# Approximation Algorithms for Bipartite Matching with Metric and Geometric Costs

Pankaj K. Agarwal
Duke University

R. Sharathkumar
Stanford University

## ABSTRACT

Let $\mathrm{G} = \mathrm{G}(A \cup B, A \times B)$, with $|A| = |B| = n$, be a weighted bipartite graph, and let $\mathtt{d}(\cdot,\cdot)$ be the cost function on the edges. Let $\mathtt{w}(M)$ denote the weight of a matching in G, and $M^*$ a minimum-cost perfect matching in G. We call a perfect matching $M$ *c-approximate*, for $c \geq 1$, if $\mathtt{w}(M) \leq c \cdot \mathtt{w}(M^*)$. We present three approximation algorithms for computing minimum-cost perfect matchings in G.

First, we consider the case when $\mathtt{d}(\cdot,\cdot)$ is a metric. For any $\delta > 0$, we present an algorithm that, in $O(n^{2+\delta} \log n \log^2(1/\delta))$ time, computes a $O(1/\delta^{\alpha})$-approximate matching of G, where $\alpha = \log_3 2 \approx 0.631$. Next, we assume the existence of a dynamic data structure for answering approximate nearest neighbor (ANN) queries under $\mathtt{d}(\cdot,\cdot)$, with a query and update time of $\tau(n,\varepsilon)$. Given two parameters $\varepsilon, \delta \in (0,1)$, we present an algorithm that, in $O(\varepsilon^{-2} n^{1+\delta} \tau(n,\varepsilon) \log^2(n/\varepsilon) \log(1/\delta))$ time, computes a $O(1/\delta^{\alpha})$-approximate matching of G, where $\alpha = 1 + \log_2(1+\varepsilon)$.

Finally, we present an algorithm that works even if $\mathtt{d}(\cdot,\cdot)$ is not a metric but still admits an ANN data structure for $\mathtt{d}(\cdot,\cdot)$ with a query and update time of $\tau(n,\varepsilon)$. In particular, we present an algorithm that, in $O(\varepsilon^{-1} n^{3/2} \tau(n,\varepsilon) \log^3(n/\varepsilon))$ time, computes a $(1+\varepsilon)$-approximate matching of $A$ and $B$.

We show that our results lead to faster matching algorithms for many geometric settings.

## Categories and Subject Descriptors

F.2.2 [**Theory of Computation**]: Nonnumerical Algorithms and Problems—*geometrical problems and computations*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms*

## General Terms

Algorithms, Theory

## Keywords

Matching, approximation algorithms

## 1. INTRODUCTION

**Problem Statement.** Let $\mathrm{G} := \mathrm{G}(A,B) = \mathrm{G}(A \cup B, A \times B)$ be a bipartite graph with $|A| = |B| = n$, and let $\mathtt{d} : A \times B \to \mathbb{R}_{\geq 0}$ be the edge-cost function. Throughout this paper we assume that $\mathtt{d}(\cdot,\cdot)$ can be evaluated in $O(1)$ time. A *perfect matching* $M$ is a set of $n$ vertex-disjoint edges in $\mathrm{G}(A,B)$. The cost of $M$, $\mathtt{w}(M)$, is the sum of cost of its edges, i.e., $\mathtt{w}(M) = \sum_{(a,b) \in M} \mathtt{d}(a,b)$. When the cost function $\mathtt{d}(\cdot,\cdot)$ is not obvious from the context, we will use the notation $\mathtt{w}(M,\mathtt{d})$ to represent the cost of $M$ under $\mathtt{d}$. The *minimum-cost bipartite matching* (or *optimal matching* for brevity) is a perfect matching with the smallest cost; let $M^*$ be such a matching. A perfect matching $M$ for G is called *c-approximate*, for $c \geq 1$, if $\mathtt{w}(M) \leq c \cdot \mathtt{w}(M^*)$. In this paper, we develop approximation algorithms for computing optimal matchings when $A$ and $B$ are points in a metric space, and when $A$ and $B$ are point sets in $\mathbb{R}^d$ and $\mathtt{d}(\cdot,\cdot)$ is not necessarily a metric. The problem of computing an optimal matching in metric and geometric settings arises in several applications such as computer vision, shape analysis, computer graphics. For example, the problem in computer vision of tracking multiple objects can be formulated as computing an optimal matching between two sets of objects, and a metric is defined to measure the distance between a pair of objects [6].

**Previous work.** Computing a maximum cardinality matching in a graph (both bipartite and general graphs) with $n$ vertices and $m$ edges takes $O(m\sqrt{n})$ time [20, 23]. The bound was improved for sparse graphs by Madry [22] (see also [17, 28]). For weighted bipartite graphs, an optimal matching with $n$ vertices and $m$ edges can be computed, in $O(mn + n^2 \log n)$ time, using the Hungarian algorithm [12]. If the edge costs are positive integers bounded by $n^{O(1)}$, Gabow and Tarjan show that an optimal matching can be computed in $O(m\sqrt{n} \log n)$ time for bipartite graphs [14], and $O(m\sqrt{n} \log^{3/2} n)$ time for general graphs [15]—almost matching the time bounds for that of the unweighted case. There are algorithms that use algebraic methods to compute optimal matching in matrix multiplication time [8, 27]. Bipartite matching is a special case of general graph matching, and the known algorithms for the latter are more complex.

If $A$ and $B$ are points in a metric space, computing an optimal bipartite matching of $A$ and $B$ seems more challenging than computing an optimal matching on a complete graph of $2n$ points in the same metric space.[1] For instance, an optimal matching in a set of $2n$ points in $\mathbb{E}^2$, the 2D Euclidean

[1]Note that bipartite matching is not a special case of the non-bipartite matching in this setting.

plane, can be computed in $O(n^{3/2} \log^5 n)$ time [33], while the best known algorithm for computing an optimal matching between two point sets of size $n$ each in $\mathbb{E}^2$ takes $O(n^{2+\delta})$ time, for any constant $\delta > 0$ [1]. Recently, Sharathkumar [29] presented a $O(n^{3/2+\delta} \log(n\Delta))$-time algorithm to compute an optimal matching between two point sets in $\mathbb{E}^2$ with positive integer coordinates bounded by $\Delta$. A sub-quadratic algorithm for computing an optimal bipartite matching of two arbitrary point sets in $\mathbb{E}^2$ has remained elusive so far.

The same trend holds for approximation algorithms. Several approximation algorithms are known for computing a (perfect) matching in a set of points (i.e., non-bipartite case) in arbitrary metric space [25, 26]. The best among those—a 2-approximation algorithm by Goemans and Williamson [18]–runs in near-quadratic time [7, 13]. However, it does not extend to the bipartite setting. The only known near-quadratic time algorithm for the bipartite case computes a matching whose expected cost is within $O(\log n)$ of the optimal matching [11].

A near-linear time algorithm for computing a $(1+\varepsilon)$-approximate matching in a set of $n$ points in $\mathbb{E}^d$, for $d = O(1)$, was first presented by Arora [4], and later improved by Varadarajan and Agarwal [34]. Following this, considerable effort was made to design near-linear-time approximation algorithm for bipartite matching in low-dimennsional Euclidean spaces [2, 21, 31, 34]. Only recently Sharathkumar and Agarwal [31] presented a near-linear time Monte-Carlo algorithm for the bipartite case. The best-known deterministic approximation algorithm known for low-dimensional Euclidean spaces is by Varadarajan and Agarwal [34] that runs in time $O(n^{3/2}(\log(n)/\varepsilon)^{O(d)})$.

In high-dimensional Euclidean space, when $d$ is not assumed to be a constant, Goel *et al.* [16] presented an algorithm for computing a $2(1+\varepsilon)$-approximate matching in a set of $2n$ points in $\mathbb{E}^d$ that runs in $O(n^{1+1/(1+\varepsilon)})$ time, for any constant $\varepsilon > 0$. For the bipartite case, however, besides the cubic-time exact algorithm, the only known approximation algorithms are an $O(dn)$ time algorithm that computes an $O(d \log n)$-approximate matching [11], and an $O(nd \log^{O(1)} n)$-time algorithm that computes an $O(\log d \log n)$-approximate matching [3].

A closely related problem is the *maximum-cost matching* problem — compute a matching with the maximum cost. While exact algorithms for maximum-cost matching and minimum-cost matching have the same running time, approximating minimum-cost matching is considerably harder than approximating maximum-cost matching. For example, a simple greedy algorithm produces a 0.5-approximation to maximum weight matching for any weighted graph. A similar greedy algorithm gives only an $O(n^{0.58})$ approximation [26] even when the edge costs are assumed to satisfy triangle inequality. In fact, as observed by Avis [5], computing an $O(n)$-approximation of minimum-cost matching on a graph with arbitrary edge costs takes the same time as computing the maximum cardinality matching of a dense unweighted graph, i.e., $\Omega(m\sqrt{n})$ time. In contrast, Duan and Pettie [9] present an $O(m/\varepsilon \log(1/\varepsilon))$-time algorithm to compute a $(1-\varepsilon)$-approximate maximum-cost matching.

**Our results.** We present three approximation algorithms for computing a minimum-cost matching in G. First, given a constant $\delta \in (0, 1/3)$, we present, in Section 3, a deterministic algorithm that computes, in $O(n^{2+\delta} \log n \log^2 1/\delta)$ time, an $O((1/\delta)^\alpha)$-approximate matching, where $\alpha = \log_3 2 \approx 0.631$, for the case when $\mathtt{d}(\cdot,\cdot)$ is any arbitrary metric that can be evaluated in $O(1)$ time. This is the first $O(1)$-approximation algorithm that runs in $o(n^{2.5})$ time. Moreover, our algorithm computes, in $O(n^2 \log^{O(1)} n)$ time, an $O((\log n / \log\log n)^\alpha)$-approximate matching; this compares favorably to the randomized quadratic-time algorithm for computing a $O(\log n)$-approximate matching. Our algorithm is a variant of the scaling algorithms [14, 30]. Roughly speaking, our algorithm computes a partial matching at each scale, commits this matching, and solves the problem recursively for unmatched points at the next scale. We use the metric property of $\mathtt{d}(\cdot,\cdot)$ to bound the cost of an optimal matching of the unmatched points, and show that we compute an approximate matching at each scale.

In Section 4, we improve the above algorithm if $\mathtt{d}(\cdot,\cdot)$ admits an efficient dynamic data structure for answering $\varepsilon$-approximate nearest-neighbor (ANN) queries. We present an algorithm that given two parameters $\varepsilon, \delta \in (0,1)$, computes an $O(1/\delta^\alpha)$-approximate matching of $A$ and $B$ in $O(\varepsilon^{-2} n^{1+\delta} \tau(n,\varepsilon) \log^2(n/\varepsilon) \log(1/\delta))$ time. Here $\alpha = 1 + \log_2(1+\varepsilon)$ and $\tau(n,\varepsilon)$ is the query and update time of the $\varepsilon/2$-ANN data structure. Note that we sacrifice the approximation factor a little but gain substantially in the running time, provided $\tau(n,\varepsilon)$ is small. If we set $\varepsilon = 1/\log_2(1/\delta)$, the algorithm computes a $O(1/\delta)$-approximate matching.

Finally, by combining the ideas of the second algorithm with the Hungarian algorithm, we present an algorithm that works even when $\mathtt{d}(\cdot,\cdot)$ is not necessarily a metric but still admits an efficient ANN data structure (e.g. squared Euclidean distance). More precisely, we present in Section 5 an algorithm that computes a $(1+\varepsilon)$-approximate matching of $A$ and $B$ in time $O(\varepsilon^{-1} n^{3/2} \tau(n,\varepsilon) \log^3(n/\varepsilon))$. Previously, data structures for answering (additive) weighted nearest neighbor (WNN) queries were used to speed up the Hungarian algorithm for geometric settings [30]. It is tempting to attain an approximation algorithm by replacing the WNN data structure in these algorithms with the one that answers approximate WNN queries, with the hope that the latter can be answered more quickly. However, it is unlikely that approximate WNN queries can be answered quickly—in time similar to unweighted ANN queries, because exact NN queries can be reduced to answering approximate WNN queries. We overcome this difficulty by using a different approach that allows us to use a weaker notion of approximate WNN queries which can be answered using ANN data structure.

These results lead to faster matching algorithms for many geometric settings. We just mention two of them here:

**(i)** For $A, B \subseteq \mathbb{E}^d$, with $d = \Omega(\log n)$, and for a constant $\delta > 0$, an $O(1/\delta^{\log_2(1/\delta)})$-approximate matching can be computed in $O(n^{1+\delta} \log n)$ time. This is the first sub-quadratic algorithm for computing an $O(1)$-approximate matching of high-dimensional point sets.

**(ii)** For $A, B \subseteq \mathbb{R}^d$, with $d = O(1)$, and for a constant $\delta > 0$, we obtain a *deterministic* algorithm that computes $O(\frac{1}{\delta})$-approximate matching of $A$ and $B$ in $O(n^{1+\delta} \log n)$ time. In contrast, the previously best known deterministic algorithm computes an $\frac{1}{\delta}$-approximate matching in $O(n^{3/2} \log^{O(d)} n)$ time [34] (using our second algorithm). Alternatively, we also obtain a *deterministic* algorithm that computes an $(1+\varepsilon)$-approximate matching of $A$ and $B$ in $O(\frac{1}{\varepsilon^{O(d)}} n^{3/2} \log^5(n/\varepsilon))$

(using our third algorithm), while the algorithm in [34] takes $O(n^{3/2}(\log(n)/\varepsilon)^{O(d)})$ time.

## 2. PRELIMINARIES

In this section, we present a few concepts that will be useful for our algorithms. Let $A, B$, G, and $\mathtt{d}(\cdot,\cdot)$ be as defined earlier. Given a matching $M$ on 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 of $A \cup B$ that has not been matched, i.e., not incident on any edge of $M$. An *augmenting path* $P$ is an alternating path between two free vertices. $M$ can be *augmented* by one edge along $P$, by setting $M = M \oplus P$, where $\oplus$ is the symmetric difference, i.e., remove $P \cap M$ from $M$ and add $P \setminus M$ to $M$. We refer to this step as an *augmentation* step.

An *alternating tree* is a tree rooted at a free vertex of $B$ in which every path to the leaf node is an alternating path. While describing our algorithms, sometimes it is useful to view G as a directed graph $\vec{G}_M$, defined as follows: Every edge $(a, b) \in M$ with $a \in A$ and $b \in B$, is directed from $a$ to $b$, and every edge $(a, b) \notin M$ is directed from $b$ to $a$. The in-degree (resp. out-degree) of every vertex of $B$ (resp. $A$) is at most 1. Any alternating path is a directed path in $\vec{G}_M$. An alternating tree $\mathcal{T}$ rooted at a free vertex $b \in B$ is a directed tree in $\vec{G}_M$, in the sense that $\mathcal{T}$ has a directed path from the root $b$ to every vertex in $\mathcal{T}$.

Recall that Hungarian and many other algorithms assign dual weights to each vertex of G. Let $d(\cdot,\cdot)$ be a distance function. For a vertex $v \in A \cup B$, let $y(v)$ be its *dual weight.* A matching $M$ and a set of dual weights are *dual feasible* if

$$\begin{aligned} y(a) + y(b) &\le d(a, b) && \forall (a, b) \in A \times B, \\ y(a) + y(b) &= d(a, b) && \forall (a, b) \in M. \qquad (1) \end{aligned}$$

It is well-known that any dual-feasible perfect matching is optimal with respect to $d(\cdot,\cdot)$; see [24]. Introduced by Gabow and Tarjan [14], a 1-*feasible matching* is a matching $M$ and set of dual weights $y(\cdot)$ such that

$$\begin{aligned} y(a) + y(b) &\le d(a, b) + 1 && \forall (a, b) \in A \times B, \\ y(a) + y(b) &= d(a, b) && \forall (a, b) \in M. \qquad (2) \end{aligned}$$

A 1-*optimal matching* is a perfect matching that is 1-feasible. The notion of 1-feasible matching will be used in Section 3. An edge $(a, b) \in A \times B$ is called *eligible* if

$$\begin{aligned} y(a) + y(b) &= d(a, b) && (a, b) \in M, \\ y(a) + y(b) &= d(a, b) + 1 && (a, b) \notin M. \qquad (3) \end{aligned}$$

The subgraph of G induced by the set of eligible edges is called the *eligible graph* (with respect to $M$).

Relaxing the notion of feasible edges differently, a matching $M$ is called $\varepsilon$-*relaxed* if the dual weights satisfy the following condition:

$$\begin{aligned} y(a) + y(b) &\le \mathtt{d}(a, b) + \lceil \varepsilon \mathtt{d}(a, b) \rceil, && \forall (a, b) \in A \times B, \qquad (4) \\ y(a) + y(b) &= \mathtt{d}(a, b) && \forall (a, b) \in M. \qquad (5) \end{aligned}$$

The following lemma shows the significance of $\varepsilon$-relaxed matchings.

LEMMA 2.1. *Let $M$ be a perfect $(\varepsilon/2)$-relaxed matching of $A$ and $B$, and let $M$ be an optimal matching of them. If $\mathtt{w}(M^*) \ge 2n/\varepsilon$, then*

$$\mathtt{w}(M) \le (1 + \varepsilon)\mathtt{w}(M^*).$$

PROOF. Since $M$ is a perfect matching, from (5),we have

$$\mathtt{w}(M) = \sum_{(u,v)\in M} \mathtt{d}(u, v) = \sum_{(u,v)\in M} y(u) + y(v) = \sum_{v\in A\cup B} y(v).$$

Every edge $(a, b) \in M^*$ satisfies (4) and $\mathtt{w}(M^*, \mathtt{d}) \ge \frac{2n}{\varepsilon}$, therefore

$$\begin{aligned} \sum_{a\in A\cup B} y(a) &\le \sum_{(u,v)\in M^*} \mathtt{d}(u, v) + \left\lceil \frac{\varepsilon}{2}\mathtt{d}(u, v) \right\rceil \\ &\le (1 + \frac{\varepsilon}{2})\mathtt{w}(M^*) + n \le (1 + \varepsilon)\mathtt{w}(M^*, \mathtt{d}). \end{aligned}$$

□

## 3. ALGORITHM FOR ARBITRARY METRIC SPACE

Let $A, B$ be two point sets of size $n$ each in a metric space, and let $\mathtt{d}(\cdot,\cdot)$ be the underlying metric such that the distance between an two points of $A \cup B$ can be computed in $O(1)$ time. We describe an approximation algorithm for computing an optimal matching of $A$ and $B$ under $\mathtt{d}(\cdot,\cdot)$. Our algorithm is a variant of the *scaling algorithms* for matching [14, 30]. Before presenting the algorithm, we breifly describe the main idea behind it. This description, although somewhat incorrect, conveys the main intuition.

**Main idea.** Suppose we fix a scale of the Gabow-Tarjan algorithm [14]. At a fixed scale, their algorithm runs in stages, each of which computes, in $O(n^2)$ time, a maximal set of vertex-disjoint augmenting paths, augments the matching along these paths, and updates the dual weights of the vertices. If we run their algorithm for $n^\alpha$ stages, then it computes a matching $M_1$ that matches, in $O(n^{2+\alpha})$ time, all but $O(n^{1-\alpha})$ points of $A$ and $B$, and $\mathtt{w}(M_1) \le 2\mathtt{w}(M^*)$, where $M^*$ is the optimal matching of $A, B$. Next, we compute an optimal matching $M_2$ of the free points of $A$ and $B$, using the Hungarian algorithm, which takes $O(n^{3(1-\alpha)})$ time, and return $M_1 \cup M_2$ as a matching of $A, B$. Using the fact that $\mathtt{d}(\cdot,\cdot)$ is a metric, we can argue that $\mathtt{w}(M_2) \le \mathtt{w}(M_1) + \mathtt{w}(M^*) \le 3\mathtt{w}(M^*)$. By setting $\alpha = 1/4$, we obtain a 5-approximate matching of $A, B$ in $O(n^{9/4})$ time. We can repeat this process once more, but this time use the $O(n^{9/4})$ algorithm, instead of Hungarian algorithm, to compute $M_2$. We obtain a faster algorithm that returns an $O(1)$-approximate matching, though the constant is now larger than 5. By iterating this process multiple times, we can obtain a trade-off between the approximation factor and the running time. How we do this correctly and obtain a good trade-off requires some care, as described below.

There are three main differences between our and traditional scaling algorithms [14, 30]: (i) We commit the edges computed in the matching computed in one scale, and the next scale computes a matching of only unmatched points. (ii) We compute a partial matching at each scale. By choosing the number of steps for which we run the algorithm at each scale, we keep both the running time and the cost of the matching under control. (iii) In successive scales, we scale down the edge costs instead of scaling up, so that the cost of the optimal matching of unmatched points is linear in the number of the points.

**Algorithm.** We now describe the algorithm in detail. We choose a parameter $\delta \in (0, 1/3]$ and set $\varepsilon = \frac{1}{2\log_3 1/\delta}$. Let $\omega > 0$ be a real value such that $\mathtt{w}(M^*)/2 \le \omega \le \mathtt{w}(M^*)$. For now, we assume that we have such a value of $\omega$ at our disposal. At the end, we describe how to choose $\omega$. The algorithm works in phases. In the beginning of phase $i$, for $i \geq 0$, we have sets $A_i \subseteq A$ and $B_i \subseteq B$ of points and a matching $\mathbb{M}_i$ of the points in $A \setminus A_i$ and $B \setminus B_i$; set $|A_i| = |B_i| = n_i$. For $i = 0$, $A_0 = A$, $B_0 = B$, $n_0 = n$, and $\mathbb{M}_0 = \emptyset$. If $n_i \leq n^{2/3}$, we compute an optimal matching of $A_i$ and $B_i$ using the Hungarian algorithm, which takes $O(n_i^3) = O(n^2)$ time. So, assume $n_i \geq n^{2/3}$. In phase $i$, we set the distance function to be $\mathtt{d}_i(\cdot,\cdot)$, defined as follows:

$$\mathtt{d}_i(a,b) = \begin{cases} \left\lceil \dfrac{2\mathtt{d}(a,b)\cdot n}{\varepsilon\omega} \right\rceil & \text{if } i = 0, \\ \left\lceil \dfrac{\mathtt{d}_{i-1}(a,b)}{2(1+\varepsilon)^2 n^{\varphi_{i-1}}} \right\rceil & \text{if } i > 0, \end{cases} \tag{6}$$

where $\varphi_i = 3^i\delta$.

We compute a (possibly partial) matching $M_i$ of $A_i$ and $B_i$ under $\mathtt{d}_i(\cdot,\cdot)$ using the algorithm Phase-Match described below. We set $\mathbb{M}_{i+1} \leftarrow \mathbb{M}_i \cup M_i$. If $M_i$ is a perfect matching between $A_i$ and $B_i$, we stop and return $\mathbb{M}_{i+1}$. Otherwise, we set $A_{i+1}$ (resp. $B_{i+1}$) to be the set of free vertices of $A_i$ (resp. $B_i$) and move to phase $i+1$.

We now describe the Phase-Match algorithm, which is similar to Gabow and Tarjan's [14] algorithm but with some additional twists. We set $\kappa_i = \frac{30}{\varepsilon} n^{\varphi_i}$. The procedure works in stages. In the beginning of each stage, we have a set $A_i^F \subseteq A_i$ and $B_i^F \subseteq B_i$ of free vertices, dual weights $y(v)$ for each point in $A_i \cup B_i$ and a 1-feasible matching $M$ computed so far. Initially, $A_i^F = A_i$, $B_i^F = B_i$, and $M = \emptyset$, and $y(v) = 0$ for all $v \in A_i \cup B_i$. At the end of the stage, if we have computed a perfect matching of $A_i, B_i$ or run the procedure for $\kappa_i$ stages, we stop and return $M$ as $M_i$.

Each stage of Phase-Match consists of the following three steps:
***(i) Finding augmenting paths.*** Let $\mathcal{E} \subseteq A_i \times B_i$ be the set of *eligible* edges. We compute, in $O(n_i^2)$ time, a maximal set $\mathbb{P}$ of vertex-disjoint augmenting paths in $\mathcal{E}$, as described in [14, 30].
***(ii) Augmentation step.*** We augment $M$ along $\mathbb{P}$, i.e., set $M = M \oplus (\bigcup_{P\in\mathbb{P}} P)$. For each vertex $b \in B_i$ that lies on a path in $\mathbb{P}$, set $y(b) \leftarrow y(b) - 1$, to ensure that $M$ continues to be 1-feasible after the augmentation step. This step takes $O(n_i)$ time because the paths in $\mathbb{P}$ are vertex disjoint.
***(iii) Dual-adjustment step.*** Consider the directed subgraph $G_i$ formed by $\mathcal{E}$, the set of eligible edges, as defined in Section 2. Let $S$ (resp. $T$) be the set of vertices of $B_i$ (resp. $A_i$) that are reachable from a vertex of $B_i^F$ in $G_i$. Set $y(b) = y(b) + 1$ for all $b \in S$ and $y(a) = y(a) - 1$ for all $a \in T$. This step takes $O(n_i^2)$ time.

This completes the description of the Phase-Match procedure. Since each stage takes $O(n_i^2)$ time, the total time taken by Phase-Match is $O(n_i^2\kappa_i)$.

**Analysis.** Before analyzing the performance of the algorithm, we make a few observations about the Phase-Match procedure, which are well-known; see [14, 30].

Lemma 3.1. *(i) The matching $M$ is always 1-feasible. (ii) At the end of the augmentation step in each stage, there is no augmenting path composed solely of eligible edges. (iii) For any vertex $v \in A_i \cup B_i$ that is free after $k$ stages, $y(v) = 0$ if $v \in A_i$ and $y(v) = k$ if $v \in B_i$.*

We claim that our (overall) algorithm maintains the following invariants after $i$ phases. Let $M_i^*$ be the optimal perfect matching of $A_i$ and $B_i$ under $\mathtt{d}(\cdot,\cdot)$; note that $M_0^* = M^*$.

(I1) $\mathtt{w}(M_i) \leq (1+\varepsilon)\mathtt{w}(M_i^*)$,

(I2) $\mathtt{w}(M_{i+1}^*) \leq (2+\varepsilon)\mathtt{w}(M_i^*)$, and

(I3) $n_i \leq n^{1-\Phi_i}$, where $\Phi_i = \sum_{j=0}^{i-1}\varphi_i = \frac{3^i-1}{2}\delta$.

We note that (I2) holds because $\mathtt{d}(\cdot,\cdot)$ is a metric, and this is the only place where the metric property is used. Assuming that these invariants hold, we analyze the performance of the algorithm.

(I3) implies that the algorithm runs as long as $\Phi_i < 1/3$. In other words, the algorithm stops after $\mu \leq \lceil \log_3(\frac{2}{3\delta} + 1) \rceil$ phases.

Next, we analyze the running time of the algorithm. In phase $i$, Phase-Match takes $O(n_i^2\kappa_i) = O(\frac{n_i^{\varphi}}{\varepsilon} n^{2-2\Phi_i})$ time. Since

$$\varphi_i - 2\Phi_i = 3^i\delta - 2\frac{3^i-1}{2}\delta = \delta,$$

the Phase-Match procedure in phase $i$ runs in $O(n^{2+\delta}/\varepsilon)$ time. Plugging $\mu = O(\log(1/\delta))$ and $\varepsilon = \frac{1}{2\log_3(1/\delta)}$, the overall running time of the algorithm is $O(n^{2+\delta}\log^2(1/\delta))$.

The following lemma, which bounds the cost of the matching computed by the algorithm, follows from (I1), (I2), and the value of $\mu$.

Lemma 3.2. *Let $\mathbb{M}$ be the final matching computed by the algorithm, then $\mathtt{w}(\mathbb{M}) = O(1/\delta^\alpha)\cdot\mathtt{w}(M^*)$, where $\alpha = \log_3 2 \approx 0.631$.*

Proof. Using the invariant (I1),

$$\mathtt{w}(\mathbb{M}) = \sum_{i=0}^{\mu}\mathtt{w}(M_i) \leq (1+\varepsilon)\sum_{i=0}^{\mu}\mathtt{w}(M_i^*).$$

By applying (I2) repeatedly, we obtain

$$\mathtt{w}(M_i^*) \leq 2^i(1+\varepsilon)^i\mathtt{w}(M_0^*) = 2^i(1+\varepsilon)^i\mathtt{w}(M^*).$$

Hence,

$$\mathtt{w}(\mathbb{M}) \leq (1+\varepsilon)\mathtt{w}(M^*)\sum_{i=0}^{\mu-1}2^i(1+\varepsilon)^i \leq (1+\varepsilon)(2(1+\varepsilon))^\mu\mathtt{w}(M^*).$$

Since $\mu \leq \lceil \log_3(\frac{2}{3\delta}+1) \rceil$ and $\varepsilon = \frac{1}{2\log_3(1/\delta)}$, $(1+\varepsilon)^\mu = O(1)$ and $2^\mu = O(1/\delta^\alpha)$, where $\alpha = \log_3 2$. Hence, $\mathtt{w}(\mathbb{M}) = O(1/\delta^\alpha)\mathtt{w}(M^*)$. $\square$

We now prove invariants (I1)–(I3). We first note that $\mathtt{d}_i(\cdot,\cdot)$ is not an exact scaled version of $\mathtt{d}(\cdot,\cdot)$ because of the ceiling operator in (6). Nevertheless, we can prove the following properties for $\mathtt{d}(\cdot,\cdot)$.

Lemma 3.3. *(i) For any $i \geq 0$, $\mathtt{d}_i(\cdot,\cdot)$ is a metric.*

*(ii) For $i \geq 1$, $\mathtt{d}_i(a,b) \geq \frac{6}{\varepsilon}$ for any $(a,b) \in A_i \times B_i$.*

*(iii) For any $i \geq 1$, there is a scaling factor $\sigma_i$ such that*

$$(1-\varepsilon/3)\sigma_i\mathtt{d}_i(a,b) \leq \mathtt{d}(a,b) \leq \sigma_i\mathtt{d}_i(a,b).$$

PROOF. (i) Suppose $d(\cdot,\cdot)$ is a distance function that satisfies triangular inequality, i.e., for any three points $a, b$ and $c$, $d(a,b)+d(b,c) \geq d(a,c)$. This inequality continues to hold if we multiply it by a parameter $k \in \mathbb{R}$ and take the ceiling, so we get

$$\lceil kd(a,b)\rceil + \lceil kd(b,c)\rceil \quad \geq \quad \lceil kd(a,c)\rceil. \tag{7}$$

Part (i) can be proved by induction on $i$. For $i=0$, set $k=\frac{n}{\varepsilon\omega}$ and $d(\cdot,\cdot)$ to $\mathtt{d}(\cdot,\cdot)$, and we have $\mathtt{d}_0(\cdot,\cdot)$ satisfying triangle inequality. Assume $\mathtt{d}_{i-1}(\cdot,\cdot)$ satisfies triangle inequality. Setting $k=\frac{1}{2(1+\varepsilon)^2 n^{\varphi_{i-1}}}$ and $d(\cdot,\cdot)=\mathtt{d}_{i-1}(\cdot,\cdot)$, we have $\mathtt{d}_i(\cdot,\cdot)$ satisfying triangle inequality.

(ii) If the algorithm does not terminate in phase $i$, then phase $i$ runs for $\frac{30}{\varepsilon}n^{\varphi_i}$ iterations, so for any $(a,b)\in A_i^F\times B_i^F$, Lemma 3.1(iii) implies that

$$y_i(a)+y_i(b)=\tfrac{30}{\varepsilon}n^{\varphi_i}\leq \mathtt{d}_i(a,b)+1$$

or $\mathtt{d}_i(a,b)\geq \frac{30}{\varepsilon}n^{\varphi_i}-1$. But

$$\mathtt{d}_{i+1}(a,b)\geq \frac{\mathtt{d}_i(a,b)}{2(1+\varepsilon)^2 n^{\varphi_i}}\geq \frac{30}{2(1+\varepsilon)^2\varepsilon}-1\geq \frac{6}{\varepsilon}.$$

The last inequality follows since $\varepsilon=\frac{1}{2\log_3(1/\delta)}\leq 1/2$.

(iii) By ignoring the ceiling operator in (6), we only decrease the value of $\mathtt{d}_i(\cdot,\cdot)$, so using (6) repeatedly, we obtain

$$1/\sigma_i\cdot \mathtt{d}(a,b)\leq \mathtt{d}_i(a,b) \tag{8}$$

where, $\sigma_i=\frac{\omega\varepsilon(2(1+\varepsilon)^2)^i}{n^{1-\Phi_i}}$. On the other hand,

$$\mathtt{d}_i(a,b)\leq \frac{1}{2(1+\varepsilon)^2 n^{\varphi_{i-1}}}\mathtt{d}_{i-1}(a,b)+1 \tag{9}$$

Unraveling the recurrence and following a straight-forward, albeit somewhat tedious, calculation, we obtain

$$\mathtt{d}_i(a,b)\leq \frac{\mathtt{d}(a,b)}{\sigma_i}+2 \tag{10}$$

Now using part (ii) of the lemma, we obtain

$$(1-\varepsilon/3)\sigma_i\mathtt{d}_i(a,b)\leq \mathtt{d}(a,b)\leq \sigma_i\mathtt{d}_i(a,b). \tag{11}$$

□

COROLLARY 3.4. *For $i\geq 1$, Let $M$ and $M'$ be two (possibly partial) matchings of $A_i, B_i$.*

*(i) If $M$ is a perfect matching, then $\mathtt{w}(M,\mathtt{d}_i)\geq \frac{6}{\varepsilon}n_i$.*

*(ii)* $$(1-\varepsilon/3)\frac{\mathtt{w}(M,\mathtt{d}_i)}{\mathtt{w}(M',\mathtt{d}_i)}\leq \frac{\mathtt{w}(M,\mathtt{d})}{\mathtt{w}(M',\mathtt{d})}\leq \frac{1}{1-\varepsilon/3}\frac{\mathtt{w}(M,\mathtt{d}_i)}{\mathtt{w}(M',\mathtt{d}_i)}.$$

Now, we are ready to prove the invariants (I1)–(I3).

**Proof of (I1):** For every edge $(a,b)\in M_i$, $y(a)+y(b)=\mathtt{d}_i(a,b)$ and for every free vertex $v$ of $A_i\cup B_i$, $y(v)\geq 0$, therefore,

$$\mathtt{w}(M_i,\mathtt{d}_i)\leq \sum_{v\in A_i\cup B_i} y(v).$$

For every edge $(a,b)\in M_i^*$, $y(a)+y(b)\leq \mathtt{d}_i(a,b)+1$. Every vertex of $A_i\cup B_i$ is incident on exactly one edge of $M_i^*$, so

$$\sum_{v\in A_i\cup B_i} y(v)\leq \sum_{(a,b)\in M_i^*}\mathtt{d}_i(a,b)+n_i\leq \mathtt{w}(M_i^*,\mathtt{d}_i)+n_i. \tag{12}$$

Consequently,

$$\mathtt{w}(M_i,\mathtt{d}_i)\leq \mathtt{w}(M_i^*,\mathtt{d}_i)+n_i. \tag{13}$$

We prove (I1) for cases $i=0$ and $i>0$ separately:

**(i) Case $i=0$:** From (13), we have $\mathtt{w}(M_0,\mathtt{d}_0)\leq \mathtt{w}(M^*,\mathtt{d}_0)+n$. Since $\mathtt{d}_0(a,b)=\left\lceil\frac{2\mathtt{d}(a,b)\cdot n}{\varepsilon\omega}\right\rceil$ and $\omega\leq \mathtt{w}(M^*)$, we obtain the following inequalities:

$$\begin{aligned}\mathtt{w}(M_0,\mathtt{d}_0) &\leq \sum_{(a,b)\in M^*}\left(\frac{2n}{\varepsilon\omega}\mathtt{d}(a,b)+1\right)+n\\ &\leq \frac{2n}{\varepsilon\omega}\mathtt{w}(M^*)+2n,. \qquad (14)\\ \mathtt{w}(M_0,\mathtt{d}_0) &\geq \frac{2n}{\varepsilon\omega}\sum_{(a,b)\in M_0}\mathtt{d}(a,b)=\frac{2n}{\varepsilon\omega}\mathtt{w}(M_0). \qquad (15)\end{aligned}$$

Combining (14) and (15) and using $\omega\leq \mathtt{w}(M^*)$,

$$\mathtt{w}(M_0)\leq \mathtt{w}(M^*)+\varepsilon\omega\leq (1+\varepsilon)\mathtt{w}(M^*).$$

**(ii) Case $i>0$:** From (13) and Corollary 3.4 (i), we get

$$\mathtt{w}(M,\mathtt{d}_i)\leq \mathtt{w}(M_i^*,\mathtt{d}_i)+n_i\leq (1+\varepsilon/6)\mathtt{w}(M^*,\mathtt{d}_i).$$

Finally, by Corollary 3.4(ii),

$$\mathtt{w}(M_i)\leq \frac{1+\varepsilon/6}{1-\varepsilon/3}\mathtt{w}(M_i^*)\leq (1+\varepsilon)\mathtt{w}(M_i^*,\mathtt{d}).$$

**Proof of (I2):** $M_i\oplus M_i^*$ is a set of alternating cycles and a set $\mathbb{P}$ of vertex-disjoint augmenting paths. Each path in $\mathbb{P}$ connects a free vertex of $B_i$ to a free vertex of $A_i$, i.e., $|\mathbb{P}|=n_{i+1}$. Using (I1),

$$\sum_{P\in\mathbb{P}}\sum_{(a,b)\in P}\mathtt{d}(a,b)=\mathtt{w}(M_i)+\mathtt{w}(M_i^*)\leq (2+\varepsilon)\mathtt{w}(M_i^*).$$

Recall that $\mathtt{d}(\cdot,\cdot)$ satisfies the triangle inequality, so if the endpoints of a path $P_j\in\mathbb{P}$ are $(a_j,b_j)\in A_i\times B_i$, then $\mathtt{d}(a_j,b_j)\leq \sum_{(a,b)\in P_j}\mathtt{d}(a,b)$. Hence,

$$\mathtt{w}(M_{i+1}^*,\mathtt{d})\leq \sum_{P\in\mathbb{P}}\sum_{(a,b)\in P}\mathtt{d}(a,b)\leq (2+\varepsilon)\mathtt{w}(M_i^*,\mathtt{d}).$$

**Proof of (I3):** We claim that

$$\mathtt{w}(M_i^*,\mathtt{d}_i)\leq \frac{5}{\varepsilon}n^{1-\Phi_i}. \tag{16}$$

Suppose this claim is true. Then $n_i\leq n^{1-\Phi_i}$ because $\mathtt{w}(M_i^*,\mathtt{d}_i)\geq \frac{5}{\varepsilon}n_i$, by Corollary 3.4 (i).

It thus suffices to prove the above claim, which we prove by induction on $i$. For $i=0$, $\mathtt{d}_0(a,b)-1\leq \frac{2n}{\varepsilon\omega}\mathtt{d}(a,b)$. Since $\omega\geq \mathtt{w}(M^*)/2$ we have,

$$\mathtt{w}(M_0^*,\mathtt{d}_0)-n\leq \frac{2n}{\varepsilon\omega}\mathtt{w}(M_0^*)\leq \frac{2n}{\varepsilon\omega}\mathtt{w}(M^*,\mathtt{d})\leq \frac{4n}{\varepsilon}.$$

Therefore, $\mathtt{w}(M_0^*,\mathtt{d}_0)\leq \frac{5}{\varepsilon}n$.

By induction hypothesis, assume that

$$\mathtt{w}(M_{i-1}^*,\mathtt{d}_{i-1})\leq \frac{5}{\varepsilon}n^{1-\Phi_{i-1}}.$$

Using (I1), (I2), (6) and Corollary 3.4(ii), we can write

$$\mathtt{w}(M_i^*, \mathtt{d}_{i-1}) \le \frac{2+\varepsilon}{1-\varepsilon/3}\mathtt{w}(M_{i-1}^*, \mathtt{d}_{i-1}) \le 2(1+\varepsilon)\frac{5}{\varepsilon}n^{1-\Phi_{i-1}}; \tag{17}$$

the last inequality follows because $\varepsilon \le 1/2$. On the other hand, by Lemma 3.3(ii),

$$(1-\varepsilon/6)\mathtt{d}_i(a,b) \le \mathtt{d}_i(a,b) - 1 \le \frac{\mathtt{d}_{i-1}(a,b)}{2(1+\varepsilon)^2 n^{\varphi_{i-1}}}.$$

Therefore

$$\mathtt{d}_i(a,b) \le \frac{\mathtt{d}_{i-1}(a,b)}{2(1+\varepsilon)n^{\varphi_{i-1}}}. \tag{18}$$

Combining (17) and (18),

$$\mathtt{w}(M_i^*, \mathtt{d}_i) \le \frac{5}{\varepsilon}n^{1-\Phi_{i-1}-\varphi_{i-1}} = \frac{5}{\varepsilon}n^{1-\Phi_i},$$

as desired. Finally, to guess the value of $\omega$, we compute an $n$-approximate matching of $A$ and $B$ in $O(n^2)$ time [30]. Let $\Theta$ be the cost of the matching, then we run the above algorithm with $\omega = \Theta/2^i$, for all $1 \le i \le \lceil \log_2 n \rceil$, and choose the minimum-cost matching among them. Since one of these values of $\omega$ lies in the range $[\mathtt{w}(M^*)/2, \mathtt{w}(M^*)]$, we obtain the following.

THEOREM 3.5. *Let $A, B$ be two point sets of size $n$ each in a metric space such that the distance between any two points of $A$ and $B$ can be computed in $O(1)$ time, and let $\delta \in (0, 1/3)$ be a constant. Then an $O(1/\delta^\alpha)$-approximate matching of $A, B$ can be computed in $O(n^{2+\delta}\log n \log^2(1/\delta))$ time, where $\alpha = \log_3 2 \approx 0.631$.*

## 4. ANN DATA STRUCTURE BASED ALGORITHM

In this section, we present a more efficient algorithm by adapting the algorithm described in Section 3 and using a dynamic data structure for answering approximate nearest-neighbor (ANN) queries. Given two parameters, $0 \le \delta \le 1/3$ and $0 < \varepsilon < 1$, it computes a $O(1/\delta^\alpha)$-approximate matching, for $\alpha = 1 + \log_2(1+\varepsilon)$, in $O(n^{1+\delta}\tau(n,\varepsilon)\log^2(n/\varepsilon))$, where $\tau(n,\varepsilon)$ is the update and query time of an $\varepsilon/2$-approximate nearest neighbor ($\varepsilon/2$-ANN) data structure.

We will now use the notion of $\varepsilon$-relaxed matching described in Section 2, instead of 1-feasible matching. Given an $\varepsilon$-relaxed matching $M$, we call an edge $(a,b) \in A \times B$ *strongly eligible* if $(a,b) \in M$ (in which case $y(a) + y(b) = \mathtt{d}(a,b)$) or $(a,b) \notin M$ and $y(a) + y(b) = \mathtt{d}(a,b) + \lceil \varepsilon \mathtt{d}(a,b) \rceil$. We call $(a,b) \notin M$ *weakly eligible* if $\mathtt{d}(a,b) < y(a) + y(b) < \mathtt{d}(a,b) + \lceil \varepsilon \mathtt{d}(a,b) \rceil$. Given $M$, let $\Sigma$ (resp. $\Omega$) be the set of all strongly (resp. weakly) eligible edges of $A \times B$.

We call a graph $G = (A \cup B, \mathcal{E})$ an *$\varepsilon$-eligible graph* if $\Sigma \subseteq \mathcal{E}$ and $\mathcal{E} \subseteq \Sigma \cup \Omega$, i.e., $\mathcal{E}$ contains all strongly eligible edges and possibly some of the weakly eligible edges; an edge that is not weakly or strongly eligible is not in $\mathcal{E}$. Note that $G$ is not unique as we do not specify which of the weakly eligible edges should be in $G$. We also note that we do not construct $G$ explictly. Instead, we will be relying on a data structure to explore an $\varepsilon$-eligible graph.

**Algorithm.** The structure of the overall algorithm is the same as in Section 3, we just change some of the parameters. In particular, in the beginning of phase $i$, for $i \ge 0$, we have sets $A_i$ and $B_i$ of $n_i$ points each and a matching $\mathbb{M}_i$ of $A \setminus A_i$ and $B \setminus B_i$. We set the threshold of $n_i$ to $n^{1/3}$ instead of $n^{2/3}$, i.e., if $n_i < n^{1/3}$ we execute the Hungarian algorithm to compute an optimal matching of $A_i, B_i$, which takes $O(n_i^3) = O(n)$ time. So assume $n_i \ge n^{1/3}$. We set the distance function in phase $i$ to be $\mathtt{d}_i(\cdot,\cdot)$ as defined in (6) except that we set $\varphi_i = 2^i\delta$ (instead of $3^i\delta$)

Suppose we have a data structure, which we describe later, that given an $\varepsilon$-relaxed matching $M$ and weights $y(v)$ for all $v \in A_i \cup B_i$, maintains a set $K \subset A_i$ and supports the following two operations:

- DELETE $(a)$: Deletes $a$ from $K$.
- ELIGIBLE $(b)$: Given a point $b \in B_i$, if there is a strongly eligible edge incident on $b$ whose other end-point is in $K$, it returns an edge $(a,b)$ with $a \in K$ and $(a,b) \in \Sigma \cup \Omega$, i.e., $(a,b)$ is strongly or weakly eligible edge. If no strongly eligible edge is incident upon $b$, the procedure either returns NULL or an edge $(a,b) \in \Omega$ with $a$ in $K$.

Let $\tilde{\tau}(n_i,\varepsilon)$ be the time taken by each of these two operations. The data structure can be constructed in $O(n_i\tilde{\tau}(n_i,\varepsilon))$ time.

The PHASE-MATCH procedure also has the same structure as in Section 3, except that we use the above data structure to compute augmenting paths. PHASE-MATCH works in stages and computes a $(\varepsilon/2)$-relaxed matching $M$ of $A_i$ and $B_i$. We set $\kappa_i = \frac{30}{\varepsilon}n^{\varphi_i}$, where $\varphi_i = 2^i\delta$ In the beginning of each stage, we have sets $A_i^F \subseteq A_i$, $B_I^F \subseteq B_i$ of free vertices, and we build the above data structure on $A_i$. Each stage again consists of three steps. We describe the modifications to the other two steps.

***(i) Finding augmenting paths.*** As before, we construct a maximal set of vertex-disjoint augmenting paths $\mathbb{P}$ of edges that are strongly or weakly eligible – the paths are maximal in the sense that no additional augmenting path composed of only strongly eligible edges exists. We compute $\mathbb{P}$ by doing a depth first search from the free vertices of $B_i$. Suppose we are currently at a vertex $b \in B_i$, then we call ELIGIBLE$(b)$ to find a vertex of $a \in A_i$ that can be reached from $b$ using a strongly or weakly eligible edge. If we visit a vertex $a \in A_i$, we delete $a$ from the data structure. We omit the details from here.

The total time spent by the procedure is $O(n\tilde{\tau}(n_i,\varepsilon))$. Let $T \subset A_i$ and $S \subset B_i$ be all the points that are visited by this step.

***(ii) Augmentation step.*** We augment along every path of $\mathbb{P}$. The only difference is that, for every path $P \in \mathbb{P}$, we decrease the dual weights of points in $B \cap P$ so that every new edge $(a,b)$ added to the matching satisfies $y(a)+y(b) = \mathtt{d}(a,b)$. Since any edge $(a,b) \in P$ that is not in the matching is either weakly or strongly eligible, it satisfies $y(a) + y(b) > \mathtt{d}(a,b)$. Therefore, the reduction of the dual weight is at least 1. Total time taken by this step is $O(n_i)$.

***(iii) Dual-adjustment step.*** Increase the dual weight of every point in $S$ by 1, and reduce the dual weight of every point in $T$ by 1.

This completes the description of PHASE-MATCH. Each stage of the algorithm runs in $O(n_i\tilde{\tau}(n_i,\varepsilon))$ time, so the total time for each phase is $O(n_i\kappa_i\tilde{\tau}(n_i,\varepsilon))$.

**Analysis.** The following variant of Lemma 3.1 now holds:

LEMMA 4.1. *(i) The matching $M$ is always $(\varepsilon/2)$-relaxed. (ii) At the end of the augmentation step in each stage, there* *is no augmenting paths composed of only strongly eligible edges. (iii) For any vertex $v \in A_i \cup B_i$ that is free after $k$ stages, $y(v) = 0$ if $v \in A_i$ and $y(v) = k$ if $v \in B_i$.*

We now analyze the performance of the algorithm. The invariants (I1)–(I3) of Section 3 still hold but $\Phi_i = \sum_{j=0}^{i} \varphi_i = (2^i - 1)\delta$. Following the same analysis as in Section 3, we obtain that the algorithm stops within $\mu = \lceil \log_2(2/3\delta + 2) \rceil = O(\log 1/\delta)$ phases, and the phase $i$ takes $O(n_i \tilde{\tau}(n_i, \varepsilon)\kappa_i) = O(\varepsilon^{-1} n^{1+\delta} \tilde{\tau}(n_i, \varepsilon))$ steps. Summing over all phases, the total running time is $O(\varepsilon^{-1} n^{1+\delta} \tilde{\tau}(n_i, \varepsilon) \log(1/\delta))$. The same argument as in Lemma 3.2 shows that if $\mathbb{M}$ is the final matching computed by the algorithm, then $\mathtt{w}(\mathbb{M}) \le O(1/\delta^{\alpha})\mathtt{w}(M^*)$ where $\alpha = 1 + \log_2(1 + \varepsilon)$. Again, we guess the value of $\omega$ as in Section 3.

**Data structure.** We now describe the data structure used by Phase-Match algorithm. It uses, as a black-box, a data structure to answer approximate nearest-neighbor (ANN) queries for $\mathtt{d}(\cdot,\cdot)$. As proved in [30], an ANN data structure for $\mathtt{d}(\cdot,\cdot)$ can also answer queries under $\mathtt{d}_i(\cdot,\cdot)$ — i.e., the ceiling operator can be ignored. So, we assume that we have an ANN data structure on $\mathtt{d}_i(\cdot,\cdot)$ and that it can answer a query and delete a point in $\tau(n, \varepsilon)$ time.

For two integers $i, j$, $i \le j$, we use $[i : j]$ to represent all integers between $i$ and $j$; we also view $[i : j]$ as in interval. Two intervals $\mathsf{X} = [i_1 : j_1]$ and $\mathsf{Y} = [i_2 : j_2]$ with $i_2 \ge i_1$ are called *$\rho$-well separated* (or simply well-separated) if $[i_1 : j_1] \cap [i_2 : j_2] = \emptyset$ and $j_1 - i_1, j_2 - i_2 \le \rho(i_2 - j_1)$. We refer to $j_1 - i_1$ as the *length* of $\mathsf{X}$. A family $\mathbb{I} = \{(\mathsf{X}_1, \mathsf{Y}_1), (\mathsf{X}_2, \mathsf{Y}_2), \ldots, (\mathsf{X}_u, \mathsf{Y}_u)\}$ of pairs of intervals, with $\mathsf{X}_i$ lying to the left of $\mathsf{Y}_i$, where $\mathsf{X}_i = [x_i : x'_i]$ and $\mathsf{Y}_i = [y_i : y'_i]$ is called a *$\rho$-well separated pair decomposition* of $[0 : N]$ if

(i) Each $\mathsf{X}_i$ and $\mathsf{Y}_i$ is a $\rho$-well separated pair, and

(ii) for any pair $x, y \in [0 : N]$, with $x < y$, there is an index $i$ such that $x \in \mathsf{X}_i$ and $y \in \mathsf{Y}_i$.

This is the 1-dimensional version of the general well-separated pair decomposition of a point set; see [19, Chapter III]. A standard binary decomposition can generate $\mathbb{I}$ such that (a) $u = O(N/\rho)$, and (b) for any integer $x \in [0 : N]$, there are $O(\rho^{-1} \log N)$ pairs in $\mathbb{I}$ such that $x$ is contained in one of the two intervals in the pair.

Suppose $K \subseteq A_i$ is the set of $m$ points on which we wish to build the data structure. We set $\rho = \varepsilon/10$ and $N = \frac{5}{\varepsilon} n$ (it can be shown that the maximum absolute value of any dual weight is at most $\frac{5}{\varepsilon} n$), and construct, in $O(n/\varepsilon^2 \log(n/\varepsilon))$ time, a WSPD $\mathbb{I}$ of $[0 : N]$. For every pair in $(\mathsf{X}_i, \mathsf{Y}_i) \in \mathbb{I}$, let $K_i = \{a \mid a \in A_i, |y(a)| \in \mathsf{X}_i\}$; recall that dual weights of all points of $A$ are non-positive but we consider their absolute values. For every $i$, we maintain a dynamic $(\varepsilon/2)$-ANN data structure on $K_i$. Let $\tau(m, \varepsilon)$ be the query and update time of this data structure. Delete operation is straightforward and takes $\tilde{\tau}(m, \varepsilon) = O(\varepsilon^{-1} \log(n/\varepsilon) \tau(m, \varepsilon))$ time. Eligible($b$) works as follows: Identify all pairs $(\mathsf{X}_i, \mathsf{Y}_i) \in \mathbb{I}$ such that $y(b) \in \mathsf{Y}_i$.[2] Let $Q \subseteq \mathbb{I}$ be this set. For each pair in $(\mathsf{X}_i, \mathsf{Y}_i) \in Q$, retrieve an $(\varepsilon/2)$-ANN $a_i$ of $b$ from the corresponding ANN data structure. If the edge $(a_i, b)$, for any of these retrieved $a_i$'s is weakly or strongly eligible, the procedure returns $a_i$ and Null otherwise. The time taken by the procedure is $O(\varepsilon^{-1} \log(n/\varepsilon) \tau(m, \varepsilon))$.

[2]Note that, for every eligible edge, $\mathtt{d}(\cdot,\cdot)$ is positive, so $y(b) > |y(a)|$ and no need to consider the case where $y(b) \in \mathsf{X}_i$.

The following lemma establishes the correctness of Eligible procedure.

Lemma 4.2. Eligible*($b$) returns* Null *only when there are no strongly eligible edges of the form $(a, b)$ where $a \in K$.*

Proof. Suppose Eligible($b$) returns NULL and there is a strongly eligible edge incident on $b$. Let $(a, b)$ be this edge, i.e., $y(a) + y(b) = \mathtt{d}_i(a, b) + \lceil \varepsilon \mathtt{d}_i(a, b) \rceil$. Since $y(a) \le 0$, we can write this as

$$|y(b)| - |y(a)| \ge (1 + \varepsilon)\mathtt{d}_i(a, b). \tag{19}$$

By property (i) of $\mathbb{I}$ and since $|y(a)|, |y(b)|$ are integers at most $N$, there is a pair $(\mathsf{X}_i, \mathsf{Y}_i) \in \mathbb{I}$ such that $|y(a)| \in \mathsf{X}_i, |y(b)| \in \mathsf{Y}_i$. Since $\mathsf{X}_i$ and $\mathsf{Y}_i$ are $(\varepsilon/10)$-well separated, the length of $\mathsf{X}_i$ (resp. $\mathsf{Y}_i$) is at most $(\varepsilon/10)(|y(b)| - |y(a)|) \le (1 + \varepsilon)(\varepsilon/10)\mathtt{d}_i(a, b) \le (\varepsilon/5)\mathtt{d}_i(a, b)$. Consider the corresponding ANN data structure for $K_i$ and observe that $a$ is in this data structure. Let $a'$ be the $\varepsilon/2$-approximate nearest neighbor of $b$ returned by the ANN data structure. Then $\mathtt{d}_i(a', b) \le (1 + \varepsilon/2)\mathtt{d}_i(a, b)$. Since our procedure returns NULL, $(a', b)$ is not eligible and therefore

$$\begin{aligned} |y(b)| - |y(a')| &\le \mathtt{d}_i(a', b) \le (1 + \varepsilon/2)\mathtt{d}_i(a, b) \\ &\le |y(b)| - |y(a)| - (\varepsilon/2)\mathtt{d}_i(a, b) \end{aligned}$$

or $|y(a')| - |y(a)| \ge (\varepsilon/2)\mathtt{d}_i(a, b)$ which is a contradiction of the fact that the length of interval $\mathsf{X}_i$ that contains the $|y(a)|, |y(a')|$ is $(\varepsilon/10)\mathtt{d}_i(a, b)$. □

Putting everything together, we obtain the following.

Theorem 4.3. *Given two point sets $A$ and $B$ of size $n$ each, a metric $\mathtt{d}(\cdot,\cdot)$ that can be evaluated in $O(1)$ time, two parameters $\varepsilon, \delta \in (0, 1)$, and an ANN data structure under $\mathtt{d}(\cdot,\cdot)$, an $O(1/\delta^{\alpha})$-approximate matching of $A$ and $B$ can be computed in $O(\varepsilon^{-2} n^{1+\delta} \tau(n, \varepsilon) \log^2(n/\varepsilon) \log(1/\delta))$ time, where $\alpha = 1 + \log_2(1 + \varepsilon)$ and $\tau(n, \varepsilon)$ is the query and update time of an $\varepsilon/2$-ANN data structure.*

Using existing dynamic data structures for ANN queries, we obtain the results mentioned in Introduction for geometric settings. Because of lack of space, we omit the details from here.

## 5. MATCHING WITH ANN UNDER AN ARBITRARY COST FUNCTION

We now present an algorithm to compute a $(1+\varepsilon)$-approximate matching for any cost function $\mathtt{d}(\cdot,\cdot)$, provided we have a data structure that can answer $\varepsilon/c$-ANN queries under $\mathtt{d}(\cdot,\cdot)$ in $\tau(n, \varepsilon)$ time, for some $c > 1$. While this algorithm is slower than the one presented in Section 4, it does not require the metric assumption on $\mathtt{d}(\cdot,\cdot)$ and the approximation factor is much smaller.

At a high-level, this algorithm can be viewed as the execution of the phase 0 of the algorithm of Section 4, but instead of running it for at most $\frac{30}{\varepsilon} n^{\delta}$ stages, we let it run for $\sqrt{n}$ stages. When the algorithm stops, it has computed, in $O(\varepsilon^{-1} n^{3/2} \tau(n, \varepsilon) \log n)$ time, a partial $(\varepsilon/2)$-relaxed matching $M_1$ of $A$ and $B$ so that only $O(\sqrt{n})$ points of $A$ and $B$ are free – this is similar to a number of previous algorithms; see e.g. [14, 20, 30]. For the remaining $O(\sqrt{n})$ free points, unlike Section 4, we do not commit $M_1$. Instead we iteratively compute augmenting paths using Hungarian search, starting with $M_1$, and compute a perfect $(\varepsilon/2)$-relaxed matching $M$ of $A$ and $B$. By Lemma 2.1, $M$ is an $\varepsilon$-approximate matching of $A, B$.

In the rest of the section, we focus on the Hungarian search procedure. At each stage, it takes a partial $(\varepsilon/2)$-relaxed matching and its dual weights as input, adjusts the dual weights and computes an augmenting path consisting only of strongly and weakly eligible edges. The search procedure maintains an alternating tree consisting of eligible edges rooted at the free vertices of $B$. Let $S$ be the set of points of $B$ that are in the alternating tree and $T$ be the set of points of $A$ that are not in any alternating tree constructed so far. Initially $S = B^F$ and $T = A$ (or $A \setminus T = \emptyset$). For simplicity, with a slight abuse of notation, we use $\mathtt{d}(\cdot,\cdot)$ to denote $\mathtt{d}_0(\cdot,\cdot)$.

For a pair $(a,b) \in A \times B$, let $\xi(a,b) = \mathtt{d}(a,b) - y(a) - y(b)$. Due to $\varepsilon/2$-relaxed conditions, $\xi(a,b) \geq -(\varepsilon/2)\mathtt{d}(a,b)$. Let $a^*, b^* = \mathrm{argmin}_{a\in T, b\in S}\, \xi(a,b)$ be the *weighted bi-chromatic closet pair* (BCP) of $S$ and $T$. If $S \times T$ contains a strongly eligible edge, then $\xi(a^*,b^*) = -(\varepsilon/2)\mathtt{d}(a,b)$.

We define a weaker notion of *$\delta$-approximate BCP* (or $\delta$-BCP for brevity), which is sufficient for our algorithm. A pair $\tilde{a} \in T$ and $\tilde{b} \in S$ is called a $\delta$-BCP if $\xi(\tilde{a},\tilde{b}) \leq \xi(a,b) + \delta\mathtt{d}(a,b)$ for any pair $(a,b) \in S \times T$. We describe a data structure which using an $(\delta/8)$-ANN data structure maintains an $\delta$-approximate BCP under the following three operations, each of which takes $\tilde{\tau}(n,\varepsilon)$ time.

(i) Insert$(b)$: Insert a weighted point $b$ with weight $y(b)$ to $S$.

(ii) Delete$(a)$: Delete a point $a$ from $T$.

(iii) Wt-Update$(\Delta)$: For each point $b \in S$, set its weight $y(b) = y(b) + \Delta$.

For our algorithm, we set up $\delta = \varepsilon/2$. Using an $(\varepsilon/2)$-BCP data structure, we execute Hungarian search as follows: Let $(\tilde{a},\tilde{b})$ be the pair maintained by the data structure. There are two cases: First, if $\xi(\tilde{a},\tilde{b}) \leq 0$, then $(\tilde{a},\tilde{b})$ is weakly eligible, and we add $(\tilde{a},\tilde{b})$ to the alternating tree and remove $\tilde{a}$ from $T$. If $\tilde{a}$ is unmatched, we have found an augmenting path and the search terminates. Otherwise, let $(\tilde{a}, b')$ be the edge in the current matching. We add $(\tilde{b},\tilde{a})$ and $(\tilde{a},b')$ to the alternating tree, add $b'$ to $S$, and remove $\tilde{a}$ from $T$.

Next, if $\xi(ta,tb) > 0$, i.e., $(\tilde{a},\tilde{b})$ is not eligible, we set the dual weight of every point $b \in S$ to $y(b) = y(b) + \xi(\tilde{a},\tilde{b})$ and the dual weight of every point $a \in A \setminus T$ to $y(a) = y(a) - \xi(\tilde{a},\tilde{b})$. We update the weights in the data structure using the Wt-Update procedure. [3] After this step $(\tilde{a},\tilde{b})$ is weakly eligible, and we proceed as above.

Following the proofs of [1, 30, 32], it can be shown that the search grows alternating tree without violating the $(\varepsilon/2)$-relaxed constraints and terminates when an augmenting path is found. Omitting the details, we conclude that the time spent in computing all $O(\sqrt{n})$ augmenting path is $O(n^{3/2}\tilde{\tau}(n,\varepsilon))$.

Recall that the phase 0 of the algorithm described in Section 4 assumes that we have a value $\omega \in [\mathtt{w}(M^*)/2, \mathtt{w}(M^*)]$. Let $\Delta = \frac{\max \mathtt{d}(a,b)}{\min \mathtt{d}(a,b)}$, where the maximum and minimum are taken over all distinct pairs of $A \times B$, is the *spread* of $A \times B$.

[3]Since points in $A \setminus T$ do not participate in computing $\tilde{a}$ and $\tilde{b}$, we can update their dual weights at the end of the search procedure. The total time taken for this step would be $O(n)$.

Then we run the above algorithm $O(\log\Delta)$ different choices of $\omega$ and choose the minimum-cost matching among them. Putting everything together, the total running time of the algorithm is $O(n^{3/2}\tilde{\tau}(n,\varepsilon)\log\Delta)$.

**BCP data structure.** We now describe the data structure for maintaining a $\varepsilon$-BCP of $S$ and $T$ under the operations (i)–(iii). For a weighted point $q \in A \cup B$ and for a subset $P \subset A \cup B$, we define a *$\delta$-approximate weighted nearest neighbor* (or $\delta$-WNN) in $P$ to be a point $\tilde{p} \in Q$ such that $\xi(q,\tilde{p}) \leq \min_{p\in P} \xi(p,q) + \delta\mathtt{d}(p,q)$. The heart of our data structure is a dynamic data structure for answering $\delta$-WNN queries on a subset of $S$ or of $T$. For now suppose we have such a data structure. Eppstein [10] has described a data structure for maintaining the BCP of two point sets under insertion and deletion of points, under any given distance function provided that there is a data structure for answering the nearest-neighbor queries under that distance function. It basically maintains a set $\mathcal{C}$ of $O(n\log n)$ candidate pairs of points for bi-chromatic closest pair (BCP), using a dynamic data structure for nearest-neighbor queries. It can be shown that if we plug our data structure for answering $\delta$-WNN queries, then his data structure can be adapted in a straightforward manner to maintain a $\delta$-BCP. In particular, if the query and update time of $\delta$-WNN data structure on a set of $m$ points is $O(\varphi(m,\delta))$, then $\delta$-BSP of $S \times T$ can be maintained in $O(\varphi(n,\delta)\log^2 n)$ time. We omit the details from this version.

**$\delta$-WNN data structure.**

XXXX *Details of the data structure.* Here we describe the data structure, and the description of the update procedures is given in Appendix. We begin by defining the notion of *ordered $\varepsilon$-nearest neighbor*, introduced in [10]. For two point sets, $P$ and $Q$, its ordered weighted $\varepsilon$-ANN is a sequence of points $\langle p_1, q_1, p_2, q_2, \ldots\rangle$ such that $p_1$ is an arbitrary point $P$, $q_i$ is a weighted $\varepsilon$-ANN of $p_i$ in $Q \setminus \{q_1,\ldots,q_{i-1}\}$, and $p_{i+1}$ is a weighted $\varepsilon$-ANN of $q_i$ in $P \setminus \{p_1,\ldots,p_{i-1}\}$, for $i \geq 1$. The sequence ends when one of $P$ and $Q$ becomes empty; here the weight of a point is its dual weight. Set $\mathcal{C}(P,Q) = \{(p_i,q_i),(q_i,p_{i+1}) \mid i \geq 1\}$.

As in [10], we partition $S \cup T$ into $\lceil \log_2 m\rceil$ layers, where layer $i$, for $i \geq 0$, contains at most $2^i$ points. Let $S_i \subseteq S$, $T_i \subseteq T$ be the sets of points in layer $i$. For each layer $i$, we construct the set $\mathcal{C}_i = \mathcal{C}(S_i,T) \cup \mathcal{C}(T_i,S)$. Set $\mathcal{C} = \bigcup_{i\geq 0}\mathcal{C}_i$. It can be verified that $\min_{(a,b)\in\mathcal{C}} \xi(a,b) \leq \xi(a^*,b^*) + \varepsilon\mathtt{d}(a^*,b^*)$. For all points $v \in S_i \cup T_i$, we maintain a weight $\tilde{y}(v)$, which does not change until layer $i$ is reconstructed. We also maintain an offset $\Delta_i$ for layer $i$: $y(b) = \tilde{y}(b) + \Delta_i$ for all $b \in S_i$ and $y(a) = \tilde{y}(a)$ for all $a \in T_i$ (recall that the weight of a point in $T$ does not change). Since the value $\xi(a,b)$ for a pair $(a,b) \in \mathcal{C}$ can change over time, we use the proxy-value technique used by Vaidya [32] and others [1, 30], mentioned above, so that $\mathrm{argmin}_{(a,b)\in\mathcal{C}}\, \xi(a,b)$ can be maintained using a priority queue; we omit the details from here and just assume that insert, delete, delete-min, and Wt-Update operations on $\mathcal{C}$ can be performed in $O(\log n)$ time. Finally, to compute the lists $\mathcal{C}(S_i,T)$ and $\mathcal{C}(T_i,S)$, for each $i$, we construct WSPD $\mathbb{I}$ of $[0 : 5n/\varepsilon]$, as in Section 4. For each pair $(\mathsf{X}_j,\mathsf{Y}_j) \in \mathbb{I}$ and for every integer $i \geq 0$, let $S_i^j = \{b \in S_i \mid \tilde{y}(b) \in \mathsf{Y}_j\}$, $\tilde{S}_i^j = \{b \in S_i \mid \tilde{y}(b) \in \mathsf{X}_j\}$, $T_i^j = \{a \in T_i \mid \tilde{y}(a) \in \mathsf{X}_j\}$ and $\tilde{T}_i^j = \{b \in T_i \mid \tilde{y}(a) \in \mathsf{Y}_j\}$.[4] We preprocess $S_i^j$, $\tilde{S}_i^j$, $T_i^j$ and

[4]Unlike in Section 4, we create ANN structure for all the four

$\tilde{T}_i^j$ each in separate data structures for answering unweighted $(\varepsilon/4)$-ANN queries, i.e., for point $q$, it returns a point $\tilde{b}$ of $S_i^j$ such that $\mathtt{d}(q, \tilde{b}) \leq (1 + \varepsilon/4) \min_{b \in S_i^j} \mathtt{d}(q, b)$, and the same is true for the other three sets.

This completes the description of the data structure. Given a point $a \in T$ (resp. $b \in S$) and a layer $i$, its weighted $\varepsilon$-ANN in $S_i$ (resp. $T_i$) can be computed in $O(\varepsilon^{-1}\tau(n, \varepsilon) \log(n/\varepsilon))$ time. Using this as a subroutine, the set $\mathcal{C}$ can be initially constructed in $O(\varepsilon^{-1} n\tau(n, \varepsilon) \log(n/\varepsilon))$ time. The Insert, Delete, operations are similar to those in [10] and can be performed in $O(\varepsilon^{-1}\tau(n, \varepsilon) \log^3(n/\varepsilon))$ amortized time. Finally, Wt-Update procedure simply updates the off-set $\Delta_i$ of each layer. We provide the details of these operations in the appendix. Putting everything together we obtain:

Theorem 5.1. *Given* $\mathrm{G}(A \cup B, A \times B)$ *and a distance function* $\mathtt{d}(\cdot, \cdot)$ *on the edges, for a parameter* $0 < \varepsilon \leq 1$*, a* $(1 + \varepsilon)$*-approximate matching of* $A$ *and* $B$ *can be computed in* $O(\varepsilon^{-1} n^{3/2} \tau(n, \varepsilon) \log^3(n/\varepsilon) \log \Delta)$ *time, where* $\tau(n, \varepsilon)$ *is the query and update time of an* $\varepsilon/c$*-approximate nearest neighbor data structure, for some constant* $c > 1$*.*

**Acknowledgments.** Work by Pankaj Agarwal is supported by NSF under grants CCF-09-40671, CCF-10-12254, and CCF-11-61359, by an ARO grant, by Grant 2012/229 from the U.S.-Israel Binational Science Foundation, and by an ERDC contract W9132V-11-C-0003. Work by R. Sharathkumar is supported by grant FA9550-12-1-XXXX from the U.S. Air Force Office of Scientific Research (AFOSR) and the Defense Advanced Research Projects Agency (DARPA).

## 6. 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 (1999), 39–50.

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

[3] A. Andoni, P. Indyk, and R. Krauthgamer, Earth mover distance over high-dimensional spaces, *Proc. 19th Annual ACM-SIAM Sympos. Discrete Algo.*, 2008, pp. 343–352.

[4] S. Arora, Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems, *J. ACM*, 45 (1998), 753–782.

[5] D. Avis, Two greedy heuristics for the weighted matching problem, *Proc. of 9th S. E. Conf. on Comb., Graph Theory, and Computing*, 1978, pp. 65–76.

[6] H.-T. Chen, H.-H. Lin, and T.-L. Liu, Multi-object tracking using dynamical graph matching, *IEEE Conference Comp. Vision Pattern Recog.*, 2 (2001), 210.

[7] R. Cole, R. Hariharan, M. Lewenstein, and E. Porat, A faster implementation of the Goemans-Williamson clustering algorithm., *Proc. 11th Annual Sympos. Discrete Algo.*, 2001, pp. 17–25.

[8] M. Cygan, H. N. Gabow, and P. Sankowski, Algorithmic applications of Baur-Strassen's theorem: Shortest cycles, diameter and matchings, *Proc. 54th Annual IEEE Sympos. Found. Comp. Sci.*, (2012), 531–540.

[9] R. Duan and S. Pettie, Linear-time approximation for maximum weight matching, *J. ACM*, 61 (2014), 1:1–1:23.

[10] D. Eppstein, Dynamic euclidean minimum spanning trees and extrema of binary functions, *Discrete Comput. Geom.*, 13 (1995), 111–122.

[11] J. Fakcharoenphol, S. Rao, and K. Talwar, A tight bound on approximating arbitrary metrics by tree metrics, *Proc. 35th Annual ACM Sympos. Theory Comput.*, 2003, pp. 448–455.

[12] M. L. Fredman and R. E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, *J. ACM*, 34 (1987), 596–615.

[13] H. Gabow and S. Pettie, The dynamic vertex minimum problem and its application to clustering-type approximation algorithms, in: *Proc. 8th Scandinavian Workshop Algo. Theory*, 2002, pp. 190–199.

[14] H. N. Gabow and R. Tarjan, Faster scaling algorithms for network problems, *SIAM J. Comput.*, 18 (1989), 1013–1036.

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

[16] A. Goel, P. Indyk, and K. Varadarajan, Reductions among high dimensional proximity problems, *Proc. 12th Annual ACM-SIAM Sympos. Discrete Algo.*, 2001, pp. 769–778.

[17] A. Goel, M. Kapralov, and S. Khanna, Perfect matchings in O($n$ log $n$) time in regular bipartite graphs, *Proc. 42nd Annual Sympos. Theory Comput.*, 2010, pp. 39–46.

[18] M. Goemans and D. Williamson, A general approximation technique for constrained forest problems, *SIAM J. Comput.*, 24 (1995), 296–317.

[19] S. Har-peled, *Geometric Approximation Algorithms*, American Mathematical Society, 2011.

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

[21] P. Indyk, A near linear time constant factor approximation for euclidean bichromatic matching (cost), *Proc. 18th Annual ACM-SIAM Sympos. Discrete Algo.*, 2007, pp. 39–42.

[22] A. Madry, Navigating central path with electrical flows: from dlows to matchings, and back point clouds, *Proc. 54th Annual IEEE Sympos Foundat. Comp. Sc.*, 2013, pp. 253–262.

[23] S. Micali and V. V. Vazirani, An o(sqrt(v)e) algorithm for finding maximum matching in general graphs, *Proc. 21st Annual IEEE Sympos. Foundat. Comp. Sc.*, 1980, pp. 17–27.

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

[25] D. A. Plaisted, Heuristic matching for graphs satisfying the triangle inequality, *J. Algorithms*, 5 (1984), 163 – 179.

[26] E. M. Reingold and R. E. Tarjan, On a greedy heuristic for complete matching, *SIAM J. Comput.*, (1981), 676–681.

sets because we can no longer guarantee that $y(b) \geq |y(a)|$, and furthermore we update weights in a lazy manner.

[27] P. Sankowski, Weighted bipartite matching in matrix multiplication time, in: *Proc. 33rd Intn. Conf. Automata, Languages and Programming*, Vol. 4051, 2006, pp. 274–285.

[28] P. Sankowski, Maximum weight bipartite matching in matrix multiplication time, *Theo. Comput. Sci.*, 410 (2009), 4480 – 4488.

[29] R. Sharathkumar, A sub-quadratic algorithm for bipartite matching of planar points with bounded integer coordinates, *Proc. 29th Annual Sympos. Comput. Geom.*, 2013, pp. 9–16.

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

[31] R. Sharathkumar and P. K. Agarwal, A near-linear time approximation algorithm for geometric biparitte matching., *Proc. 44th Annual ACM Annual Sympos. Theory Comput.*, 2012, pp. 385–394.

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

[33] K. R. Varadarajan, A divide-and-conquer algorithm for min-cost perfect matching in the plane, *Proc. 39th Annual IEEE Sympos. Foundat. Comp. Sc.*, 1998, pp. 320–331.

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