*$ with a weight w, suppose $(p, y) \in K_1^*$. We reduce the weight of $(p, y) \in K^*$ by w. If the new weight is not positive, we remove (p, y) from K^*.

Overview of the algorithm. We describe the algorithm for computing a 1-optimal matching. Algorithm 1-OptimalMatch maintains on M and the dual weights of S_A and S_B the following invariants.

(I1) M is a 1-feasible matching,

(I2) $\forall v \in S_A \cup S_B$, $y(v)$ is an integer,

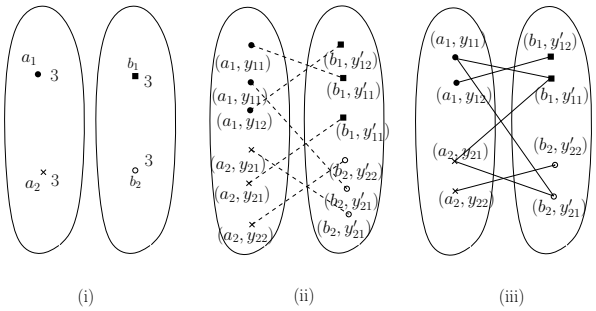

Figure 3: (i) A, B along with their demand and supply values; (ii) S_A, S_B and a matching M; (iii) compact representation S_A^*, S_B^* and $\mathcal{M}$.

.

(I3) $|S_A^*| \leq 3n$, $|S_B^*| \leq n$ and $|\mathcal{M}| \leq 4n$.

(I4) The admissible graph on S_A and S_B does not contain any alternating cycle.

Lines 3, 4, 5 of 1-OptimalMatch are identical to Gabow-Tarjan algorithm. Line 5 augments the current matching and constructs a new matching M', but its compact representation $\mathcal{M}'$ may violate (I3). In Line 6, we compute another 1-feasible matching M such that $\mathcal{M}$ is a forest and satisfies (I3). Such a forest representation is crucial for efficient implementations of Lines 3 and 4. First, we describe how Lines 3, 4 and 6 are implemented efficiently, then we prove the correctness of the algorithm. For clarity, we first describe the steps on multisets S_A and S_B, and then describe how to implement them using their compact representations.

3.1 Implementing Hungarian Search

Hungarian search involves searching for an augmenting path in the admissible graph. For each free vertex $q \in S_B$, we grow an alternating tree whose paths are alternating paths starting at q. Each point in an alternating tree is reachable from its root by an alternating path that consists only of admissible edges. Let S (resp. T) denote the points of S_B (resp. S_A) that lie in some alternating tree. In the beginning of Hungarian search, S is a set of free vertices of S_B and $T = \emptyset$. Let

$$\alpha = \min_{p_i \in S_A - T, q_j \in S} \{\mathtt{d}_\delta(p_i, q_j) + 1 - y(p_i) - y(p_j)\}.$$

At each step, the search takes one of the following actions, depending on whether $\alpha = 0$ or $\alpha > 0$.

- *Case 1:* $\alpha > 0$. The dual weight is updated as follows. For each vertex $p_i \in T$, $y(p_i) \leftarrow y(p_i) - \alpha$ and for each $q_j \in S$, $y(q_j) \leftarrow y(q_j) + \alpha$.

Algorithm ScaleMatch(A, B)

$\delta = \Delta$ $\forall v \in S_A \cup S_B$ $y(v) \leftarrow 0$
repeat
  $(\mathcal{M}, y) \leftarrow$ 1-OptimalMatch(S_A, S_B, δ, y)
  $\delta \leftarrow \delta/2$
  $\forall v \in S_A \cup S_B$ $y(v) \leftarrow 2(y(v) - 1)$
until $\delta \leq \varepsilon/3U$
return $\mathcal{M}$

Algorithm 1-OptimalMatch(S_A, S_B, δ, y)

$G \leftarrow \mathcal{G}_\delta(S_A, S_B)$, $\mathcal{M} = \emptyset$
repeat
  $y \leftarrow$ HungarianSearch(G, y)
  $\mathcal{P} \leftarrow$ DFS(G, y)
  $\mathcal{M}' \leftarrow$Augment$(\mathcal{P})$
  $\mathcal{M} \leftarrow$Acyclify$(\mathcal{M}')$
until $\mathcal{M}$ is perfect
return $\mathcal{M}, y$

Figure 2: Our transportation algorithm.

- *Case 2:* $\alpha = 0$. Grow the alternating tree. Let (p_i, q_j), $p_i \in S_A - T, q_j \in S$ be the admissible edge. If p_i is unmatched, we have found an augmenting path and the search ends. Otherwise, let (p_i, q_k) be the corresponding edge in the matching. We add (p_i, q_j) and (p_i, q_k) to the alternating tree.

The search repeats these steps until an augmenting path is found. We now describe how this step is performed using the compact representation of S_A and S_B.

We obtain compact representations S^* and T^* of S and T from $\mathcal{M}$. For every surplus tuple $(b, y) \in S_B^*$ with a surplus of $w'(b, y)$, we add a tuple (b, y) with a weight of $w'(b, y)$ in S^*. $T^* = \emptyset$. Instead of explicitly updating dual weights at each step, we use the following approach suggested by Vaidya [14]. We maintain a variable ω (initialized to 0). We implicitly maintain the dual weights of vertices in S^* and T^* as follows. Namely we maintain the tuples $(v, \hat{y}) \in S_A^* \cup S_B^*$. If $(v, \hat{y}) \notin S^* \cup T^*$, then $\hat{y} = y(v)$. If $(v, \hat{y}) \in T^*$, then $\hat{y} = y(v) + \omega$, and if $(v, \hat{y}) \in S^*$ then $\hat{y} = y(v) - \omega$. At any time during the search it can be shown that

$$\alpha = \min_{(p_i, \hat{y}_i) \in S_A^* - T^*, (q_j, \hat{y}_j) \in S^*} \{\mathtt{d}_\delta(p_i, q_j) - \hat{y}_i - \hat{y}_j\} - \omega + 1.$$

- *Case 1:* $\alpha > 0$. Update $\omega \leftarrow \omega + \alpha$ (Note that we do not update any tuples in $S^* \cup T^*$).

- *Case 2:* $\alpha = 0$. Grow the alternating tree. Let $((p_i, \hat{y}), (q_j, \hat{y}'))$, $(p_i, \hat{y}) \in S_A^* - T^*, (q_j, \hat{y}') \in S^*$ be the edge for which $\alpha = 0$. All copies of $p_i \in S_A - T$ with dual y form an admissible edge with every copy $q_j \in S$ with dual y'. So, we add $(p_i, \hat{y} + \omega)$ to T^*. Its weight in T^* is the same as the weight of (p_i, y) in S_A^*. If (p_i, y) is a deficit tuple, an augmenting path is found and the search ends. Otherwise, let $N^* = \{(q_1, \hat{y}_1 - \omega), \ldots, (q_k, \hat{y}_k - \omega)\}$ be the set of tuples with which $(p_i, \hat{y})$ has an edge in $\mathcal{M}$ and let each $(q_h, \hat{y}_h) \in N^*$ have a weight $w((p_i, \hat{y}), (q_h, \hat{y}_h))$. We set $S^* \leftarrow N^* \cup S^*$.

The search repeats these steps until an augmenting path has been found.

Once the search for an augmenting path has ended, we update S_A^*, S_B^* and $\mathcal{M}$ to reflect the true dual weights. For each tuple $(p_i, \hat{y}) \in T^*$, $(q_j, \hat{y}') \in S^*$, we set $(p_i, y) \leftarrow (p_i, \hat{y} - \omega)$ and $(q_j, y) \leftarrow (q_j, \hat{y} + \omega)$. The edges of $\mathcal{M}$ that were not in any alternating tree do not change. Every edge $((p, \hat{y}), (q, \hat{y}'))$ in $\mathcal{M}$ that is in the alternating tree is now an edge between points $((p, \hat{y} - \omega), (q, \hat{y} + \omega))$. The weights of these edges remain the same. This completes the description of our implementation of Hungarian Search.

During each step of the search, the value of α can be computed by maintaining weighted bichromatic closest pair between the sets $S_A^* - T^*$ and S^* under $\mathtt{d}_\delta(\cdot, \cdot)$. When a tuple $(p, \hat{y})$ is added to T^*, we delete the point p with weight $\hat{y}$ and when a tuple $(p, \hat{y})$ is added to S^* we add p with a weight $\hat{y}$ to the data structure. The following lemma shows that to compute α it suffices to compute a weighted closest pair under $\mathtt{d}(\cdot, \cdot)$.

Lemma 3.2. *Let* $((p, \hat{y}), (q, \hat{y}')) =$

$$\text{argmin}_{(p_i, \hat{y}_i) \in S_A^* - T^*, (q_j, \hat{y}_j) \in S^*} \{\mathtt{d}(p_i, q_j) - \delta\hat{y}_i - \delta\hat{y}_j\}.$$

Then $\alpha = \{\mathtt{d}_\delta(p, q) - \hat{y} - \hat{y}'\} - \omega + 1$.

Proof. Since adding or multiplying a constant and taking a ceiling of a fraction does not affect the minimum, we have $((p, \hat{y}), (q, \hat{y}')) =$

$$\underset{(p_i, \hat{y}_i) \in S_A^* - T^*, (q_j, \hat{y}_j) \in S^*}{\text{argmin}} \{\mathtt{d}(p_i, q_j) - \delta(\hat{y}_i + \hat{y}_j + \omega - 1)\}.$$

$=$

$$\underset{(p_i, \hat{y}_i) \in S_A^* - T^*, (q_j, \hat{y}_j) \in S^*}{\text{argmin}} \left\lceil \frac{\mathtt{d}(p_i, q_j) - \delta(\hat{y}_i + \hat{y}_j + \omega - 1)}{\delta} \right\rceil.$$

$=$

$$\underset{(p_i, \hat{y}_i) \in S_A^* - T^*, (q_j, \hat{y}_j) \in S^*}{\text{argmin}} \left\{ \left\lceil \frac{\mathtt{d}(p_i, q_j)}{\delta} \right\rceil - \hat{y}_i - \hat{y}_j \right\} - \omega + 1.$$

The last equality follows from (I2) and the fact that ω is an integer and thus all $\hat{y}$'s are integers. But, $\alpha =$

$$\min_{(p_i,\hat{y}_i)\in S_A^*-T^*,(q_j,\hat{y}_j)\in S^*} \left\{ \left\lceil \frac{\mathtt{d}(p_i,q_j)}{\delta} \right\rceil - \hat{y}_i - \hat{y}_j \right\} - \omega + 1.$$

From the above two equations, it follows that,

$$\alpha = \left\{ \left\lceil \frac{\mathtt{d}(p,q)}{\delta} \right\rceil - \hat{y} - \hat{y}' \right\} - \omega + 1.$$

Initializing $S^*, T^*, S_A^* - T^*$ and $S_B^* - S^*$ takes $O(|S_A^*| + |S_B^*|) = O(n)$ time. Growing the alternating tree (when $\alpha = 0$) involves adding each $(p, y) \in S_A^*, ((p, y), (q, y')) \in \mathcal{M}$ to the alternating tree at most once. Thus Hungarian Search takes $O(|S_A^*|+|\mathcal{M}|)$ steps, which can be bounded by $O(n)$ using (I3). Executing each step requires us to compute a weighted bichromatic closest pair. As shown by Eppstein [6], dynamic weighted nearest neighbor data structure can be used to maintain dynamic weighted bichromatic closest pair for point sets in $\mathbb{R}^d$. The update and query times is $O(\Phi(n)\log^2 n)$. Hence, the total time taken by Hungarian search is $O(n\Phi(n)\log^2 n)$. Finally, the time taken to update S_A^*, S_B^* and $\mathcal{M}$ in order to reflect the correct dual weight is $O(S_A^* + S_B^* + |\mathcal{M}|) = O(n)$.

3.2 Implementing Depth First Search We perform depth first search to find a maximal set of vertex disjoint augmenting paths $\mathcal{P}$ as follows. An alternating path P is initiated from a free vertex of S_B. The search marks every visited vertex. At each step, the last vertex on P is a vertex v of S_B. We grow P by scanning all admissible edges adjacent to v and finding an edge (v, u) such that u is unmarked; (v, u) is added to P. If u is free, P is an augmenting path, we add P to $\mathcal{P}$ and start a new alternating path. Otherwise, if u is matched to $u' \in S_B$, we add (u, u') to the path and continue with the search from u'. If no such u exists, then v and its predecessor in P are removed from P. The search continues until a path has been initiated from every free vertex in S_B. This completes the description of the depth first search. We now give an efficient implementation of this procedure using $\mathcal{M}$.

We maintain three sets $C^* \subseteq S_A^*$, $Q^* \subseteq S_B^*$ and $L^* \subseteq \mathcal{M}$, which correspond to the vertices of $S_A \cup S_B$ and the edges of $\mathcal{M}$ that have not been explored. By the depth first procedure initially $C^* = S_A^*$, $Q^* = S_B^*$ and $L^* = \mathcal{M}$. For a surplus tuple $(q_1, y_1) \in Q^*$, we initiate a path P and reduce the surplus of (q_1, y_1) in Q^* by 1. Suppose currently $P = \langle (q_1, y_1), (p_1, y_1'), (q_2, y_2) \ldots, (p_k, y_k'), (q_k, y_k)\rangle$. To grow P, we compute

$$\eta(q_k, y_k) = \min_{(p,y)\in C^*} \{\mathtt{d}_\delta(q_k, p) + 1 - y - y_k\}.$$

- *Case 1:* $\eta(q_k, y_k) = 0$. Let $((q_k, y_k), (p_k, y_k'))$ be the admissible edge. We add (p_k, y_k') to the path P and reduce the weight of (p_k, y_k') in C^* by 1. If (p_k, y_k') is a deficit tuple, then P is an augmenting path. We add P to $\mathcal{P}$ and initiate a new path. On the other hand, if (p, y') is not a deficit tuple, then L^* has an edge $((p_k, y_k'), (q_{k+1}, y_{k+1}))$. We add (p_k, y_k') and (q_{k+1}, y_{k+1}) to P. We reduce the weight of (q_{k+1}, y_{k+1}) in Q^* and $((p_k, y_k'), (q_{k+1}, y_{k+1}))$ in L^* by 1 and continue our search.
- *Case 2:* $\eta(q_k, y_k) > 0$. There is no admissible edge from (q_k, y_k). We remove (q_k, y_k) from Q^* and all edges incident on (q_k, y_k) from L^*. If there is an edge $((p_{k-1}, y_{k-1}'), (q, y)) \in L^*$, we add it to P and continue. Otherwise, we remove (p_{k-1}, y_{k-1}') from C^* and continue our search from (q_{k-1}, y_{k-1}).

The search stops when no surplus tuple is left in Q^*. At each step of the procedure, we use a dynamic nearest-neighbor data structure to compute $\eta(q_k, y_k)$. As in Lemma 3.2, we can show that if

$$(p, y) = \operatorname*{argmin}_{(p',y')\in C^*} \{\mathtt{d}(q_k, p') - \delta y_k - \delta y' + \delta\}$$

then

$$\eta(q_k, y_k) = \mathtt{d}_\delta(q_k, p) + 1 - y_k - y.$$

We can thus construct a weighted nearest-neighbor query data structure on C^* under $\mathtt{d}(\cdot,\cdot)$ to compute $\eta(q_k, y_k)$. When a point is deleted from C^*, it is also deleted from the data structure. Hence, finding $\eta(q_k, y_k)$ and updating the data structure takes $\Phi(n)$ time.

The following lemma will be critical to analyze the running time of depth first search procedure.

LEMMA 3.3. *Let π be an alternating path constructed by the search procedure. Then each tuple $(v, y) \in S_A^* \cup S_B^*$ appears at most once in π.*

Proof. Suppose two copies of $(v, y) \in S_B^*$ lie on π. Let $\chi = \langle (v, y) = (v_1, y_1), (v_2, y_2) \ldots, (v_k, y_k), (v, y)\rangle$ be the portion of the alternating path of even length between the two copies of (v, y). Suppose $((v_1, y_1), (v_2, y_2)), ((v_3, y_3), (v_4, y_4)), \ldots, ((v_{k-1}, y_{k-1}), (v_k, y_k))$ are the matching edges of $\mathcal{M}$. Let $\overline{v}_1, \overline{v}_2, \ldots, \overline{v}_k$ be the copies of $v_1, \ldots, v_k$ with dual weights $y_1 \ldots, y_k$ corresponding to these matching edges. Note that for every non-matching edge $((v_i, y_i), (v_{i+1}, y_{i+1}))$ in χ is admissible. Furthermore, for every copy $\tilde{v}_i$ of v_i with dual weight y_i and every copy $\tilde{v}_{i+1}$ of v_{i+1} with dual weight y_{i+1}, the corresponding edge $(\tilde{v}_i, \tilde{v}_{i+1})$ is also admissible. In particular $((\overline{v}_k, y_k), (\overline{v}_1, y_1))$ is also admissible implying that $\overline{v}_1, \overline{v}_2, \ldots, \overline{v}_k, \overline{v}_1$ is an alternating cycle in the admissible graph of S_A, S_B, which contradicts (I4). Hence, the lemma is true.

The time spent by the search procedure on computing a path initiated from a surplus tuple $(v, y) \in S_B^*$ is proportional to the number of vertices visited by the procedure plus the number of edges of L^* deleted in Case 2. By Lemma 3.3, each tuple is visited at most once and futhermore every tuple not appearing on the augmenting path from (v, y) is deleted from Q^* or C^*. Each tuple of $Q^* \cup C^*$ and each edge of L^* is deleted at most once. Hence, the total time spent by the procedure in constructing $\mathcal{P}$ is $O((|S_A^*|+|S_B^*|+|\mathcal{M}|+\mu)\Phi(n))$ where μ is the total length of all the augmenting paths in $\mathcal{P}$. By (I3), this quantity is $O((n+\mu)\Phi(n))$.

3.3 Augmentation and Acyclify Now we describe Lines 5 and 6 of the algorithm. Line 5 generates the compact representation $\mathcal{M}'$ of a new matching M' by augmenting the current matching M and updating the dual weights of every vertex $u \in S_B$ that appears in an augmenting path $P \in \mathcal{P}$. For every $(b, y) \in S_B^* \cap P$, we reduce the weight of (b, y) by 1 and increase the weight of $(b, y-1)$ by 1. If the weight of (b, y) becomes 0, we delete it; similarly if there was no tuple $(b, y-1)$ in S_B^* we add it and set its weight to 1. Furthermore, we add 1 to the weights on the new edges that enter the matching and remove 1 from the edges that go out of the matching.

The compact representation $\mathcal{M}'$ of the new matching has $O(n+\mu)$ edges, where $\mu = \sum_{P\in\mathcal{P}} |P|$. Next, the Acyclify procedure converts this 1-feasible matching M' and its compact representation $\mathcal{M}'$ to another 1-feasible matching M'' and its compact representation $\mathcal{M}''$ such that $|M'| = |M''|$ and $\mathcal{M}''$ does not have any cycles, so $|\mathcal{M}''| = O(n)$ We set $\mathcal{M}$ to $\mathcal{M}''$. We order the edges of $\mathcal{M}'$ in an arbitrary order. Let $\gamma_1, \ldots, \gamma_u$ be the resulting sequence of edges, and let $w(\gamma_i)$ denote the weight of γ_i in $\mathcal{M}'$. For $k \le u$, let $\Gamma_k = \{\gamma_1, \ldots, \gamma_k\}$ and $\mathbb{G}_k = (S_A^* \cup S_B^*, \Gamma_k)$. We will maintain a subgraph $\mathbb{F}_k = (S_A^* \cup S_B^*, \mathcal{M}_k)$, where $\mathcal{M}_k \subseteq \Gamma_k$ and $\mathbb{F}_k$ is a forest. For each $\gamma \in \mathcal{M}_k$, we maintain a weight $h(\gamma) \ge 0$ such that

$$\forall \xi \in S_A^*, \sum_{(\xi,\eta)\in\mathcal{M}_k} h(\xi,\eta) = \sum_{(\xi,\eta)\in\Gamma_k} w(\xi,\eta), \tag{3.3}$$

$$\forall \eta \in S_B^*, \sum_{(\xi,\eta)\in\mathcal{M}_k} h(\xi,\eta) = \sum_{(\xi,\eta)\in\Gamma_k} w(\xi,\eta). \tag{3.4}$$

In particular, $\sum_{\gamma\in\Gamma_k} w(\gamma) = \sum_{\gamma\in\mathcal{M}_k} h(\gamma)$. Then $\mathcal{M}_u$ gives the comapact representation of the desired matching M'' since $\mathbb{F}_u$ is a forest. We now describe how to maintain $\mathbb{F}_k$.

Initially, $\mathcal{M}_0 = \emptyset$. We compute $\mathbb{F}_{k+1}$ from $\mathbb{F}_k$ as follows. If $\mathcal{M}_k \cup \{\gamma_{k+1}\}$ does not contain a cycle, we set $\mathcal{M}_{k+1} = \mathcal{M}_k \cup \{\gamma_{k+1}\}$ and $h(\gamma_{k+1}) = w(\gamma_{k+1})$.

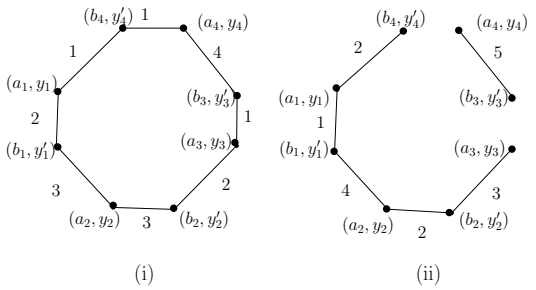

Figure 4: (i) $|M'| = 17$ and $\mathcal{M}$ has a cycle; (ii)$|M''| = 17$ and $\mathcal{M}''$ is acyclic.

Otherwise, it contains an even length cycle Σ and γ_{k+1} is an edge in Σ. We set $h(\gamma_{k+1}) = w(\gamma_{k+1})$. We compute the minimum-weight edge e of Σ, i.e. $e = \text{argmin}_{e'\in\Sigma} h(e')$. We label the edges of Σ so that $\Sigma = \langle e = e_1, e_2, \ldots, e_{2s}\rangle$. For $1 \le j \le s$, we set $h(e_{2j-1}) \leftarrow h(e_{2j-1}) - h(e)$ and $h(e_{2j}) \leftarrow h(e_{2j}) + h(e)$. We set $\mathcal{M}_{k+1} = \mathcal{M}_k \cup \{\gamma_{k+1}\} \setminus \{e\}$. By construction $\mathbb{F}_{k+1}$ is a forest. Since the weight of alternating edges in Σ is increased and decreased by $h(e)$, it can be verified that $\mathbb{F}_{k+1}$ satisfies (3.3) and (3.4).

At each step, we perform three operations on $\mathbb{F}_k$: (i) Check whether two vertices $\xi \in S_A^*$ and $\eta \in S_B^*$ lie in the same connected component of $\mathbb{F}_k$, (ii) given a path π in $\mathbb{F}_k$ and a non-negative integer κ, increase the weight of every odd edge and decrease the weight of every even edge by κ, and (iii) insert/delete an edge. Using the dynamic tree data structure by Sleator and Tarjan [13], each of these operations can be performed in $O(\log n)$ amortized time. Hence the total time spent by the Acyclify procedure is $O(\mu \log n)$.

Finally every edge $((u, y), (v, y'))$ in $\mathcal{M}'$, $y + y' = \mathtt{d}_\delta(u, v)$. Since $\mathcal{M}'' \subset \mathcal{M}'$, we can conclude that every edge in $\mathcal{M}''$ is feasible.

3.4 Proof of Correctness We now prove that invariants (I1), (I2), (I3) and (I4) hold.

The algorithm maintains 1-feasibility: Dual weights are modified during the Hungarian Search and the Augementation Step. By construction, Hungarian Search preserves 1-feasibility. The augmentation step only reduces the associated dual weight of points of S_B that lie on some augmenting path. Any reduction of the dual weights of points in S_B does not violate 1-feasibility. Thus (I1) holds. Note that the cost of edges in $\mathcal{G}_\delta$ are integers. Initially each dual weight is set to 0, and each operation either adds or multiplies an integer to it. Hence all dual weights are always integers, proving (I2).

The next two lemmas prove (I3) and (I4).

LEMMA 3.4. *Each point $b \in B$ has at most three tuples $(b, y_1), (b, y_2)$ (b, y_3) in S_B^* and each point $a \in A$ has one tuple (a, y) in S_A^*.*

Proof. First, we show that for any two copies $(b, y_1), (b, y_2) \in S_B^*$ of b with $y_1 < y_2$, if (b, y_1) is matched, then $y_2 = y_1 + 1$. Suppose $y_2 > y_1 + 1$. Let (a_1, y_1') be the tuple that (b, y_1) is matched to. The feasibility condition (2.2) implies that $y_1 + y_1' = \lceil \mathtt{d}(a_1, b)/\delta \rceil$. Thus for the edge $((b, y_2), (a_1, y_1))$, we have

$$y_2 + y_1' > y_1 + y_1' + 1 = \lceil \mathtt{d}(a_1, b)/\delta \rceil + 1,$$

implying that $((b, y_2), (a_1, y_1'))$ is not 1-feasible and that (I1) is violated.

The above argument implies that when Algorithm 1-OPTIMALMATCH returns a perfect matching, there are at most two copies $(b, y_1), (b, y_2)$ of each vertex $b \in B$ with $y_2 - y_1 = 1$. Line 4 of Algorithm SCALEMATCH doubles the difference between the two copies. Hence, at the start of each scale, we have at most two copies (b, y_1) and $b(y_2)$. If there are two copies, then $y_2 = y_1 + 2$. Next, we show that during the computation of 1-optimal matching, there cannot be more than three copies. Suppose during the course of the algorithm there are four copies of b, $((b, y_1'), (b, y_2'), (b, y_3'), (b, y_4'))$, with $y_1' < y_2' < y_3' < y_4'$, then $y_4' \geq y_1' + 3$ and the above argument implies that (b, y_1) must be free. Let γ be the sum of all the dual changes made by the Hungarian search in this call of 1-OPTIMALMATCH. By the description of Hungarian Search, every free vertex would have undergone a dual change of γ. Since (b, y_1') is free, $y_1' \geq y_1 + \gamma$, and $y_4' \leq y_1 + \gamma + 2 \leq y_1' + 2$ which is a contradiction of our assumption that $y_1' + 2 < y_4'$.

Every vertex $a \in A$ has exactly one tuple $(a, y) \in S_A^*$. At the start of the algorithm, this is true. The only place where points in S_A undergo a dual change is the Hungarian Search. But, our implementation of Hungarian Search ensures that if one copy of (a, y) is part of the alternating tree, then all copies are in the alternating tree. Hence, no two copies will have different dual weights.

$|\mathcal{M}| = O(n)$: This is true at the start of the algorithm. The cardinality of $\mathcal{M}$ does not change during Hungarian Search and Depth First Search. $\mathcal{M}$ is modified by the Acyclify procedure. Since Acyclify generates a forest on S_A^* and S_B^*, $|\mathcal{M}| \leq |S_A^*| + |S_B^*| \leq 4n$.

LEMMA 3.5. *There are no alternating cycles of admissible edges generated by Algorithm* 1-OPTIMALMATCH.

Proof. Initially there are no matched edges. So, there are no alternating cycles of admissible edges. Any cycle can be created only when either the dual weights change or the admissible graph changes. The dual weights change during the Hungarian Search and the Augmentation steps. Using arguments similar to the one in [7], we show that Hungarian search (Line 3 of Algorithm 1-OPTIMALMATCH) does not create an alternating cycle. Suppose it creates a cycle C. There are at least two edges e_1, e_2 in C such that one end point of e_1, e_2 lies in the alternating tree and the other does not lie in tree. By construction, both (e_1, e_2) are not in the current matching. Since C is of even length, one of e_1, e_2, say e_1 is between a vertex $a \in T$ and $b \in S_B - S$. But e_1 cannot become admissible during the Hungarian Search since the dual weight $y(a)$ only reduces in the search making (a, b) inadmissible.

Augmentation (Line 6 of Algorithm 1-OPTIMALMATCH) does not create alternating cycles. Similar to Hungarian search, the dual weights $y(b)$ for $b \in S_B$ will always reduce, hence they cannot create admissible edges.

Line 6 of Algorithm 1-OPTIMALMATCH modifies the edges in the matching and hence modifies the admissible graph. Such a modification does not create alternating cycles. Suppose it does. By construction $\mathcal{M}'' \subseteq \mathcal{M}'$. Hence, if $\mathcal{M}''$ has an alternating cycle then $\mathcal{M}'$ also has one, which is a contradiction.

The algorithm terminates with a 1-optimal matching because Hungarian Search maintains 1-feasibility and ensures the existance of at least one augmenting path each step. The i^{th} iteration of Algorithm 1-OPTIMALMATCH can be implemented in $O((n \log^2 n + |\mu_i|)\Phi(n) + |\mu_i| \log n)$ time, where μ_i is the length of augmenting paths found by the depth first search procedure. Similar to the Gabow-Tarjan algorithm, in each scale the residual costs of the edges corresponding to the 1-optimal matching is $O(n)$ and in each iteration of Algorithm 1-OPTIMALMATCH , the Hungarian Search increases the dual weight of every unmatched tuple by at least 1. Adapting the analysis in [7], we prove that the number of iterations of 1-OPTIMALMATCH is $O(\sqrt{U})$ and $\sum_i |\mu_i| = O(U \log U)$. Furthermore, the total number of times it is invoked $O(\log(U\Delta/\varepsilon))$ times by SCALEMATCH. Putting everything together we obtain the following.

THEOREM 3.1. *Let $A, B \subset \mathbb{R}^d$, with $|A| = |B| = n$ and U being their total demand/supply, let $\mathtt{d}(.,.)$ be a cost function, and let the diameter of $A \cup B$ under $\mathtt{d}(\cdot, \cdot)$ be bounded by Δ. For any $\varepsilon > 0$, we can compute an ε-close assignment in time $O((n\sqrt{U} \log^2 n + U \log U)\Phi(n) \log(\Delta U/\varepsilon))$, where $\Phi(n)$ is the query and update times of a dynamic weighted nearest neighbor data structure.*

3.5 Applications If $A, B \subset \mathbb{R}^d$ and $\mathtt{d}(\cdot,\cdot)$ is a metric cost function, using Theorem 3.1, we show how to compute an ε-approximate assignment on A, B. Let $T = \langle e_1, e_2, \ldots e_{n-1}\rangle$ be the minimum spanning tree of $A \cup B$ labelled in decreasing order of their costs, i.e., $w(e_1) \geq w(e_2) \ldots \geq w(e_{n-1})$. Let C_i be the set of connected components obtained by removing edges $\{e_1, \ldots e_i\}$ from T. For any $C \in C_i$, let A_C and B_C be the points of A and B that are in C. Now, we describe our algorithm for computing ε-approximate assignment of A, B.

First, we compute the smallest index j such that there is a component $C \in C_j$ with $\sum_{a\in A_C} d_a \geq \sum_{b\in B_C} s_b$. Next, we compute C_k where $k < j$ is the largest index with the property $w(e_k) \geq w(e_j)nU$. Note that since $w(e_j) < w(e_k)$, each component $C \in C_k$ has $\sum_{a\in A_C} d_a = \sum_{b\in B_C} s_b$. We compute, for each $C \in C_k$ an $(\varepsilon w(e_i)/n)$-close assignment σ_C of points A_C, B_C under $\mathtt{d}(\cdot,\cdot)$. We claim that $\sigma = \bigcup_{C\in C_k} \sigma_C$ is an ε-approximate assignment on A, B.

Correctness. By construction, there exists a component $C \in C_j$ such that $\sum_{a\in A_C} d_a > \sum_{b\in B_C} s_b$. Hence, at least one unit of demand in C is satisfied by points in other components, the cost of which is at least $w(e_j)$, implying that

$$\mathtt{w}(\sigma_{\mathrm{OPT}}) \geq w(e_j).$$

On the other hand, every component $C \in C_{j-1}$ has equal demands and supply. Also, since $\mathtt{d}(\cdot,\cdot)$ is a metric, the diameter of $A_C \cup B_C$ is at most $w(e_j)n$. Hence, an arbitrary assignment of demands and supply in each component of C_{j-1} will cost at most $nUw(e_j)$ implying that

$$\mathtt{w}(\sigma_{\mathrm{OPT}}) \leq w(e_j)nU.$$

Using the fact that $\mathtt{d}(\cdot,\cdot)$ is a metric, it can be verified that for every component $C \in C_k$, the diameter of $A_C \cup B_C$ is at most $n^2Uw(e_j)$. The smallest distance between any pair of components $C_1, C_2 \in C_k$ is at least $nUw(e_j) \geq \mathtt{w}(\sigma_{\mathrm{OPT}})$. Hence, $\sigma_{\mathrm{OPT}} = \bigcup_{C\in C_k} \sigma_{\mathrm{OPT}}(C)$ where $\sigma_{\mathrm{OPT}}(C)$ is the optimal assignment of A_C, B_C under $\mathtt{d}(\cdot,\cdot)$. By construction,

$$\begin{aligned}
\mathtt{w}(\sigma) &= \sum_{C\in C_k} \mathtt{w}(\sigma_C) \\
&= \sum_{C\in C_k} \mathtt{w}(\sigma_{\mathrm{OPT}}(C)) + \varepsilon w(e_j)/n \\
&\leq \mathtt{w}(\sigma_{\mathrm{OPT}}) + \varepsilon w(e_j) \\
&\leq (1+\varepsilon)\sigma_{\mathrm{OPT}}.
\end{aligned}$$

Using dynamic bichromatic closest pair data structure, we can compute T, C_j and C_k in $O(n\Phi(n)\log^3 n)$ time. From Theorem 3.1, the total time taken for computing σ is $O((n\sqrt{U}\log^2 n + U\log U)\Phi(n)\log(U/\varepsilon))$.

THEOREM 3.2. *Let $A, B \subset \mathbb{R}^d$ with $|A| = |B| = n$ and U being their total demand/supply, let $\mathtt{d}(\cdot,\cdot)$ be a metric cost function, and let $\varepsilon > 0$ be a parameter. An ε-approximate assignment can be computed in time $O((n\sqrt{U}\log^2 n + U\log U)\Phi(n)\log(U/\varepsilon))$, where $\Phi(n)$ is the query and update times of a dynamic weighted nearest neighbor data structure on $\mathtt{d}(\cdot,\cdot)$.*

For point sets $A, B \subset [\Delta]^d$, and let $\mathtt{d}(\cdot,\cdot)$ be the L_1, L_∞ and RMS distance functions. Since the coordinates of every point in A and B are integers, it follows that the cost of any matching in the L_1, L_∞ or RMS is an integer. Hence any 1/2-close matching is optimal. There are dynamic weighted nearest neighbor data structures with $O(\log^d n)$ query and update time for the L_1 and L_∞ norms. From Theorem 3.1, it follows that a 1/2-close matching can be computed in time $O(n^{3/2}\log^{d+O(1)} n \log\Delta)$. For $d = 2$ and the RMS norm, there is a dynamic weighted nearest neighbor data structure with $O(n^\delta)$ query and update time. Hence an optimal matching can be computed in $O(n^{3/2+\delta}\log\Delta)$ time.

COROLLARY 1. *Let $A, B \subset [\Delta]^d$. An optimal matching of A and B under L_1 or L_∞ norm can be computed in time $O(n^{3/2}\log^{d+O(1)} n \log\Delta)$. If $d = 2$, then a minimum-cost matching of A and B under the RMS distance can be computed in time $O(n^{3/2+\delta}\log\Delta)$ for any arbitrarily small constant $\delta > 0$.*

Agarwal *et al.* [1] showed that there is a weighted nearest neighbor data structure under any L_p-norm with $O(n^\delta)$ update time, for arbitrarily small constant $\delta > 0$.

COROLLARY 2. *For any $A, B \subset \mathbb{R}^2$ and for any $\varepsilon > 0$, an ε-approximate assignment of A, B under any L_p norm can be computed in time $O(n\sqrt{U} + U\log U)n^\delta \log U/\varepsilon)$ for an arbitrarily small constant $\delta > 0$.*

4 Dynamic Data Structure for Transportation Problem

Let $A, B \subseteq [\Delta]^d$ be two sets of points. Each point $a \in A$ has a positive integral demand d_a and each point $b \in B$ has a positive integral supply s_b such that $\sum_{a\in A} d_a = \sum_{b\in B} s_b = U$. Let $\mathtt{d}(\cdot,\cdot)$ be the L_p norm. We assume that $U \leq U_{\max} \leq n^{O(1)}$. For given parameters $0 < \alpha, \beta < 1$, we describe a randomized data structure that allows insertion and deletion of points A and B and maintains an (α, β)-assignment σ of A, B with high probability. The size of the data

structure is $n(\log(n\Delta)/(\alpha\beta))^{O(d)}$ and the update time is $\log(n\Delta)/(\alpha\beta))^{O(d)}$. The updates are such that the total demand and supply are always balanced. For simplicity, we describe the data structure for $d = 2$ under the L_1 norm and we assume that the points in $A \cup B$ are disjoint. We scale the input points by a factor of Δ, i.e., every coordinate is multiplied by Δ, so $A, B \subseteq [\Delta^2]^2$.

Before outlining our approach, we introduce a concept, which will be crucial for our data structure. Suppose we run the primal-dual algorithm on A and B, under a distance function $\mathtt{d}(\cdot,\cdot)$. The algorithm maintains a partial assignment σ and dual weights y such that they satisfy the feasiblity condition

$$y(a) + y(b) \leq \mathtt{d}(a, b).$$

That is, y is a dual feasible solution. We call a vertex $a \in A$ a *deficit vertex* if $\sum_{b\in B} \sigma(a, b) < d_a$ and $b \in B$ a *surplus vertex* if $\sum_{a\in A} \sigma(a, b) < s_b$. We call such a solution a *θ-capacitated partial assignment* (θ-CPA for brevity), for a parameter $\theta > 0$ if the following conditions are satisfied.

(C1) $y(b) \leq \theta/(\alpha\beta)$ for all $b \in B$ and $y(b) = \theta/(\alpha\beta)$ if b is a surplus vertex.

(C2) $y(a) \leq 0$, for all $a \in A$ and $y(a) = 0$ if a is a deficit vertex.

The primal-dual algorithm can be adapted to compute θ-CPA. Note that a θ-CPA trivially satisfies (4) for an edge (a, b) with $\mathtt{d}(a, b) > \theta/(\alpha\beta)$ which means $\sigma(a, b) = 0$ for such an edge and thus it can be ignored for computing θ-CPA.

LEMMA 4.1. *Let A, B be an instance of the transportation problem under $\mathtt{d}(\cdot,\cdot)$ and let σ_{OPT} be an optimal assignment. If θ is a parameter such that*

$$\theta U(1/\beta - 1) \leq \mathtt{w}(\sigma_{\mathrm{OPT}}) \leq \theta U/\beta,$$

then a θ-CPA of A, B is an (α, β)-assignment.

Proof. Let σ be a θ-CPA and let y be the corresponding dual weights of points in $A\cup B$. For each $a \in A$, let $d'_a = \sum_{b\in B} \sigma(a, b)$ and for each $b \in B$, let $s'_b = \sum_{a\in A} \sigma(a, b)$. The primal-dual algorithm ensures that

$$\sum_{a\in A} d'_a y(a) + \sum_{b\in B} s'_b y(b) \geq 0. \tag{4.5}$$

Let $F = \sum_{b\in B} s_b - s'_b$. Since y is a dual feasible solution,

$$\sum_{a\in A} d_a y(a) + \sum_{b\in B} s_b y(b) \leq \mathtt{w}(\sigma_{\mathrm{OPT}}) \leq U\theta/\beta.$$

Using (4.5) and conditions (C1) and (C2), we obtain that

$$\sum_{b\in B} (s_b - s'_b) \cdot \theta/(\alpha\beta) \leq U\theta/\beta.$$

Hence, $F \leq \alpha U$, implying that σ is an (α, β)-assignment.

Lemma 4.1 shows that computing an (α, β)-assignment is equivalent to computing a θ-CPA if we know $\mathtt{w}(\sigma_{\mathrm{OPT}})$ approximately. Let

$$\Theta = \{\theta_i = (1 + \beta)^i U\Delta \mid 1 \leq i \leq \lceil \log_{1+\beta} \Delta \rceil\}.$$

Here is the outline of the algorithm: We approximate the L_1-norm with a quad-tree based distance function $\mathtt{d}_Q(\cdot,\cdot)$ and maintain (α, β)-assignment under $\mathtt{d}_Q(\cdot,\cdot)$. Using Lemma 4.1, we reduce this problem to computing a θ-CPA. Since we do not know $\mathtt{w}(\sigma_{\mathrm{OPT}})$, we maintain a θ_i-CPA for each $\theta_i \in \Theta$. Among all θ_i-CPA's that satisfy $(1 - \alpha)$ fraction of the demand, we return the one with the smallest cost. In order to maintain a θ-CPA efficiently, we approximate $\mathtt{d}_Q(\cdot,\cdot)$ by a coarser distance function $\eta(\cdot,\cdot)$ which further approximates the distance between points that are sufficiently close. We also observe that edges of length greater than $2\theta/(\alpha\beta)$ between points in A and B do not play any role in computing a θ-CPA, so we ignore them. This allows us to decompose A, B into a family of subsets of small size so that the θ-CPA can be computed in each subset independently. We now describe the algorithm in detail.

Quad-tree distance. Recall that $A, B \subseteq [\Delta^2]^2$. We choose two random integers $i, j \in [0, \Delta^2]$ and set $G = [0, 2\Delta^2] \times [0, 2\Delta^2] - (i, j)$. G is a randomly-shifted square that contains both A and B. We build a quad-tree Q of height $\log_2(2\Delta^2) = 1 + 2\log_2 \Delta$ on G — the root of Q is associated with G itself and the squares (cells) associated with the children of a node are obtained by splitting the square associated with that node into four equal squares. The nodes at height i induce a grid G_i in which each cell has a side length 2^i. We view Q as the sequence of grids $G_0, G_1, \ldots$; the final grid is G itself with a single cell. For two points $a \in A, b \in B$ we define $\mathtt{d}_Q(a, b)$ as follows: Set

$$\mu = \log_2 \Delta/(8\beta^2).$$

Let C be cell of Q corresponding to the least common ancestor of the leaves of Q containing a and b. We partition C into a $\mu \times \mu$ grid K_C — each subcell in K_C has side length $2^i/\mu$. Let a_C (resp. b_C) be the center point of cells of K_C that contain a (resp. b). We set

$$\mathtt{d}_Q(a, b) = ||a_C b_C||_1,$$

here $||\cdot||$ is the distance in the L_1 norm.

The following lemma, whose proof is omitted, shows that $\mathtt{d}_Q$ approximates L_1-norm in the expected sense.

LEMMA 4.2. *For any pair $a \in A, b \in B$,*

$$(1-\beta)||ab||_1 \leq \mathbb{E}[d_Q(a,b)] \leq (1+\beta)||ab||_1.$$

An immediate corollary of the lemma is the following:

COROLLARY 3. *Let σ_Q^*, σ_1^* be optimal assignments for A, B under $\mathtt{d}_Q(\cdot,\cdot)$ and L_1-norms. Then,*

$$(1-\beta)\mathtt{w}(\sigma_1^*) \leq E[\mathtt{w}(\sigma_Q^*)] \leq (1+\beta)\mathtt{w}(\sigma_1^*).$$

Computing a θ-CPA. We now describe an algorithm for computing a θ-CPA of A, B under $\mathtt{d}_Q(\cdot,\cdot)$. We define a new distance function

$$\eta(a,b) = \begin{cases} \mathtt{d}_Q(a,b) & \text{if } \mathtt{d}_Q(a,b) \geq \theta, \\ \theta & \text{otherwise.} \end{cases}$$

Next, we collapse the points of A (or B) that are very close to each other into a single point. Let $j = j(\theta) = \lfloor \log_2 \theta/(16\mu) \rfloor$, i.e., j is the hightest level such that the sidelength of cells in G_j is at most $\theta/16\mu = \theta\beta^2/(2\log_2 \Delta)$. For each cell $c \in G_j$, we make two copies a_c and b_c of its center point. The demand of a_c is $d_c = \sum_{a \in A \cap C} d_a$ and the supply of b_c is $s_c = \sum_{b \in B \cap C} s_b$. Set $\overline{A} = \{a_c \mid c \in G_j\}$ and $\overline{B} = \{b_c \mid c \in G_j\}$.

LEMMA 4.3. *Let σ_η^* be the optimal assignment for $\overline{A}$, $\overline{B}$ under $\eta(\cdot,\cdot)$. If $\theta U(1/\beta - 1) \leq \mathtt{w}(\sigma_Q^*) \leq \theta U/\beta$, then*

$$\mathtt{w}(\sigma_Q^*) \leq \mathtt{w}(\sigma_\eta^*) \leq (1+\beta)\mathtt{w}(\sigma_Q^*).$$

Next, we decompose the problem of computing a θ-CPA for $\overline{A}, \overline{B}$ into smaller subproblems. Let $k = k(\theta) = \lceil \log_2(\theta\mu/(\alpha\beta)) \rceil$, i.e., k is the lowest level such that the side lengths of cells in G_k is at least $\theta\mu/(\alpha\beta)$. For each cell $\xi \in G_k$, set $\overline{A}_\xi = \overline{A} \cap \xi$ and $\overline{B}_\xi = \overline{B} \cap \xi$. $|\overline{A}_\xi| = |\overline{B}_\xi| = O(\mu^4/(\alpha^2\beta^2)) = O(\log^4 \Delta/(\beta^{10}\alpha^2))$. Let $\overline{\sigma}_\xi$ be a θ-CPA for $\overline{A}_\xi$, $\overline{B}_\xi$ and let $\overline{y}(u)$ be the dual weights for points $u \in \overline{A}_\xi \cup \overline{B}_\xi$. Set $\overline{\sigma} = \bigcup_{\xi \in G_k} \overline{\sigma}_\xi$ and let $\overline{y}$ be the dual weights of points in $\overline{A} \cup \overline{B}$.

LEMMA 4.4. *$\overline{\sigma}, \overline{y}$ is a θ-CPA for $\overline{A}, \overline{B}$ under $\eta(\cdot,\cdot)$.*

Proof. By construction, each point in $\overline{A}, \overline{B}$ satisfies (C1) and (C2), so it suffices to prove that $\overline{y}(a)+\overline{y}(b) \leq \eta(a,b)$ for all $a \in \overline{A}, b \in \overline{B}$. If a and b lie in the same cell τ of G_k then (4 holds because $\overline{\sigma}_\tau, \overline{y}$ is a θ-CPA for $\overline{A}_\tau, \overline{B}_\tau$. So assume that they lie in different cells of G_k. Then their least common ancestor is at height at least $k+1$ Therefore,

$$\mathtt{d}_Q(a,b) \geq 2^{k+1}/\mu = 2\theta\mu/(\alpha\beta) \cdot 1/\mu = 2\theta/(\alpha\beta).$$

Hence $\eta(a,b) = 2\theta/(\alpha\beta) > \theta/(\alpha\beta)$. By (C1) and (C2), we obtain $\overline{y}(a) + \overline{y}(b) \leq \theta/(\alpha\beta) < \eta(a,b)$ as desired. Hence, $\overline{\sigma}, \overline{y}$ is a θ-CPA.

Putting everything together, we obtain

THEOREM 4.1. *Let $A, B \subset [\Delta]^d$ and for $0 < \alpha, \beta < 1$, there is a randomized fully dynamic data structure that maintains an (α,β)-assignment with high probability under insertions and deletions of points. The size of the data structure is $n(\log(n\Delta)/(\alpha\beta))^{O(d)}$ and the update time is $(\log(\Delta n)/(\alpha\beta))^{O(d)}$ provided $U = n^{O(1)}$.*

References

[1] P. K. Agarwal, A. Efrat, and M. Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM J. Comput.*, 29:39–50, 1999.

[2] P. K. Agarwal and K. R. Varadarajan. A near-linear constant-factor approximation for euclidean bipartite matching? In *Proc. of 12th Annual Sympos. on Comput. Geom.*, pages 247–252, 2004.

[3] A. Andoni, K. D. Ba, P. Indyk, and D. P. Woodruff. Efficient sketches for earth-mover distance, with applications. In *Proc. 50th Annual IEEE Sympos. Foundations of Comp. Sc.*, pages 324–330, 2009.

[4] D. S. Atkinson and P. M. Vaidya. Using geometry to solve the transportation problem in the plane. *Algorithmica*, 13(5):442–461, 1995.

[5] S. Baswana, M. Gupta, and S.Sen. Fully dynamic maximal matching in o(log n) update time. In *Proc. 52nd Annual IEEE Sympos. Foundations of Comp. Sc.*, page To Appear, 2011.

[6] D. Eppstein. Dynamic euclidean minimum spanning trees and extrema of binary functions. *Discrete Comput. Geom.*, 13:111–122, 1995. 10.1007/BF02574030.

[7] H. N. Gabow and R.E. Tarjan. Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18:1013–1036, October 1989.

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

[9] P. Indyk. A near linear time constant factor approximation for euclidean bichromatic matching (cost). In *Proc. Annual Sympos. on Discrete Algorithms*, pages 39–42, 2007.

[10] K. Onak and R. Rubinfeld. Maintaining a large matching and a small vertex cover. In *Sympos. Theory of Comput.*, pages 457–464, 2010.

[11] C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity.* Prentice-Hall, Inc., 1982.

[12] J. M. Phillips and P. K. Agarwal. On bipartite matching under the rms distance. In *Canadian Conf. on Comp. Geom.*, 2006.

[13] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362 – 391, 1983.

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

[15] K. R. Varadarajan and P. K. Agarwal. Approximation algorithms for bipartite and non-bipartite matching in the plane. In *Proc. 10th Annual ACM-SIAM Sympos. on Discrete Algorithms*, pages 805–814, 1999.