A Near-Linear Constant-Factor Approximation for Euclidean Bipartite Matching?*

Pankaj K. Agarwal† Kasturi R. Varadarajan‡

ABSTRACT

In the Euclidean bipartite matching problem, we are given a set R of "red" points and a set B of "blue" points in $\mathbb{R}^d$ where $|R| = |B| = n$, and we want to pair up each red point with a distinct blue point so that the sum of distances between the paired points is minimized. We present an approximation algorithm that given any parameter $0 < \varepsilon < 1$ runs in $O(n^{1+\varepsilon})$ expected time and returns a matching whose expected cost is within a multiplicative factor $O(\log(1/\varepsilon))$ of the optimal. The dimension d is considered to be a fixed constant.

Categories and Subject Descriptors: F.2.2 [Theory of Computation]: Nonnumerical Algorithms and Problems—*geometrical problems and computations*; G.2.1 [Discrete Mathematics]: Combinatorics—*combinatorial complexity*

General Terms: Theory, Combinatorial Optimization

Keywords: Matching, approximation algorithms

1. Introduction

In the Euclidean bipartite matching problem, we are given a set R of "red" points and a set B of "blue" points in $\mathbb{R}^d$ where $|R| = |B| = n$, and we want to pair up each red point with a distinct blue point so that the sum of distances between the paired points is minimized. This is a well known geometric optimization problem that has applications in operations research, pattern recognition, shape matching, statistics, and VLSI.

Related work. Euclidean bipartite matching problem is a special case of the classical bipartite matching problem in a graph. The first polynomial time algorithm in the graph setting is the famous "Hungarian" algorithm due to Kuhn [9]. The fastest known implementation of this algorithm runs in $O(|V|^3)$ time on dense graphs (see Lawler [10]) and roughly $O(|E||V|)$ time on sparse graphs [8], where $|V|$ and $|E|$ are respectively the number of vertices and edges in the graph. There is a scaling algorithm due to Gabow and Tarjan [7] that runs in $O(\sqrt{|V|}|E|\log(|V|N))$ time, where N is the largest weight of an edge in the graph (weights are assumed to be integers). For the two-dimensional Euclidean version of this problem, Vaidya [13] showed that geometry can be exploited to get algorithms running in $O(n^{5/2}\log^{O(1)} n)$ time. Agarwal et al. [1] improved the running time for the bipartite case to $O(n^{2+\delta})$, for any $\delta > 0$. Agarwal and Varadarajan [2] gave an $(1+\varepsilon)$-approximation algorithm for this problem that returns, for any $0 < \varepsilon < 1$, a perfect matching whose cost is at most $(1+\varepsilon)$ times the optimal in $O((n^{3/2}/\varepsilon^2)\log^5(n/\varepsilon))$ time. This is a geometric implementation of the scaling algorithm mentioned above. No algorithm with a better running time is known for computing even a constant factor approximation to the optimal matching. We restrict our attention to surveying the two-dimensional Euclidean case because this is a good indication of the state of the art. The best algorithms in any fixed dimension are obtained by a straightforward translation of the two-dimensional algorithms.

A generalization of the Euclidean bipartite matching is the so-called *transportation* problem, in which we are given two sets of points U and V in $\mathbb{R}^2$ and a positive integral *demand* $\lambda(p)$ for each $p \in U \cup V$ so that

$$\sum_{u\in U} \lambda(u) = \sum_{v\in V} \lambda(v).$$

A feasible solution to this problem is a subset $M \subseteq U \times V$ of edges and positive integral *weights* $w(u,v)$ for each $(u,v) \in M$ such that

$$\lambda(p) = \sum_{(p,q)\in M} w(p,q) \quad \forall p \in U$$

$$\lambda(q) = \sum_{(p,q)\in M} w(p,q) \quad \forall q \in V$$

The goal is to find a feasible solution M, w that minimizes

$$\sum_{(u,v)\in M} w(u,v)d(u,v).$$

Here $d(u,v)$ is the Euclidean distance between u and v. The bipartite matching problem is a special case of the transportation problem in which all demands are 1. Atkinson and Vaidya [4] presented an algorithm to solve the transportation problem in time $O(k^{2.5}\log k \log N)$ where $k = |U| + |V|$ and N is the maximum demand.

*Research by the first author is supported by NSF under grants CCR-00-86013, EIA-98-70724, EIA-01-31905, and CCR-02-04118, and by a grant from the U.S.–Israel Binational Science Foundation. Research by the second author is supported by NSF CAREER award CCR-0237431

† Department of Computer Science, Box 90129, Duke University, pankaj@cs.duke.edu

‡ Department of Computer Science University of Iowa, kvaradar@cs.uiowa.edu

SCG'04, June 8–11, 2004, Brooklyn, New York, USA

The nonbipartite version of Euclidean matching, where we are given a set P of $2n$ points in $\mathbb{R}^d$ and we want to pair up the points into n pairs so as to minimize the sum of the distances between paired points, is also widely studied. The first polynomial-time algorithm for the graph version of this problem is the classical algorithm due to Edmonds [6]. The best implementations of this algorithm and the best scaling algorithm have running times similar to the bipartite case. Vaidya [13] gave an algorithm for the two-dimensional Euclidean version of this problem that runs in $O(n^{5/2}\log^{O(1)} n)$ time. Varadarajan [14] later gave a divide-and-conquer algorithm that runs in $O(n^{3/2}\log^{O(1)} n)$ time. For the approximate version of this problem, Vaidya [12] gave an algorithm that, for any $\varepsilon > 0$, runs in roughly $O(n^{3/2}/\varepsilon^3)$ time and returns a $(1+\varepsilon)$-approximate perfect matching. In a seminal paper that gave improved algorithms for many geometric optimization problems like the TSP, Arora [3] gave a Monte-Carlo algorithm that runs in $O(n\log^{O(1/\varepsilon)} n)$ time and returns a $(1+\varepsilon)$-approximate matching with high probability. Building on his approach, Rao and Smith [11] give a Monte-Carlo algorithm that runs in $O(n\log n)$ time and produces (with probability atleast $1/2$) a matching whose cost is within a constant factor of the optimal. Agarwal and Varadarajan [2], also building on Arora's approach, gave a Monte Carlo algorithm that returns a $(1+\varepsilon)$-approximate matching with probability at least $1/2$ in $O((n/\varepsilon^3)\log^6 n)$ time.

From the discussion of the state of the art, it appears that bipartite matching is harder than nonbipartite matching in the geometric setting. This seems counterintuitive at first but a little reflection reveals that the bipartite case can be more "non-local" than the nonbipartite case. Indeed the near-linear approximation algorithm due to Arora [3], which is based on a hierarchical decomposition of a point set by a randomly shifted quadtree, does not extend to the bipartite case. One source of difficulty is that when a cell of the quadtree is divided into four cells, the number of edges of even an approximate matching that cross the subdividing lines may be much larger than a constant or $\log n$. This makes the number of subproblems in the natural dynamic programming approach too large, and it is not clear how to get around this difficulty.

Nevertheless the general feeling among researchers has been that a near-linear time algorithm that gives at least a constant-factor approximation must exist, and that the subdivision due to a randomly shifted quadtree should be a useful tool.

Our results. In this paper, we make substantial progress towards realizing the above intuition: we give a Monte Carlo algorithm for the two-dimensional bipartite matching problem that, for any $0 < \varepsilon < 1$, runs in $O(n^{1+\varepsilon})$ expected time and returns a matching whose expected cost is within $O(\log(1/\varepsilon))$ of the optimal. Thus the closer our asymptotic running time gets to $O(n)$, the larger is the constant in the constant-factor approximation we get.

Our algorithm uses a variant of the idea of the randomly shifted quadtree. When a cell of the quadtree is subdivided into "subcells", we compute a matching in which the number of edges that "cross" a subcell is the minimum number that needs to in any matching (due to an imbalance between the number of red and blue points in the subcell). We resolve the question of which points of the subcell are to be matched outside the subcell by picking an arbitrary subset of the right size from the points of the predominant color. We bound the expected increase in the cost of the matching that we compute using the fact that we are using a probabilistic partition. To ensure that the overall increase in cost is not too much we make sure that the number of levels in the quadtree is $O(\log(1/\varepsilon))$. To do this we allow a cell of the quadtree to be partitioned into a large number of subcells, not just 4. The size of the subproblems in the "merge" step may be quite large but we reduce this problem to a small-sized transportation problem. Our analysis of the cost of the matching computed by our algorithm has some new ideas which may be useful elsewhere.

Organization. In Section 2 we define the problem more carefully and state some preliminary lemmas and results that we will subsequently use. To simplify the presentation we first present in Section 3 an algorithm that runs in $O(n^{1+\varepsilon})$ expected time and returns a matching whose expected cost is within $O(1/\varepsilon)$ of the optimal. In Section 4 we describe our improved algorithm that returns a matching whose expected cost is within $O(\log \frac{1}{\varepsilon})$ of the optimal. We will restrict our exposition to the two-dimensional version. Our algorithms and their analysis readily generalize to any fixed dimension.

2. Preliminaries

Let R be a set of n "red" points and B a set of n "blue" points in $\mathbb{R}^2$. A *perfect bipartite matching* of $P = R \cup B$ is a subset $M \subseteq R \times B$ of red-blue pairs such that each point in P is present in exactly one pair of M; we refer to a perfect bipartite matching as simply a matching. Obviously, $|M| = n$. We define the *cost* of a matching M of P to be

$$\mu(P, M) = \sum_{(u,v)\in M} d(u, v),$$

where $d(u, v)$ is the Euclidean distance between u and v. If the set P is fixed or obvious from the context, we will use $\mu(M)$ to denote $\mu(P, M)$. Let

$$\mu(P) = \min_M \mu(P, M)$$

denote the cost of the min-cost matching of P, and let $\mathrm{M}^*(P)$ be a min-cost matching of P.

We begin with the following simple observations.

LEMMA 2.1. *Let $P = R \cup B$, let P' be the point set obtained by "moving" each point $p \in P$ to a point p', and let $\Delta = \sum_{p\in P} d(p, p')$.*

(i) Let M be any perfect matching of P, and let

$$M' = \{(p', q') \mid (p, q) \in M\}$$

be the corresponding perfect matching of P'. Then

$$|\mu(M') - \mu(M)| \le \Delta.$$

(ii) Let M be the matching in P corresponding to the optimal perfect matching of P'. Then

$$\mu(M) \le \mu(P) + 2\Delta.$$

The following lemma suggests how to compute a rough approximation of $\mu(P)$.

LEMMA 2.2. *Let R be a set of n red points and B a set of n blue points in $\mathbb{R}^2$; set $P = R \cup B$. We can compute in $O(n\log n)$ time a number α such that*

$$\alpha \le \mu(P) \le 2n^2\alpha. \tag{1}$$

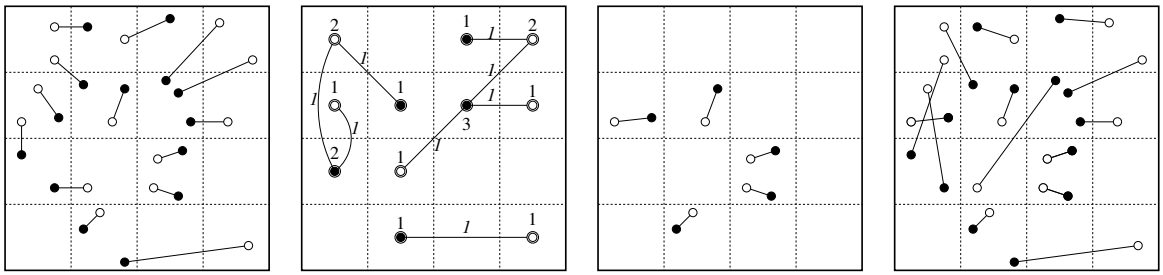

Figure 1. A recursive step of the matching algorithm: (i) An input set and its min-cost matching. (ii) The transportation problem corresponding to Q; the numbers near the points are their demands, and the numbers near the arcs are the edge weights in the solution of the transportation problem. (iii) A recursive solution for each cell in the grid. (iv) The output matching.

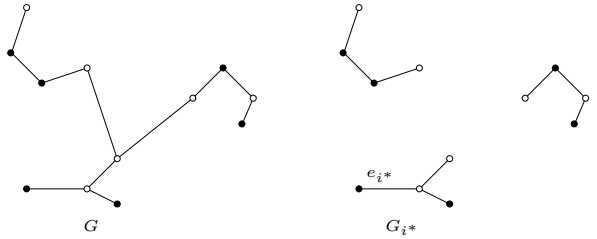

Figure 2. Computing a rough approximation of $\mu(P)$.

PROOF. We compute in $O(n \log n)$ time the minimum spanning tree T of P (ignoring the colors), under the L_∞-metric, using the algorithm by Callahan and Kosaraju [5]. Let $e_1, \ldots, e_{2n-1}$ be the edges of T in increasing order of their lengths. For $0 \leq i \leq 2n-1$, let G_i denote the subgraph induced by the edges $e_1, \ldots, e_i$, and let i^* be the smallest integer for which each component of G_{i^*} has equal number of red and blue points. Given the ordering of the edges, i^* can be computed in $O(n)$ time. The length of e_{i^*} is the desired value of α. See Figure 2.

Indeed, the graph G_{i^*-1} has at least one connected component C in which the number of red and blue points is not the same. So any perfect matching M of P has an edge e that has one endpoint in C and another endpoint in a component of G_{i^*-1} different from C. By a well known property of MSTs, $\|e\|_\infty \geq \|e_{i^*}\|_\infty$. Moreover, $\|e\|_2 \geq \|e\|_\infty$, we conclude that $\mu(P) \geq \alpha$.

Every connected component of G_{i^*} has the same number of red and blue points. We construct a perfect matching M' of P by finding an arbitrary perfect matching for the points within each component. Note that for each edge $(u, v) \in M'$ there is a path between u and v in G_{i^*}. Since each edge of G_{i^*} has length at most α, we conclude from the triangle inequality that $\|uv\|_\infty \leq n\alpha$. Thus $d(u, v) \leq 2n\alpha$ and $\mu(M') \leq 2n^2\alpha$. □

For a parameter $\delta > 0$, let $\mathbb{G}_\delta$ be the square grid formed by the horizontal lines $y = i\delta$ and the vertical lines $x = j\delta$, where $i, j \in \mathbb{Z}$. We define a *random shift* of $\mathbb{G}_\delta$ to be the grid formed by the lines $y = i\delta + a_x$ and $x = j\delta + a_y$, where a_x, a_y are two independently chosen random numbers in the interval $[0, \delta)$.

3. The Algorithm

In this section, we describe an algorithm that, given the input set $P = R \cup B$ of $2n$ points and a parameter $\varepsilon > 0$, runs in $O(n^{1+\varepsilon})$ expected time and returns a matching of P whose expected cost is at most $O(1/\varepsilon)$ times the optimal. We assume that the point set P is enclosed in a bounding square E. The algorithm is a call to the following procedure $Match$ with parameters P and E. The algorithm is described in a way that will make it easy to describe the modifications needed to obtain the improved algorithm. Throughout the algorithm, n will denote $|R| = |B|$.

Procedure $Match(S, D)$.

1. If $m = |S|/2$ is smaller than some constant, then compute an optimal matching of S using the Hungarian algorithm and and return this matching.
2. Using the algorithm of Lemma 2.2, we first compute an approximation α to $\mu(S)$ such that $\alpha \leq \mu(S) \leq 2m^2\alpha$.
3. If $2m^5\alpha$ is greater than $1/8$ times the side-length of D, we compute a matching of S by making a call to the procedure $SubMatch$ with parameters S, D, α, m and return this matching. Otherwise, we take a random shift of the grid $\mathbb{G}_{2m^5\alpha}$. Let C denote the set of grid cells that intersect D. For each grid cell $C \in \mathrm{C}$, let $S_C = S \cap C$, let $\chi(C) = ||S_C \cap R| - |S_C \cap B||$. If S_C contains more red points than blue points (resp. blue points than red points) we arbitrarily pick $\chi(C)$ red points (resp. blue points) and denote the set by Q_C. Let $Q = \cup_{C \in \mathrm{C}} Q_C$. Note that Q contains an equal number of red and blue points. We compute a perfect matching of the points in Q as follows. For each cell C, we "move" each point in Q_C to the center of the grid cell C. We compute an optimal perfect matching for the moved points using the Hungarian algorithm; let $\overline{M}$ denote the corresponding matching of Q.
4. For each cell $C \in \mathrm{C}$ for which $S_C - Q_C$ is nonempty, we compute a perfect matching $\overline{M}_C$ of the points $S_C - Q_C$ using a call to the subroutine $SubMatch$ with parameters $S_C - Q_C, C, \alpha, m$. We return the matching $\overline{M} \cup \bigcup_{C \in \mathrm{C}} \overline{M}_C$.

The subroutine $SubMatch$ is a recursive procedure that takes as input a point set S consisting of an equal number of red and blue points, a box D containing S, and parameters α and m. Note that m here will be set to the size of the point set in the original $Match$ routine that invoked $SubMatch$; and α will be the crude approximation computed by this $Match$ routine. The subroutine $SubMatch$ will compute a perfect matching of S.

Procedure $SubMatch(S, D, \alpha, m)$

1. Let L denote the side length of D. If $L \leq \alpha/m^2$, we compute an arbitrary perfect matching of S and return it.

2. Let $\delta = \varepsilon/12$. If $|S|/2 \leq n^{6\delta}$, we compute a perfect matching of S using the Hungarian algorithm and and return it.

3. We take a random shift of the grid $\mathbb{G}_{L/\max\{8,m^\delta\}}$. Let $\mathbb{C}$ denote the set of grid cells that intersect D. For each grid cell $C \in \mathbb{C}$, let $S_C = S \cap C$, let $\chi(C) = ||S_C \cap R| - |S_C \cap B||$. If S_C contains more red points than blue points (resp. blue points than red points) we arbitararily pick $\chi(C)$ red points (resp. blue points) and denote the set by Q_C. Let $Q = \cup_{C \in \mathbb{C}} Q_C$. Note that Q contains an equal number of red and blue points. We compute a perfect matching of the points in Q as follows. For each cell C, we "move" each point in Q_C to the center of the grid cell C. We compute an optimal perfect matching for the moved points using an algorithm for the transportation problem; let $\overline{M}$ denote the corresponding matching of Q.

4. For each cell $C \in \mathbb{C}$ for which $S_C - Q_C$ is nonempty, we compute a perfect matching $\overline{M}_C$ of the points $S_C - Q_C$ using a call to the subroutine $SubMatch$ with parameters $S_C - Q_C, C, \alpha, m$. We return the matching $\overline{M} \cup \bigcup_{C \in \mathbb{C}} \overline{M}_C$.

Running time analysis

Step 2 of procedure $Match$ takes $O(m \log m)$ time. The expected running time of Step 3 is $O(m)$, because $|Q| = \sum_{C \in \mathbb{C}} \chi(C)$ is bounded by the number of edges of the optimal perfect matching of S that cross the grid lines, and the probability that the latter number is greater than 0 is at most $1/m^3$ due to the large grid size. Thus the running time is $O(m^3)$ (for running the Hungarian algorithm) with probability at most $1/m^3$ and is $O(m)$ otherwise. The expected time is thus linear. The overall expected running time of $Match$ is thus $O(m \log m)$.

The running time of Step 2 of procedure $SubMatch$ is $O(|S|^3)$, where $|S| \leq n^{6\delta}$. Thus the contribution of Step 2 to the overall overall running time is $\sum_i O(n_i^3)$ given that $\sum_i n_i \leq n$ and each $n_i \leq n^{6\delta}$. Thus the cost of Step 2 overall is $O(n^{1+12\delta})$. In step 3, the size of Q may be quite large but the size of the moved point set, not counting multiplicities, is only $O(m^{2\delta}) = O(n^{2\delta})$ (because the number of grid cells in $\mathbb{C}$ is $O(m^{2\delta})$). Thus solving this matching problem by running the algorithm for the transportation problem due to Atkinson and Vaidya [4] takes $O(n^{5\delta} \log^2 n)$ time which is $O(n^{6\delta})$. This is bounded by the size of the point set S. Thus the running time of $SubMatch$ is linear in $|S|$ if we ignore Step 2.

The number of levels in the recursion is $O(1/\delta)$ because the size of the bounding box when $SubMatch$ is first invoked is at most $16m^5\alpha$, the bounding box size falls by a factor of at least m^δ with each level of the recursion, and the smallest bounding box size is α/m^2. So the overall expected running time of the algorithm is $O(m \log m + m/\delta + n^{1+12\delta})$. Note that the third term comes from Step 2 of $SubMatch$. With our choice of $\delta = \varepsilon/12$ the overall expected running time is $O(n^{1+\varepsilon})$.

Remark 3.1 The procedures $Match$ and $SubMatch$ are quite similar. The purpose of $Match$ is to handle the scenario when the initial bounding box of the input point set is too large.

Quality of the matching produced

In analyzing the quality of the matching produced, it will be convenient to speak of the hierarchical subdivision or the generalized quadtree that the procedures $Match$ and $SubMatch$ together produce. The root node of this subdivision is associated with the input point set P and its bounding square. In general, a node of the subdivision is associated with a point set $S \subseteq P$ and a square D containing S. If this is a leaf of the subdivision (corresponding to Step 1 of $Match$ and Steps 1 and 2 of $SubMatch$), the algorithm directly computes a matching of the point set S. If this is an internal node of the subdivision, the algorithm uses a randomly shifted grid of an appropriate size to break up D into a set of cells $\mathbb{C}$, computes a matching of a subset $Q \subseteq S$ of points that are then discarded, and recursively computes a matching for the points $S_C - Q_C$ within each cell $C \in \mathbb{C}$. Thus there is a node of the subdivision for each cell C for which $S_C - Q_C$ is non-empty, and each such node becomes a child of the current node.

For any node v of the subdivision Ξ that is produced by the algorithm, let S_v be the associated set of points and D_v the bounding square. If v is an internal node of Ξ, let $Q_v \subseteq S_v$ denote the set of "discarded" points, let $Z_v = S_v - Q_v$, let $\mathbb{C}_v$ denote the matching of Q_v computed by our algorithm, let $\mathbb{C}_v$ denote the set of cells into which D_v is subdivided, and let λ_v denote the side-length of any cell in $\mathbb{C}_v$. Let $\mathbb{V}_0$ denote the set of leaves of Ξ and $\mathbb{V}_1$ the set of internal nodes.

Let $\mathbb{M}$ denote the optimal perfect matching of the input set of points P. For the sake of analysis, we describe a scheme for constructing a perfect matching $\mathbb{N}_v$ for the points S_v associated with each leaf $v \in \mathbb{V}_0$. Let $M^l = \bigcup_{v \in \mathbb{V}_0} \mathbb{N}_v$. Let $M^d = \bigcup_{v \in \mathbb{V}_1} \mathbb{C}_v$. Clearly, $M^l \cup M^d$ is a perfect matching of P. The construction is best viewed as a scheme that converts $\mathbb{M}$ into $M^l \cup M^d$.

The conversion scheme. We visit Ξ in a top down manner (in a post-order fashion). At each node v we have a matching $\mathbb{I}_v$ of S_v. For the root u of $\mathbb{T}$, $\mathbb{I}_u = \mathbb{M}$. If v is an internal node, we process $\mathbb{I}_v$ in two stages, each of which involves performing a sequence of edge swaps. Let $\mathbb{I}_v^E \subseteq \mathbb{I}_v$ denote the subset of edges that are "cut" by the subdivision of D_v into $\mathbb{C}_v$, that is, those edges whose endpoints lie in different cells of $\mathbb{C}_v$.

`Stage I.` Let $M' = \mathbb{I}_v$ initially. We repeat the following step till we are done: While there are two edges (r_1, b_1) and (r_2, b_2) in M' such that both edges are cut by the subdivision of D_v into $\mathbb{C}_v$, r_1 and b_2 are both in the same cell of $\mathbb{C}_v$ and are of opposite color, we replace these by (r_1, b_2) and (r_2, b_1); see Figure 3. At the end of this stage, exactly $\chi(C) = |Q_v \cap C|$ edges of M' from each cell $C \in \mathbb{C}_v$ are cut by the subdivision into $\mathbb{C}_v$.

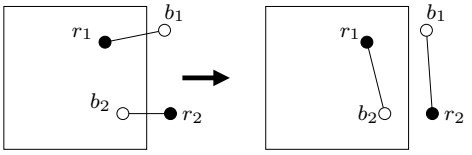

Figure 3. Swapping an edge in Stage I.

`Stage II.` For each cell $C \in \mathbb{C}_v$, we repeat the following step till we are done: if there is in M' a cut edge (r_2, b_2) where $r_2 \in Z_v \cap C$ and (by necessity) a non-cut edge (r_1, b_1) where $r_1 \in Q_v \cap C$ has the same color as r_2, we replace these edges by the cut edge (r_1, b_2) and the non-cut edge (r_2, b_1); see Figure 4.

Let $\overline{\mathbb{I}}_v$ be the matching M' at node v after having performed edge swaps in Stage I and II. Clearly, $\overline{\mathbb{I}}_v = \mathbb{C}'_v \cup \mathbb{L}_v$, where $\mathbb{C}'_v$ is a matching of Q_v and $\mathbb{L}_v$ is a matching of Z_v. Furthermore, no edge of $\mathbb{L}_v$ is cut by the subdivision of D_v into $\mathbb{C}_v$. That is, for each $C \in \mathbb{C}_v$, the restriction of $\mathbb{L}_v$ to $Z_v \cap C$ is a matching of $Z_v \cap C$ (and indeed constitutes the input matching $\mathbb{I}_w$ of $S_w = Z_v \cap C$ for the node w in Ξ corresponding to C).

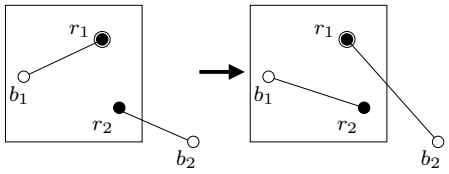

Figure 4. Swapping an edge in Stage II.

LEMMA 3.2. *After having processed an internal node $v \in \Xi$, we have the following:*

(i) $\mu(\mathbb{L}_v) + \mu(\mathbb{C}_v) \leq \mu(\mathbb{I}_v) + c\lambda_v|\mathbb{I}_v^E|$, *where $c > 0$ is a constant.*

(ii) $|\mathbb{L}_v \setminus \mathbb{I}_v| \leq 3|\mathbb{I}_v^E|$.

PROOF. Each step in Stage I decreases the number of cut edges in M' by at least one and increases the cost of the matching by at most $2\lambda_v$. Thus the number of steps is at most $|\mathbb{I}_v^E|$ and the cost of the matching has increased by at most $2\lambda_v|\mathbb{I}_v^E|$ in this stage. At the end of the stage, exactly $\chi(C)$ edges of M' from each cell $C \in \mathrm{C}_v$ are cut by the subdivision into C_v.

The number of times the step in Stage II is performed over all cells in C_v is at most $|Q_v| \leq 2|\mathbb{I}_v^E|$, and each step increases the cost of the matching by at most $2\lambda_v$. It is therefore clear that

$$\mu(\mathbb{L}_v) + \mu(\mathbb{C}'_v) \leq \mu(\mathbb{I}_v) + c'\lambda_v|\mathbb{I}_v^E|.$$

for some constant $c' > 0$. Now

$$\mu(\mathbb{C}_v) \leq \mu(\mathbb{C}'_v) + 2\lambda_v|Q_v| \leq \mu(\mathbb{C}'_v) + 8\lambda_v|\mathbb{I}_v^E|,$$

where the first inequality follows from Lemma 2.1 (ii). This completes the proof of (i). We introduced at most one new edge into M' in each step of stage 1 and stage 2. Hence the overall number of new edges is at most $3 * |\mathbb{I}_v^E|$, completing the proof of (ii). □

It is clear that at the end of the traversal, we have the matching $M^l \cup M^d$ as stated. Indeed, for any leaf $v \in \mathbb{V}_0$, $\mathbb{N}_v$ is going to be $\mathbb{I}_v$. The significance of $M^l \cup M^d$ is that if our algorithm were to compute an *optimal* matching for the points S_v associated with each leaf $v \in \mathbb{V}_0$, then the cost of the overall matching computed by our algorithm would be at most $\mu(M^l \cup M^d)$.

We therefore wish to bound $\mu(M^l \cup M^d) - \mu(\mathbb{M})$. From Lemma 3.2 (i), we see that $\mu(M^l \cup M^d) - \mu(\mathbb{M})$ is at most $\bigcup_{v \in \mathbb{V}_1} c\lambda_v|\mathbb{I}_v^E|$. We account for this by charging $c\lambda_v$ to each edge in $\mathbb{I}_v^E$ for each internal node $v \in \mathbb{V}_1$.

To bound the total charge, we do the following for each internal node $v \in \mathbb{V}_1$: For each edge $e \in \mathbb{I}_v^E$, we pick up to three edges from $\mathbb{L}_v \setminus \mathbb{I}_v$ and call these the *children* of e. We ensure that each edge in $\mathbb{L}_v \setminus \mathbb{I}_v$ is a child of exactly one edge in $\mathbb{I}_v^E$. This is possible because of Lemma 3.2 (ii).

Consider an edge f in the optimal perfect matching $\mathbb{M}$, and suppose it is cut (appears in $\mathbb{I}_v^E$) at some internal node $v \in \mathbb{V}_1$ and is charged $c\lambda_v$. Let $S_1(f)$ denote its children and for $2 \leq i$ define $S_i(f)$ to be the union of the children of the edges in $S_{i-1}(f)$. Note that $|S_i(f)| \leq 3^i$. Furthermore, each edge in $S_i(f)$ is charged at most $c\lambda_v/8^i$, because the diameter of the bounding square falls by at least 8 with each level of the subdivision. Thus the total charge accumulated by the "descendants" of f is $c\lambda_v \sum_{0 \leq i} 3^i/8^i$ which is at most $c'\lambda_v$ for some constant $c' > 0$. Thus the charge to the descendents of an edge f in $\mathbb{M}$ is proportional to the charge to the edge itself. What is the expected charge to the edge f? This is at most the number of levels of the subdivision times the expected charge to it at an internal node v of the subdivision given that $f \in \mathbb{I}_v$. The number of levels in the subdivision is $O(1/\varepsilon)$. Given that $f \in \mathbb{I}_v$, the expected charge to f at v is the probability f is in $\mathbb{I}_v^E$ times $c\lambda_v$. It is easy to see that this probability is at most $2||f||/\lambda_v$. We conclude that the expected charge to f at v is $O(||f||)$, the expected total charge to f is $O(||f||/\varepsilon)$, and the expected total charge applied to all the edges in $\mathbb{I}_v^E$ for each $v \in \mathbb{V}_1$ is $O(1/\varepsilon) * \mu(\mathbb{M})$. We conclude that the expected value of $\mu(M^l) + \mu(M^d)$ is $O(1/\varepsilon) * \mu(\mathbb{M})$.

Increase in cost at the leaves. As we have already remarked, the cost of the matching computed by our algorithm would be bounded by $\mu(M^l \cup M^d)$ if the algorithm computes an optimal matching for the points associated with each leaf of the subdivision. The algorithm in fact does this at any leaf that is handled by Step 1 of $Match$ or Step 2 of $SubMatch$. The only place where the algorithm computes a sub-optimal matching of the points associated with a leaf of the subdivision is in Step 1 of $SubMatch$. In such a situation, each edge of the computed matching has length at most $\sqrt{2}\alpha/n^2$. Thus the cost of the matching computed by our algorithm is at most

$$\begin{aligned} \mu(M^d \cup M^l) + n * \sqrt{2}\alpha/n^2 &\leq \mu(M^d \cup M^l) + \sqrt{2}\mu(\mathbb{M})/n \\ &\leq \mu(M^l \cup M^d) + \mu(\mathbb{M}). \end{aligned}$$

Since the expected value of $\mu(M^d \cup M^l)$ is $O(1/\varepsilon) * \mu(\mathbb{M})$, we have established the following result.

THEOREM 3.3. *Let R be a given set of n red points and B a given set of n blue points in $\mathbb{R}^2$, and $\varepsilon > 0$ be a parameter. We can compute in $O(n^{1+\varepsilon})$ expected time a perfect matching of $B \cup R$ whose expected cost is at most $O(1/\varepsilon)$ times the optimal.*

4. The Improved Algorithm

In this section we present our improved algorithm that, given a point set $P = R \cup B$ of $2n$ points and a parameter $0 < \varepsilon < 1$, runs in $O(n^{1+\varepsilon})$ expected time and returns a matching whose expected cost is at most $O(\log \frac{1}{\varepsilon})$ times the optimal. Note that if we ignore step 2 of subroutine $SubMatch$, the algorithm of Section 3 runs in $O(n \log n + n/\delta)$ expected time, has $1/\delta$ levels, and reduces the problem to subproblems of size at most $n^{6\delta}$. The idea of the modification is to set $\delta = 1/12$ instead of $\delta = \varepsilon/12$. Then the algorithm runs in $O(n \log n)$ expected time, has a constant number of levels, and reduces the problem to subproblems of size at most $\sqrt{n}$. We then apply the same algorithm on the subproblems till we get subproblems of size at most $n^{1/4}$. We continue in this fashion till we are left with subproblems of size at most $n^{\varepsilon/2}$, which we then solve using the Hungarian algorithm. The number of levels is now $O(\log 1/\varepsilon)$, and the analysis goes through giving an approximation of $O(\log 1/\varepsilon)$ times the optimal. The running time is $O(n \log n \log 1/\varepsilon + n^{1+\varepsilon})$.

The specific modification needed to our formal subroutines is as follows: We replace Step 2 of subroutine $SubMatch$ by the following steps:

2a If $|S|/2 \leq n^{\varepsilon/2}$ we compute an optimal matching of S using the cubic algorithm and return this matching.

2b if $|S|/2 \leq m^{1/2}$, return the matching of S computed by $Match(S, D)$.

Running time analysis. As before, the expected running time of the subroutines $Match$ and $SubMatch$ is $O(|S| \log |S|)$ if we ignore Step 2a of $SubMatch$. Using an argument very similar to the previous algorithm, the overall contribution of Step 2a to the running time is $O(n^{1+\varepsilon})$. Furthermore, by construction, there is an integer constant $k \geq 1$ such that if the recursion depth is at least $k * i$ for some integer $i \geq 0$, the size of the associated point set is at most $2n^{1/2^i}$. Since the size of the associated point set is at least $2n^{\varepsilon/2}$ when a recursive call is made, we conclude that the recursion depth is $O(\log 1/\varepsilon)$. (Similar remarks apply to the depth of the subdivision produced by the new algorithm.) Thus the overall expected running time is $O(n \log n \log 1/\varepsilon + n^{1+\varepsilon})$, which is $O(n^{1+\varepsilon})$.

Quality of the matching produced. The analysis of the expected value of $\mu(M^l)+\mu(M^d)$ proceeds in a manner identical to the previous algorithm. Since the depth of the subdivision is $O(\log 1/\varepsilon)$ now, the expected value of $\mu(M^l)+\mu(M^d)$ is $O(\log 1/\varepsilon)\mu(\mathbb{M})$. A little more care is needed to bound the increase in cost at the leaves of the subdivision. Note that the algorithm computes a suboptimal matching for the points associated with a leaf w of the subdivision only using Step 1 of $SubMatch$. Let us say that an internal node v of the subdivision is *special* if the algorithm computes a crude approximation to the optimal matching of the points S_v associated with v using Step 2 of $Match$. Note that because of Step 1 of $Match$, $|S_v| \geq 2d$ for such a node v, where d is a large enough integer constant. Let j be the smallest integer such that $d^{2^j} \geq n$, and for $1 \leq i \leq j$, let N_i be the set of all special nodes v such that $2d^{2^{i-1}} \leq |S_v| < 2d^{2^i}$. The algorithm ensures that if a point is associated with two special nodes v and v' and $|S_v| < |S_{v'}|$, then $|S_v| \leq \sqrt{|S_{v'}|}$. It follows that no point is associated with more than one node from N_i. Let M_i be the matching obtained by taking the union of the *optimal* matching of S_v for each $v \in \mathrm{N}_i$. Since the matching $M^l \cup M^d$ when restricted to S_v yields a matching of S_v, we conclude that $\mu(M_i) \leq \mu(M^l \cup M^d)$.

Consider some leaf w of the subdivision where the algorithm computes a matching for S_w using Step 1 of $SubMatch$. Corresponding to w, there is a special node v such that $S_w \subseteq S_v$, the length of each edge of the matching of S_w computed by our algorithm is at most $\sqrt{2}\alpha/(|S_v|/2)^2$, where $\alpha \leq \mu(S_v)$. We "charge" the cost of such an edge to v.

A special node v can be charged by only $|S_v|/2$ edges, so the total charge to v is at most $\sqrt{2}\mu(S_v)/(|S_v|/2)$. It follows that for any $1 \leq i \leq j$, the total charge to all the nodes in N_i is at most $\sqrt{2}\mu(M_i)/d^{2^{i-1}} \leq \sqrt{2}\mu(M^l \cup M^d)/d^{2^{i-1}}$. Thus the total charge to all the special nodes is $\sqrt{2}\mu(M^l \cup M^d) \sum_{i=1}^{j} 1/d^{2^{i-1}} = O(\mu(M^l \cup M^d))$. Since the cost of the matching M output by our algorithm is at most $\mu(M^l \cup M^d)$ plus the total charge to all the special nodes, we conclude that $\mu(M) = O(\mu(M^l \cup M^d))$. Thus the expected value of $\mu(M)$ is $O(\log 1/\varepsilon)\mu(\mathbb{M})$. Thus we have:

THEOREM 4.1. *Let R be a given set of n red points and B a given set of n blue points in $\mathbb{R}^2$, and $\varepsilon > 0$ be a parameter. We can compute in $O(n^{1+\varepsilon})$ expected time a perfect matching of $B \cup R$ whose expected cost is at most $O(\log 1/\varepsilon)$ times the optimal.*

5. Conclusions

To obtain a constant-factor approximation algorithm that runs in say $O(n \log n)$ time, we may have to allow a richer interaction than we currently do between the children of each internal node of the subdivision. It is a very interesting open question to figure out how a sufficiently rich interaction can be accomplished in the allowed time.

References

[1] P. K. Agarwal, A. Efrat, and M. Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 39–50, 1995.

[2] P. K. Agarwal and K. R. Varadarajan. Approximation algorithms for bipartite and non-bipartite matching in the plane. *Proc. SODA 99.*

[3] S. Arora. Nearly linear time approximation schemes for Euclidean TSP and other geometric problems. In *Proc. 38th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 554–563, 1997.

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

[5] Paul B. Callahan and S. Rao Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 291-300, 1993.

[6] J. Edmonds. Maximum matching and a polyhedron with (0,1) vertices. *J. Res. National Bureau of Standards*, 69B:125–130, 1965.

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

[8] Z. Galil, S. Micali, and H. N. Gabow. Priority queues with variable priority and an $o(ev \log v)$ algorithm for finding a maximal weighted matching in general graphs. In *Proc. 22nd Annual IEEE Symposium on Foundations of Computer Science*, pages 255–261, 1982.

[9] H. Kuhn. The Hungarian method for the assignment problem. *Naval Res. Logist. Q.*, 2:83–97, 1955.

[10] E. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart & Winston, New York, 1976.

[11] S. B. Rao and W. D. Smith. Improved approximation schemes for traveling salesman tours. In *Proc. 30th Annu. ACM Sympos. Theory Comput.*, 1998.

[12] P. M. Vaidya. Approximate minimum weight matching on points in k-dimensional space. *Algorithmica*, 4:569–583, 1989.

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

[14] Kasturi R. Varadarajan. A divide-and-conquer algorithm for min-cost perfect matching in the plane. *In Proc. FOCS 98.*