Geometry Helps in Bottleneck Matching and Related Problems*

Alon Efrat† Alon Itai‡ Matthew J. Katz§

This paper is accepted for publication in ALGORITHMICA

Abstract

Let A and B be two sets of n objects in $\mathbb{R}^d$, and let *Match* be a (one-to-one) matching between A and B. Let $\min(\mathit{Match})$, $\max(\mathit{Match})$, and $\Sigma(\mathit{Match})$ denote the length of the shortest edge, the length of the longest edge, and the sum of the lengths of the edges of *Match* respectively. *Bottleneck matching*—a matching that minimizes $\max(\mathit{Match})$—is suggested as a convenient way for measuring the resemblance between A and B. Several algorithms for computing, as well as approximating, this resemblance are proposed. The running time of all the algorithms involving planar objects is roughly $O(n^{1.5})$. For instance, if the objects are points in the plane, the running time of the exact algorithm is $O(n^{1.5}\log n)$. A semi-dynamic data-structure for answering containment problems for a set of congruent disks in the plane is developed. This data structure may be of independent interest.

Next, the problem of finding a translation of B that maximizes the resemblance to A under the bottleneck matching criterion is considered. When A and B are point-sets in the plane, an $O(n^5\log n)$ time algorithm for determining whether for some translated copy the resemblance gets below a given ρ is presented, thus improving the previous result of Alt, Mehlhorn, Wagener and Welzl by a factor of almost n. This result is used to compute the smallest such ρ in time $O(n^5\log^2 n)$, and an efficient approximation scheme for this problem is also given.

The *uniform matching* problem (also called the *balanced assignment* problem, or the *fair matching* problem) is to find Match^*_U, a matching that minimizes $\max(\mathit{Match}) - \min(\mathit{Match})$. A *minimum deviation matching* Match^*_D is a matching that minimizes $(1/n)\Sigma(\mathit{Match}) - \min(\mathit{Match})$. Algorithms for computing Match^*_U and Match^*_D in roughly $O(n^{10/3})$ time are presented. These algorithms are more efficient than the previous $O(n^4)$-time algorithms of Martello, Pulleyblank, Toth and de Werra, and of Gupta and Punnen, who studied these problems for general bipartite graphs.

---

*Preliminary versions of parts of this paper appeared as [20] and [21].

†School of Mathematical Sciences, Tel-Aviv University, Tel-Aviv 69978, Israel. *alone@cs.tau.ac.il*

‡Department of Computer Science, Technion – Israel Institute of Technology, Haifa 32000, Israel. *itai@cs.technion.ac.il* Supported by the fund for the promotion of research at the Technion.

§Department of Mathematics and Computer Science, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel. *matya@cs.bgu.ac.il*

1 Introduction

In the field of pattern recognition it is often required to measure the *resemblance* between two sets A and B of objects in d-dimensional space. This problem often arises when an input image is given, and we seek, among model images stored in some library, the one that is most similar to the given image.

Many methods have been suggested for quantifying this similarity. Perhaps the most common of which is the *Hausdorff distance*, defined as the maximum distance between an object in one set and its closest neighbor in the other set. Many algorithms and applications have been suggested for computing and applying the Hausdorff distance (e.g. [12, 13, 14, 33, 34]). However, measuring the resemblance by the Hausdorff distance suffers from the following problem which is sometimes a fundamental drawback: the mapping defined by associating each object in A to its closest neighbor in B is not necessarily a bijection (one-to-one).

Figure 1: A set A of points represented as solid disks, and a set B of points, represented as empty disks.

Quite often it is required that each object in an image be matched by one and only one object in the other image. In such cases the Hausdorff distance is meaningless, see Figure 1.

In this paper we propose a different measure of similarity: We assume that both images A and B have the same number of objects, a *perfect bipartite matching* is a bijection $Match$ from A to B. Let $\max(Match)$ denote the maximal distance between any matched pair of objects. We seek a matching $Match$ that minimizes $\max(Match)$. We refer to this measure as the *bottleneck matching* criterion, and define the distance between the two images as the longest distance between any matched pair. Let $Match(A, B)$ denote this distance.

The disadvantage of bottleneck matching, as well as any distance that relies on one-to-one matching, is that it is probably more complicated to compute than the Hausdorff distance, and the algorithms tend to be less efficient. A partial explanation is that the known algorithms attack the problem as a purely graph-theoretic one without taking advantage of its geometric nature.

Furthermore, the problem of minimizing the resemblance under some rigid motion or other transformation of one image relative to the other, has been investigated mainly from a practical point of view, and the best known algorithms are either computationally inefficient (see Alt et al. [5]), or significantly restrict the inputs (see Arkin et al. [6]).

For the case where the sets A and B are points in the plane, Vaidya [46] explored the geometric structure of the problem to obtain an algorithm for finding a matching between A and B, for which the sum of distances between the matched points is minimal (among all perfect matchings between A and B). (This criterion is different from our bottleneck criterion.) He obtained an $O(n^{2.5} \log n)$-time algorithm for the Euclidean distance, and an

Dim	A	B	Norm	Oracle	Thm.	*Match*	Thm.
$\mathbb{R}^2$	points	points	$L_p\ \forall p$	$O(n^{1.5}\log n)$	5.4	$O(n^{1.5}\log n)$	5.10
			additive weights	$O(n^{1.5+\varepsilon})$	6.7	$O(n^{1.5+\varepsilon})$	6.8
		segments	$L_p\ \forall p$	$O(n^{1.5+\varepsilon})$	6.7	$O(n^{1.5+\varepsilon})$	6.8
$\mathbb{R}^3$	points	points	L_2	$O(n^{11/6+\varepsilon})$	6.2	$O(n^{11/6+\varepsilon})$	6.3
$\mathbb{R}^d$	points	points	L_∞	$O(n^{1.5}\log^{d-1} n)$	6.4	$O(n^{1.5}\log^{d} n)$	6.5

Table 1: Computing *Match*(A, B) in different settings

$O(n^2 \log^3 n)$-time algorithm for the L_∞ distance. The solution of the Euclidean case has recently been improved by Agarwal et al. [2] to $O(n^{2+\varepsilon})$[1]. However, the resulting algorithms remain relatively complicated. See also Buss and Yianilos [11] for fast algorithms for other types of graphs related to geometric configurations.

For computing *Match*(A, B) we introduce in Section 3 an oracle that determines, for a parameter r, whether *Match*$(A, B) \leq r$. The exact running times depend on the norm and the dimension.

The oracle is then used to find *Match*(A, B); that is, the minimal r for which *Match*$(A, B) \leq r$. Clearly, *Match*(A, B) must equal a distance between an object of A and an object in B. Thus our search space is confined to n^2 such distances. In Section 4 we show how to conduct the search efficiently. In some cases (Sections 5.3, 6.1) the time required for finding the matching is the same as the oracle time.

Sections 5-6 discuss the implementation of the data structures needed for the oracle and for finding the matching itself, for different choices of the dimension of the space, the sets A and B and the underlying norm. These results are listed in Table 1.

When $A, B \subseteq \mathbb{R}^2$ are point-sets, and the underlying norm is L_2 (the *planar Euclidean point-sets case*) our algorithm runs in time $O(n^{1.5} \log n)$. For this case, we developed (Section 5.1) a semi-dynamic linear-size data structure for a set S of equal-size disks in the plane, so that finding a disk containing a query point, and deleting a disk from S, can both be performed in time $O(\log n)$. We believe that this data structure is of interest of its own.

In Section 5.2 we show how to conduct the search efficiently, so the running time is $O(\textit{Oracle-Time} \cdot \log n) = O(n^{1.5} \log^2 n)$. Moreover, for this case we can shortcut the generic algorithm and find the matching in the same time as the oracle (Section 5.3), i.e., in time $O(n^{1.5} \log n)$ (Theorem 5.10).

Additional settings are discussed in Section 6. Assume first that A and B are point-sets in $\mathbb{R}^d$. For $d = 3$ and the L_2 norm (the *3-dimensional Euclidean point-sets case*), we propose a $O(n^{11/6} + \varepsilon)$ time algorithm (Theorem 6.3). When the norm is L_∞ (the L_∞ *point-sets d-space case*), the running time is $O(n^{1.5} \log^d n)$ (Theorem 6.5).

When A is a set of n points in the plane, B is a set of n segments in the plane, and the norm is an arbitrary L_p, or when A and B are sets of points in the plane and the

[1] Throughout the paper, ε stands for a positive constant which can be chosen arbitrarily small with an appropriate choice of other constants of the algorithms.

distance is additively weighted (i.e., $dist_w(a, b) = ||a - b||_p + w(b)$ for some non-negative weight function w), the running time of the algorithm for computing $Match(A, B)$ is slightly worse— $O(n^{1.5+\varepsilon})$, for any $\varepsilon > 0$ (Theorem 6.8).

Section 7 presents an approximation scheme that computes an ε-approximation for $Match(A, B)$, in any dimension in time $O(n^{1.5} \log n)$, where A and B are point-sets and the constant of proportionality depends on the dimension and on ε. We believe that this scheme is relatively easy to implement, with reasonably small constant of proportionality, and therefore would do reasonably well in practice.

We also show in Section 8 an application of our technique for the *translation problem*: Let A and B be two n-point sets in the plane, and ρ a fixed number. The problem is to find a translation B' of B such that $Match(A, B')$ is at most ρ, or determine that no such translation exists. Alt et al. [5] gave an $O(n^6)$-time algorithm for this problem. We improve this bound to $O(n^5 \log n)$, and show how to find in $O(n^5 \log^2 n)$ time a translation B^* of B that minimizes $Match(A, B')$, over all translations B'. We also present a scheme to find a translation that approximates $Match(A, B^*)$.

In Section 9 we discuss two problems strongly related to the matching problem. The first is *Partial Matching* in which we are given A, B (not necessarily of the same cardinality) and a parameter $1 \le p \le \min\{|A|, |B|\}$, and we seek a matching of cardinality p for which its longest edge is as short as possible. The second problem is *Longest Perfect Matching* in which we are given A, B, and seek $\overline{Match}(A, B)$, the largest r for which a perfect matching exists, such that the length of all its edges is r or more. Surprisingly, for points in $\mathbb{R}^3$, this problem is easier to tackle than the problem of finding $Match(A, B)$.

Finally, we consider the problem of finding a matching $Match$ between A and B which is as balanced as possible. We consider *(most) uniform matching* M_U^* which minimizes $\max(Match) - \min(Match)$, where $\min(Match)$ is the the minimum distance between any matched pair. Martello et al. [40] considered this problem (or a *balanced* assignment, as they called it) for general bipartite graphs, and presented an $O(n^4)$-time solution. In Section 9.4 we present an $O(n^{10/3} \log n)$-time solution for this problem in the geometric setting. Our solution uses both the technique for computing a bottleneck matching, and a technique for batched range searching, where the ranges are congruent annuli (see Katz and Sharir [36]).

Another criterion for balancing matchings is to minimize $(1/n)\Sigma(Match) - \min(Match)$, where $\Sigma(Match)$ is the sum of lengths of the edges of $Match$. A best matching under this criterion is called a *minimum-deviation matching* M_D^* and is discussed in [19] and in [21].

2 Matching in General Bipartite Graphs

Let us first discuss the connection between our problem and standard graph-matching theory. A *graph-matching* of a bipartite graph $G = (X \cup Y, E)$ is a set of edges $M \subseteq E(G)$ such that no vertex of G is incident to more than one edge of M. A graph-matching M is *perfect* if every vertex of G is incident to an edge of M. The problem of finding a perfect matching in a bipartite (or arbitrary) graph has been well studied. See for example [38, 39] for textbooks on this subject. The best known algorithm for finding a perfect matching in a bipartite graph runs in time $O(m\sqrt{n})$ (where n is the number of vertices and m is the number of edges) and is due to Hopcroft and Karp [32]. When a weight is associated with each edge, and we seek a perfect matching for which the sum of weights of its edges is minimal, the best known algorithm runs in time $O(n^3)$, using the so called *Hungarian method*, and is due to Kuhn [37].

Let us define our problem in graph-theoretical terms: The images A and B are each a set of n vertices of a complete bipartite graph $G = (A \cup B, E)$. The weight of the edge $(a, b) \in E$ is $dist(a, b)$—the distance between $a \in A$ and $b \in B$. Let $\max(M)$ denote as above, the weight of the heaviest edge of a graph-matching M. The bottleneck matching is a perfect graph-matching $M \subseteq E$ that minimizes $\max(M)$.

Let $G[r]$ be the bipartite graph whose vertex set is $A \cup B$, and whose edges consist of all pairs (a, b) $a \in A$, $b \in B$ for which $dist(a, b) \leq r$. Note that $Match(A, B) \leq r$ if and only if there exists a perfect graph-matching in $G[r]$. We therefore focus on finding a maximum graph-matching in $G[r]$—a graph-matching of largest cardinality.

Given a graph-matching M of a bipartite graph $G = (A \cup B, E)$, the vertices incident to edges of M are called *matched* and the remaining vertices are *exposed*. The path $\pi = (v_1, ..., v_{2t})$ is an *alternating path* if v_1 is an exposed vertex of A, $(v_{2i}, v_{2i+1}) \in M$ $(i = 1, ..., t-1)$ and $(v_{2i-1}, v_{2i}) \in E \setminus M$ $(i = 2, ..., t)$. Note that the odd vertices of π belong to A, and the even ones to B. This path is called *an augmenting path* if v_{2t} is an exposed vertex. If π is an augmenting path then $M' = M \oplus \pi = (M \setminus \pi) \cup (\pi \setminus M)$ is a graph-matching too and $|M'| = 1 + |M|$.

A theorem of Berge [10] states that a matching is maximum if and only if there are no augmenting paths. Thus one may start with the empty matching and augment it by augmenting paths found in a greedy fashion.

Edmonds and Karp [18] showed how to compute augmenting paths by order of increasing length. Instead of finding the augmenting paths one by one, Hopcroft and Karp [32], and also Karzanov [35] who followed the techniques of Dinitz [16], find all shortest augmenting paths together. We follow Dinitz's terminology (see also [45]).

To find all shortest augmenting paths, we conduct a breadth-first-search to get layers $L_1, \ldots, L_{2t}$. The first layer, L_1, consists of all exposed vertices of A; L_{2i} contains all vertices of B not appearing in $\bigcup_{j<2i} L_j$ and connected (in G) to some vertex of L_{2i-1}. If L_{2i} contains exposed vertices, then it is the last layer. Otherwise, we define L_{2i+1} to contain all vertices connected (in the matching M) to vertices in L_{2i}. Note that the odd layers contain only vertices of A and the even layers only vertices of B.

The layered graph $\mathcal{L}$ consists of the vertex set $\bigcup_{i=1}^{2t} L_i$, and edges of M that connect vertices of L_{2j} to vertices of L_{2j+1}, and edges of G that connect vertices of L_{2j-1} to vertices of L_{2j}.

Dinitz showed how to compute a maximal set of edge-disjoint augmenting paths by conducting a depth-first search of the layered graph. His algorithm requires $O(|E|)$ time to construct the layered graph and to find the augmenting paths. For sufficiently large values of r, $G[r]$ contains $\Theta(n^2)$ edges, hence his algorithm applied to our setting requires $O(n^2)$ time per layered graph. We take advantage of the geometric features of $G[r]$ to improve the efficiency of Dinitz's algorithm. We will represent the edges of $\mathcal{L}$ implicitly, and thus our construction will enable us to find the augmenting paths in $\mathcal{L}$ in almost $O(n)$ time.

3 Maximal matching in $G[r]$

In this section we describe an *oracle* to decide whether a given r is less than, equal to or greater than $r^* = \mathit{Match}(A, B)$. The oracle searches for a perfect matching in $G[r]$, using Dinitz's algorithm and taking advantage of the geometric setting.

3.1 Constructing $\mathcal{L}$ implicitly

Our goal is to find the set of vertices of each layer L_i; however, we will not explicitly construct all the edges of $\mathcal{L}$. Instead, we shall use an abstract data-structure $\mathcal{D}_r(S)$ for a set of objects $S \subseteq B$. The data-structure supports the following operations:

- $\mathbf{neighbor}_r(\mathcal{D}_r(S), q)$: For a query point q, return an element $s \in S$ whose distance from q is at most r. If no such s exists, then $\mathbf{neighbor}_r(\mathcal{D}_r(S), q) = \emptyset$.
- $\mathbf{delete}_r(\mathcal{D}_r(S), s)$: Delete the object s from S.

The implementation of $\mathcal{D}_r(\cdot)$ depends on the dimension, the objects of S and the underlying norm. Various implementations will be described in Sections 5–6.3. Let $T(|S|)$ denote an upper bound on the time of performing one of these two operations on $\mathcal{D}_r(S)$. We disregard the time needed to construct the data structure, since in all relevant cases it is bounded by $O(n \cdot T(n))$, and does not influence the overall complexity.

Let us turn now to the algorithm for generating $\mathcal{L}$. Initially, set $\mathcal{D} \leftarrow \mathcal{D}_r(B)$. In the course of the algorithm, some vertices of B will be deleted. Using this data structure, the layered graph is constructed by the following procedure:

```
procedure ConstructLayerGraph(G, M)
L_1 ← exposed vertices of A;
i ← 1; D ← D_r(B);
Repeat forever
    L_2i ← ∅;
    For each a ∈ L_2i−1 Do
        /* Find all b's which are neighbors of some a in G[r] */
        While neighbor_r(D, a) ≠ ∅
            b ← neighbor_r(D, a);
            Add b to L_2i;
            delete_r(D, b);    /* in order to prevent re-finding b */
        End
    End
    If L_2i is empty
        Then no augmenting path exists. Stop.
    Else If L_2i contains exposed vertices,
        Then the construction of L is complete;
            Output L;
    Else L_2i+1 ←   all vertices of A adjacent to L_2i via edges of M.
    i ← i + 1;
End
```

Each matched vertex of A is reached in $O(1)$ time from its pair in M. Also, each vertex of B is found at most once by a query of $\mathbf{neighbor}_r(\mathcal{D}, \cdot)$ and deleted from $\mathcal{D}$ at most once. Thus the construction time of $\mathcal{L}$ is $O(n \cdot T(n))$.

3.2 Finding augmenting paths in $\mathcal{L}$

We now show that the augmenting paths in any maximal set of edge-disjoint augmenting paths are vertex disjoint.

Lemma 3.1 *Let M be a graph-matching of a bipartite graph $G = (A \cup B, E)$, let Π be a set of edge-disjoint augmenting paths, and let v be an intermediate vertex of some path of Π. Then v cannot participate in any other augmenting path of Π.*

Proof: Since v is neither the first nor the last vertex of the augmenting path, v is not exposed so it must be incident to exactly one edge $(v, v') \in M$. Suppose $v \in L_{2j}$. By our construction, (v, v') connects L_{2j} and L_{2j+1}. Hence, every augmenting path that contains v must also contain the edge (v, v'). Since the paths of Π are edge disjoint, v cannot belong to any other path of Π. A similar argument holds when v belongs to an odd layer. $\square$

Next we look for augmenting paths from the exposed vertices of L_1 to exposed vertices of L_{2t} (the last layer). First we construct $\mathcal{D}_{2i} \equiv \mathcal{D}_r(L_{2i})$ for each of the even layers $L_{2i} \subseteq B$. Then we conduct a depth-first search: We start from an exposed vertex in L_1 and construct an alternating path. To advance from a vertex $a \in L_{2i-1}$, we perform $\mathbf{neighbor}_r(\mathcal{D}_{2i}, a)$.

If it returns a vertex $b \in L_{2i}$ then we add (a, b) to the current path and advance to b. Otherwise, it returns $\emptyset$ indicating that no neighbors of a remain in L_{2i}. Thus a does not lead to an exposed vertex of L_{2t} and we should backtrack.

To advance from $b \in L_{2i}$ $(i < t)$, let $(b, a^+) \in M$. We add (b, a^+) to the path and advance to a^+ (b is not exposed since in $\cap L$ all exposed vertices of B belong to L_{2t}). If $b \in L_{2t}$ is an exposed vertex then we have found an augmenting path. We increase M and delete all its intermediate vertices from the appropriate L_{2i}'s. (This is justified by Lemma 3.1.)

To backtrack from $a \in L_{2i-1}$ $(i \geq 2)$, let $(b^-, a) \in M$ and let a^- be the vertex preceding b^- on the path. We remove a and b^- from the path and continue from a^-. If $a \in L_1$ we simply delete it from L_1.

The search for augmenting paths (and the phase) terminates when there remain no more exposed vertices in L_1.

If all the vertices are matched, then we conclude that $r^* \leq r$, otherwise, we conclude that $r^* > r$. If during the construction of $\mathcal{L}$ one doesn't reach any exposed vertex of B, then $G[r]$ contains no perfect matching. We therefore halt and conclude that $r^* > r$.

Note that the time spent on finding all alternating paths in a single layered graph is again $O(n \cdot T(n))$. By a theorem of Hopcroft and Karp [32], Dinitz's matching algorithm requires $O(\sqrt{n})$ phases. Hence we have the following theorem:

Theorem 3.2 *Let A and B be two sets of n objects and $r > 0$. Then the oracle that determines whether $r \leq Match(A, B)$ requires time $O(n^{1.5} \cdot T(n))$, where $T(|S|)$ is a (monotonically nondecreasing) upper bound on the time required to perform an operation on $\mathcal{D}_r(S)$.*

4 Finding the Optimum Matching

The oracle is now used to find $Match(A, B)$; that is, the minimal r for which $Match(A, B) \leq r$. Clearly, $Match(A, B)$ must equal a distance between an object of A and an object in B. Thus our search space is confined to n^2 such distances.

Rather than calling the oracle for all these distances, we wish to conduct a binary search. Thus, naively, we would have to first calculate all n^2 distances, sort them, and then conduct the binary search, calling the oracle at most $2 \log n$ times. However, if the oracle requires time $o(n^2)$, the time to find the distances and sort them will dominate the total running time.

In order to minimize the number of times the oracle is called, we need to efficiently solve the following variant of the the k'th distance selection problem. For $a_i \in A, b_j \in B$ let $dist(a_i, b_j)$ denote the distance from a_i to b_j. The *k'th bi-chromatic distance selection problem* is to find $dist^{(k)}$, the k'th largest value in the multiset $\{dist(a_i, b_j) | 1 \leq i, j \leq n\}$, where k is a given parameter.

If we can find $dist^{(i)}$ in time *Select-Time* $= o(n^2)$, then since the time required by each iteration requires *Select-Time* + *Oracle-Time*, the time to find a minimum matching will become $O((\textit{Select-Time} + \textit{Oracle-Time}) \log n)$. If *Select-Time* $= O(\textit{Oracle-Time})$ then *Select-Time* can be ignored.

5 The Euclidean Planar Case

We start with our most involved example—points in the Euclidean plane—for which we have the strongest results. Let A and B be sets of n points in $\mathbb{R}^2$ and let the underlying norm be the Euclidean norm—L_2. The same data structure may also be used for the norms L_p, for any $1 \le p \le \infty$.

5.1 The oracle

To get the oracle that checks whether $r < \mathit{Match}(A, B)$, we have to show how to implement $\mathbf{neighbor}_r(\mathcal{D}_r(S), q)$ for a query point $q \in \mathbb{R}^2$, and $S \subseteq B$, and $\mathbf{delete}_r(\mathcal{D}_r(S), q)$—delete the point q from S.

To simplify the notation, we scale the coordinates so that $r = 1$. Let $S = \{d_1, \ldots, d_n\}$ be a set of unit disks, $q \in \mathbb{R}^2$, and let the operation $\mathit{member}(q, S)$ return a disk $d_i \in S$ containing q, and $\emptyset$ if no such disk exists. In order to implement our algorithm, we need a data structure that supports efficiently membership queries and deletion of disks.

To that end, we divide the plane using the axis-parallel grid Γ consisting of orthogonal cells of edge length 1/2 that passes through the origin. Since the disks have unit radius, each disk intersects $O(1)$ cells, hence, only $O(n)$ cells have a non-empty intersection with disks of S. We maintain these cells in a balanced search tree (ordered lexicographically). For each such cell Q, we maintain a list of disks whose center lies in Q, and a data structure $\mathcal{D}_b$, which maintains the upper envelope of S_b^Q—the disks set of disks that intersect Q, and whose centers lie below the line containing the lower boundary of Q. (The *upper envelope* of S_b^Q consists of all points $p \in Q \cap \bigcup_{D \in S_b^Q} D$ such that no point of this union lies above p.) Similar data structures $\mathcal{D}_l, \mathcal{D}_r$ and $\mathcal{D}_a$ are maintained for S_l^Q, S_r^Q and S_a^Q, the set of disks intersecting Q whose centers lie (respectively) to the left of, to the right of and above the lines containing the left, right and upper boundaries of Q. The space needed for $\mathcal{D}_b$ will be shown to be $O(|S_b^Q|)$, and similarity for the other data structures. Since each disk intersects $O(1)$ grid cells, the space requirement for all these data structures is $O(n)$.

To answer the query $\mathit{member}(q, S)$, we consider the cell Q of Γ containing q in its interior (we ignore the degenerate and easy situation that q is on a boundary of a cell). If the center of a disk d_i lies inside Q then $q \in d_i$, and can be output as $d_i = \mathit{member}(q, S)$. Otherwise, we use $\mathcal{D}_b$ to find if any disk of S_b^Q contains q, which happens if and only if q lies below the upper envelope of S_b^Q. We repeat this process (if needed) for $\mathcal{D}_l, \mathcal{D}_r$ and $\mathcal{D}_a$.

Let us describe $\mathcal{D}_b$. The data structures $\mathcal{D}_l, \mathcal{D}_r$ and $\mathcal{D}_a$ are similar. Similar data structure was also used by Sharir [44]. $\mathcal{D}_b$ is similar to the segment tree (Overmars and van Leeuwen [42]) and the one used by Hershberger and Suri [30]. Order the disks of S_b^Q from left to right by their centers, and construct a complete binary tree $\mathcal{T}$ whose leaves are these disks. With each node $v \in \mathcal{T}$ we associate the set $S(v)$ of disks corresponding to the leaves of the subtree rooted at v. Let $\mathit{UE}(v)$ denote the upper envelope of $S(v)$. As easily seen, not all the disks of $S(v)$ must participate in $\mathit{UE}(v)$, but those that do, appear along $\mathit{UE}(v)$ (when scanned, from left to right) in the same order as the order of their centers, from left to right.

Lemma 5.1 *Let v be a node of $\mathcal{T}$, and let left(v) and right(v) denote its left and right* *children. Then UE(left(v)) and UE(right(v)) have at most one intersecting point.*

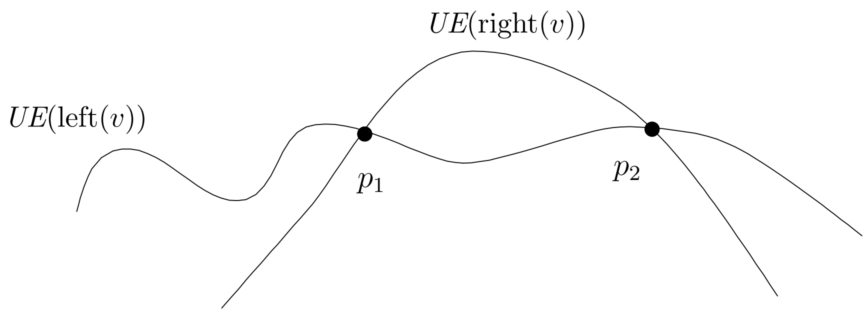

Figure 2: The proof of Lemma 5.1

Proof: Refer to Figure 2. Assume that two such intersection points exist, say p_1 and p_2, where p_1 is to the left of p_2, and no third intersection point exists between them. Assume without lost of generality that in the open infinite vertical strip whose boundaries pass through p_1, and p_2, *UE*(left(v)) is below *UE*(right(v)). Consider d_l and d_r, the disks of S(left(v)) and S(right(v)) respectively, containing p_2 on their boundaries. As easily seen, the center of d_l is to the right of the center of d_r, which is a contradiction. □

Corollary 5.2 *Let UE(left(v)) consist of arcs of the (boundary of the) disks $\ell_1, \ldots, \ell_L$, and let UE(right(v)) consists of arcs of* $= r_1, \ldots, r_R$, *where the centers of these disks are ordered from left to right in this order. Then there exist* i, j ($0 \le i \le L$, $1 \le j \le R+1$) *such that UE(v) consists of the arcs of disks* $\ell_1, \ldots, \ell_i, r_j, \ldots, r_R$ *in this left-to-right order.*

The data structure: Let $p(v)$ be the (single) intersection point of *UE*(left(v)) and *UE*(right(v)). We call this point the *junction point* of v. Associated with v we keep $p(v)$, ℓ_i, r_j and $LIST(v)$—a doubly-linked list of the vertices of *UE*(v) (with the disks defining them) that do not belong to *UE*($parent(v)$). (If a disk does not intersect any disk to its left (right) we add the left (right) intersection of the disk with the bottom line of Q.) Observe that $LIST(v)$ represents a connected portion of *UE*(left(v)) concatenated to a connected portion of *UE*(right(v)), where $p(v)$ is a common endpoint of these two portions. We maintain pointers from v to the corresponding "middle" vertices in $LIST$(left(v)) and $LIST$(right(v)).

Construction of the data structure: The construction is performed bottom-up from the leaves of $\mathcal{T}$ up to its root. In each step, we are at a node v, and we have computed *UE*(left(v)) and *UE*(right(v)). We merge *UE*(left(v)) and *UE*(right(v)), in linear time using a standard line sweep procedure to find $p(v)$, and we store in $LIST$(left(v)) (resp. $LIST$(right(v))) the portion of *UE*(left(v)) (resp. *UE*(right(v))) which does not appear in *UE*(v). Since at each level of the tree we process $O(n)$ disks, the time required for the entire construction is $O(n \log n)$.

Membership queries: To carry out $member(q, S)$, we consider the tree as a binary search tree on the values $\mathbf{x}(p(v))$—the x-coordinate of $p(v)$'s. Let $\mathbf{x}(p_1) < \ldots < \mathbf{x}(p_{n-1})$ denote these values. Then the i'th leaf u_i corresponds to the interval $(\mathbf{x}(p_{i-1}), \mathbf{x}(p_i))$. We find a leaf u_j such that $\mathbf{x}(p_{j-1}) < \mathbf{x}(q) < \mathbf{x}(p_j)$. The query point q is covered by $\bigcup S_b^Q$ if and only if it belongs to d_j. The time complexity of this operation is $O(\log n)$.

Deletions: The difficulty with deletions is that deleting a disk d might cause disks that were occluded by d to appear in the upper-envelope $UE(v)$. The deletion of d proceeds bottom up: We first *mark* the leaf corresponding to d as being deleted, update $LIST(v) = \emptyset$, and continue to v's parent. No change takes place in the topology of the tree itself.

In a general step we are at a node v, and d appears in $S(v)$, say in $S(\text{left}(v))$. The case that $d \in S(\text{right}(v))$ is symmetric. We obtain the following information from the previous step:

- A linked list L of all disks presently in $UE(\text{left}(v))$ which were occluded by d. Let u_1 and u_2 be the left and right endpoints of L, respectively.
- Pointers q_1 and q_2 to the vertices (in the appropriate $LIST(\cdot)$ fields) u_1 and u_2. See Figure 3 for a demonstration.

Let $p'(v)$ denote $p(v)$ before the deletion of d took place.

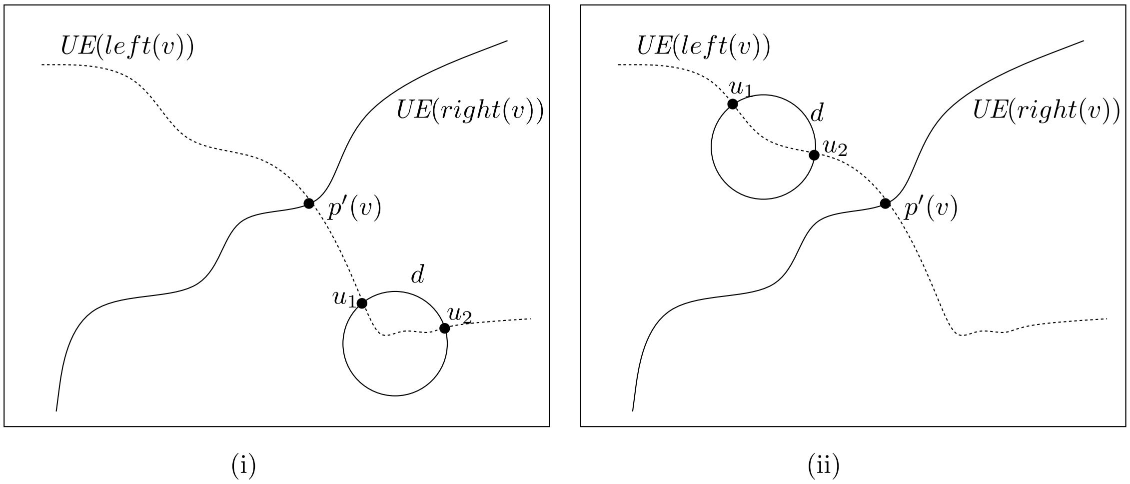

Figure 3: The two cases where d does not contain $p(v)$.

Three cases might arise:

(i) $\mathbf{x}(p'(v)) \le \mathbf{x}(u_1)$

(see Figure 3 (i)). This implies that d does not appear in $UE(v)$, that is $p'(v) = p(v)$. This only requires us to insert L into $LIST(\text{left}(v))$ at the appropriate place, which is pointed at by q_2. This case terminates the deletion process.

(ii) $\mathbf{x}(u_2) \leq \mathbf{x}(p'(v))$

(see Figure 3(ii).) We conclude again that $p(v) = p'(v)$, we do not change L, q_1, q_2 nor the fields within v, and continue to $parent(v)$.

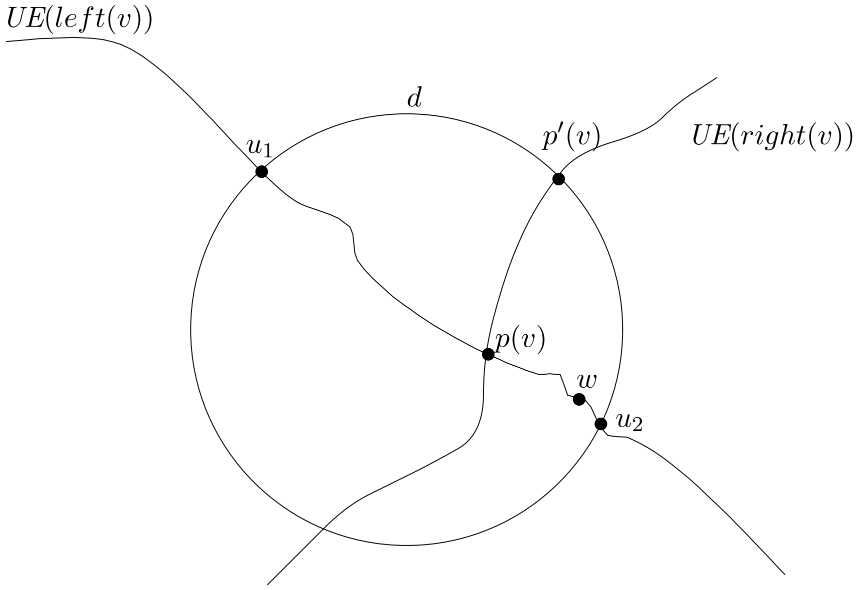

Figure 4: Exploring a new junction point $p(v)$, which was occluded by d

(iii) $\mathbf{x}(u_1) \leq \mathbf{x}(p'(v)) \leq \mathbf{x}(u_2)$

(see Figure 4). This is the most involved case. Let $p(v)p'(v)$ denote the part of $UE(\text{right}(v))$ from $p(v)$ to $p'(v)$, and let $p(v)u_2$ denote the part of $UE(\text{left}(v))$ from $p(v)$ to u_2. We can traverse L, since it is organized as a linked list. Moreover, observe that $p(v)p'(v)$ must be a connected portion of $LIST(\text{right}(v))$ (organized as a linked list), hence we can easily travel along this list as well.

Traversing along the envelopes

Out of the points $p'(v)$ and u_2, choose the point which is furthest to the right (u_2 in Figure 4). We roll from this point to the left, along the corresponding envelope of $UE(\text{left}(v))$, until we arrive at a point w with the same x-coordinate as the other point ($p'(v)$ in Figure 4). Next we travel simultaneously on both $UE(\text{left}(v))$ and $UE(\text{right}(v))$ leftwards, maintaining the points we are in on both chains vertically one below the other, until we reach $p(v)$. The time complexity of this stage is proportional to the number of disks of $p(v)u_2$ plus that of $p(v)p'(v)$.

Next we delete $p(v)p'(v)$ from $LIST(\text{right}(v))$, and insert $p(v)u_2$ into $LIST(\text{left}(v))$ just after the vertex u_2 pointed at by q_2 (and we remove the arc $p'(v)u_2$ of d from this list).

We now need to prepare the output of the procedure. The list L is the portion $u_1p(v)$ of the "old" L concatenated with the portion $p(v)p'(v)$ just discovered, (u_1 and q_1 are not changed). u_2 is set to be $p'(v)$, and we update q_2 accordingly.

Time analysis for the deletion operation: As is easily seen, at a node v we spend time $O(1+\lambda)$, where λ is the length of L plus the length of of $p(v)p'(v)$. We need an amortized argument to bound the sum of these quantities over the course of the algorithm.

From its definition, $\mathcal{T}$ is a complete binary tree with at most n leaves. Each disk corresponds to a leaf, and appears only in the ancestors of that leaf. Hence, a disk d_0 might appear in at most $O(\log n)$ $LIST(\cdot)$ fields, say $LIST(v_1), \ldots, LIST(v_m)$. However, in all but at most one of these fields, ∂d_0 must contain the corresponding $p(v_i)$. For $d_i \in S$ let $level(d_i)$ denote the distance from the root of $\mathcal{T}$ to a lowest node v for which ∂d_i contains a vertex of $LIST(v)$. We say that d_0 was *promoted* due to the deletion of another disk d, if $level(d_0)$ decreased due to this operation. Obviously the level of a disk never increases, and since each disk can be promoted at most $\log n$ times (the height of the tree), the total number of promotions in the course of the algorithm is $O(n \log n)$

Consider the disks contained in the list L and in $p(v)p'(v)$, excluding the endpoints of these lists. The travel along $p(v)p'(v)$ and $p(v)u_2$ can be charged to such promotions: Each disk of $UE(\text{right}(v))$ that we scanned (excluding the endpoints) was in $LIST(\text{right}(v))$, and will be promoted to $LIST(v)$ or to a higher node. Each disk of $UE(\text{left}(v))$ that we scanned (excluding the endpoints) was promoted from the $LIST(\cdot)$ field of some proper descendent of left(v) to $LIST(\text{left}(v))$. Hence the total time dedicated to traversals, over the entire course of the algorithm, is $O(n \log n)$.

Hence we have

Lemma 5.3 *Given a set S of n unit disks in the plane, we can construct in time $O(n \log n)$ a linear size data structure, such that finding a disk containing a query point, and deleting this disk, requires amortized time $O(\log n)$ per operation.*

This lemma and Theorem 3.2 yield:

Theorem 5.4 *Let A and B be sets of points in $\mathbb{R}^2$, and $r > 0$. Then the oracle that determines whether $r < Match(A, B)$ requires time $O(n^{1.5} \log n)$.*

Remark 5.5: It is easy to show that Lemma 5.1 holds for any Minkowski L_p metric. Once this lemma is established, the rest of the analysis carries through, and we thus conclude that Theorem 5.4 holds for all L_p.

5.2 Finding the matching

In order to minimize the number of times the oracle is called, we need to efficiently solve the k'th bi-chromatic distance selection problem of Section 4.

Lemma 5.6 *(Katz and Sharir [36]) Let $A, B \subseteq \mathbb{R}^2$ be sets of n points, L_2 the underlying norm and $1 \le k \le n^2$ an integer. Then $dist^{(k)}$ can be found in time $O(n^{4/3} \log^2 n)$.*

Theorem 5.4 and this lemma together with the considerations of Section 4 yield the following:

Theorem 5.7 *Let A, B be sets of n points in $\mathbb{R}^2$. Then Match(A, B) can be computed in time $O(n^{1.5} \log^2 n)$.*

5.3 Accelerating the algorithm

By combining the oracle phase and the generic part, the running time of the algorithm can be improved by a $\log n$ factor.

Recall that $dist^{(i)}$ is the ith largest distance between $a \in A$ and $b \in B$. We maintain a lower bound, $dist^{(\ell)}$ (initially $\ell = 1$), and an upper bound, $dist^{(u)}$ (initially $u = n^2$), on the value of r^*. In Section 5.2 we conducted a binary search on the values $dist^{(1)}, \ldots, dist^{(n^2)}$, thus introducing a log factor. The purpose of this subsection is to eliminate this factor.

In the course of the algorithm, we maintain a maximum matching M of $G[dist^{(\ell)}]$, and use it as an initial matching for $G[dist^{(i)}]$, $(\ell < i < u)$. If $dist^{(i)} < r^*$, we fail to find a perfect matching, and at some stage we even fail to construct $\mathcal{L}$, i.e., we do not reach any exposed vertex of B. If our first attempt to construct $\mathcal{L}$ fails, then M is a maximum matching of $G[dist^{(i)}]$. Otherwise, we update M. Since the size of every matching is bounded by n, M is updated at most n times, and at all other times only one layered graph is constructed.

If the new matching is perfect then we update $dist^{(u)}$ to r. However, we might have wasted a lot of time in constructing several layered graphs. Therefore, a first step toward the desired improvement is to construct $\mathcal{L}$ only a constant number of times for $dist^{(i)} > r^*$.

The key observation is that sometimes we can conclude that $G[dist^{(i)}]$ does not contain a perfect matching, without even finding the maximum matching. For a partial matching $M \subseteq G$ let $\ell(M, G)$ denote the length of the shortest augmenting path for M ($\ell(M, G)$ is equal to the number of layers of the layered graph constructed by the procedure *ConstructLayerGraph*(G, M) of Section 3.1 starting with the matching M).

Lemma 5.8 *Let M be a partial matching of G. If $|M| < n - \sqrt{n}$ and $\ell(M, G) > \sqrt{n}$ then G does not contain a perfect matching.*

Proof: Let M^{max} be a maximum matching of G. $M \oplus M^{max}$ consists of $p = |M^{max}| - |M|$ vertex disjoint augmenting paths $P_1, \ldots, P_p$ (and some alternating cycles). The length of each augmenting path P_i satisfies

$$|P_i| > \ell(M, G)| > \sqrt{n} \ .$$

Since the augmenting paths are vertex disjoint,

$$n \geq \sum_{i=1}^{p} |P_i| > p \cdot \sqrt{n} \ .$$

Hence the number of paths satisfies

$$p < \sqrt{n} \ .$$

Therefore, M^{max} satisfies

$$|M^{max}| = |M| + p < (n - \sqrt{n}) + \sqrt{n} = n \ .$$

Hence M^{max} is not a perfect matching. $\square$

Let $|\mathcal{L}|$ denote the number of layers in the layer-graph $\mathcal{L}$. The following algorithm finds a maximum matching:

```
ℓ ← 1;   u ← n²;   M¹ ← empty matching;
While ℓ + 1 < u Do
    step ← ⌈(u − ℓ + 1)/n^{1/7}⌉;  i ← ℓ + step;
    While i < u Do
        Use the bi-chromatic distance selection algorithm to find dist^(i);
        M ← M^ℓ;
        L ← ConstructLayerGraph( M, G[dist^(i)] ) ;
        While (L contains exposed vertices of B)
         and ( |L| ≤ √n  or  |M| ≥ n − √n )  Do
                Update M by the procedure of Section 3.2;
                L ← ConstructLayerGraph( M, G[dist^(i)] ) ;
        End
        If |M| = n   Then u ← i ;
        Else ℓ ← i;   M^ℓ ← M;   i ← i + step ;
    End
End
```

Lemma 5.9 *The outermost loop (While $\ell + 1 < u$) is executed at most 14 times.*

Proof: Each time the loop is executed then the range, $u - \ell + 1$, decreases at least by a factor of $n^{1/7}$. Since initially, $u - \ell + 1 = n^2$, the number of interactions is at most

$$\log_{n^{1/7}} n^2 = 14 \ .$$

$\square$

We argue now that each execution of the outermost loop takes time $O(n^{1.5} \log n)$. Note first that we solve the bi-chromatic distance selection problem $n^{1/7}$ times. Since by [36] this problem can be solved in time $O(n^{4/3} \log^2 n)$, this sums up to time $O(n^{31/21} \log^2 n) = O(n^{1.5})$. The number of times the layered graph is constructed is bounded by the number of times that its construction procedure terminates successfully—($O(\sqrt{n})$, by Hopcroft and Karp [32], as described in Section 3.2) plus the number of times that this procedure fails, which is no more than the number of times we consult the oracle, which is $O(n^{1/7})$. The time needed for finding augmenting paths consists of the time spent when the layered graph is of depth smaller than $\sqrt{n}$, and the time when the layered graph is of larger depth. In the former case, the time for finding such a path is $O(n \log n)$, while in the latter case there are $O(\sqrt{n})$ paths. Note that there are only 14 phases. This discussion and Remark 5.5 yield the following theorem:

Theorem 5.10 *Let A and B be sets of points in $\mathbb{R}^2$ and L_p the underlying norm for any $1 \le p \le \infty$. Then Match(A, B) can be found in time $O(n^{1.5} \log n)$.*

6 Additional Settings

6.1 Points in 3-space

In a recent paper [22], Efrat et al. obtained results concerning matching points into fat shapes that contain them in two and three dimensions. These algorithms use the matching procedure of Section 3.1, but use different data structures $\mathcal{D}_r(S)$ than those used here. The following result will be useful:

Theorem 6.1 *(Efrat et al. [22]) Let A be a set of n points in $\mathbb{R}^3$, and B a set of n balls in $\mathbb{R}^3$. Then in time $O(n^{11/6+\varepsilon})$ we can either find a matching between A and B, such that each point $a \in A$ is contained in the object of B matched to a, or determine that no such matching exists.*

An immediate consequence of the theorem above is a result for bottleneck matching for two point sets in 3-space.

Corollary 6.2 *Let A and B be two sets of n points in $\mathbb{R}^3$. Then the oracle $r < Match(A, B)$ requires time $O(n^{11/6+\varepsilon})$.*

Proof: For each point $b \in B$ let b^r be a ball of radius r centered at b. $||a - b|| \leq r$ if and only if $a \in b^r$. □

Theorem 6.3 *Let A and B be two sets of n points in $\mathbb{R}^3$. Then $Match(A, B)$ can be found in time $O(n^{11/6+\varepsilon})$.*

Proof: To select the kth distance we use a 3-dimensional bi-chromatic version of the planar distance selection algorithm of Aronov et al. [1], that selects the k-th bi-chromatic distance. The selection requires time $O(n^{7/4+\varepsilon}) = O(\textit{Oracle-time})$, and hence does not affect the overall running time. □

6.2 Arbitrary dimension

Here we assume that $S \subseteq B$ is a set of points in $\mathbb{R}^d$, for fixed d, and the underlying norm is L_∞.

6.2.1 The oracle

Our goal is to obtain a data structure $\mathcal{D}_r(S)$, supporting the operations $\mathbf{neighbor}_r(\mathcal{D}_r(S), q)$ and $\mathbf{delete}_r(\mathcal{D}_r(S), s)$, defined in Section 3.1. We maintain a set $\mathcal{S}$ of d-dimensional cubes of edge-length $2r$, centered at the points of S. $\mathcal{D}_r(S)$ consists of $(d-2)$-level segment-trees (on the projection of the cubes on the first $d-2$ axes) [8, pp. 221-225], and the two-dimensional data structure of Section 5.1 built on the projection of the cubes on the last two axes. It is easy to show how to perform both operations $\mathbf{neighbor}_r$ and $\mathbf{delete}_r$ in this data structure in time $O(\log^{d-1} n)$ each. The preprocessing of the structure is easily shown to be $O(n \log^{d-1} n)$, and does not effect the overall running of the algorithm. Hence $T(n) = O(\log^{d-1} n)$.

The space requirements are as follows: A segment tree on n segments requires space $O(n \log n)$, therefore, the $d-2$ level segment tree requires space $O(n \log^{d-2} n)$. Since the 2-dimensional structure requires linear space, the entire space requirements are $O(n \log^{d-2} n)$. (This data structure is reminiscent of the orthogonal range trees described in Vaidya [46].) We summarize with the following theorem:

Theorem 6.4 *Let A and B be two sets of n points in $\mathbb{R}^d$, for fixed d, and let the underlying norm be L_∞. Then there is an oracle that determines whether $\text{Match}(A, B) < r$ in time $O(n^{1.5} \cdot \log^{d-1} n)$. The space requirements are $O(n \log^{d-2} n)$.*

6.2.2 Finding the matching

As a method to generate critical distances, we use the approach taken by Chew and Kedem [12]. Note that when L_∞ is the underlying norm, r^* is the distance between the projection of some $a \in A$ and $b \in B$ on one of the axes X_m. I.e., let $(q)_m$ be the projection of point q on axis m, then $dist(a_i, b_j) = |(a_i)_m - (b_j)_m|$ for some $m \in \{1, \ldots, d\}$.

While previously we used the oracle to perform a binary search on the distances, here we conduct the search on a superset of size $2dn^2$. For dimension m rearrange A so that $(a_1)_m < (a_2)_m < \cdots < (a_n)_m$, and likewise for B. Now consider the matrix D_m defined as $(D_m)_{i,j} = (a_i)_m - (b_j)_m$. Each row and column of this matrix is monotonically increasing. However, some of the entries are negative. Let $(\bar{D}_m)_{i,j} = (b_j)_m - (a_i)_m$. Thus each distance $dist(a_i, b_j) = |(a_i)_m - (b_j)_m|$ appears as an entry in $\mathcal{D} = \{D_1, \ldots, D_d, \bar{D}_1, \ldots, \bar{D}_d\}$. We now use Frederickson and Johnson's [25] algorithm to select the k'th value of D in time $O(dn)$.

We now discuss the time complexity of the method. The preprocessing consists of sorting $\{(a_i)_m\}_{i=1}^n$ and $\{(b_j)_m\}_{j=1}^n$ for each dimension. Thus the preprocessing requires time $O(dn \log n)$. After this we need $t = \log 2dn^2 = O(\log d + \log n)$ selections each requiring time $O(dn)$, and t oracle calls. Since the time for the oracle calls dominates the preprocessing the selections, we have:

Theorem 6.5 *Let A, B be sets of n points in $\mathbb{R}^d$ ($d \geq 2$ a constant), with L_∞ as the underlying norm. Then finding $\text{Match}(A, B)$ requires time $O(n^{1.5} \log^d n)$ and space $O(n \log^{d-2} n)$.*

Remark 6.6: Naturally, for $d = 2$ and the L_∞ norm, we can use the shortcut of Section 5.3 on this data structure to get a slightly faster method that runs in time $O(n^{1.5} \log n)$.

6.3 Other Objects in the Plane

We next extend our techniques to find matching between a set A of n points and a set $B \subseteq \mathbb{R}^2$ of n objects, which may be one of the following:

(i) a set of disjoint segments, where the distance from a point $q \in \mathbb{R}^2$ to a segment $b \in B$ is defined as the distance from q to its closest point of b.

(ii) a set of points, and each $b_i \in B$ is associated with a non-negative weight w_i, so that for a point $q \in \mathbb{R}^2$, we have $dist(q, b_i) = w_i + ||q - b_i||$.

In both cases the underlying norm may be any L_p-norm. The two cases are handled similarly. To implement the oracle of Section 3 we only need to implement the data structure $\mathcal{D}_r(S)$. For this purpose, we use the dynamic nearest-neighbor scheme of Agarwal et al. [2] who presented such data structures for both these problems. The data structure can be constructed in time $O(n^{1+\varepsilon})$. This data structures enable us to find and delete the closest object of B (either a segment or a point) to the query point q in time $O(n^{\varepsilon})$, and hence to implement $\mathbf{neighbor}_r(\mathcal{D}_r(B), q)$, by checking if the distance of the closest object of S to q is at most r. These data structures support deletions of objects in time $O(n^{\varepsilon})$ as well. Hence $T(n) = O(n^{\varepsilon})$, for any $\varepsilon > 0$, leading to the following result.

Theorem 6.7 *For A and B as defined above, the oracle for testing whether $r < \mathit{Match}(A, B)$ requires time $O(n^{1.5+\varepsilon})$ for any $\varepsilon > 0$.*

Thus, to find the optimal matching, we need to implement efficiently the k-th bi-chromatic distance problem. This may be done in time $O(n^{1.5} \log^3 n)$ by a straightforward extension of the k'th distance selection problem of Agarwal et al. [1]. We summarize:

Theorem 6.8 *Let $A \subseteq \mathbb{R}^2$ be a set of n points, and $B \subseteq \mathbb{R}^2$ a set of either (i) n disjoint segments or (ii), n points, where each point $b_i \in B$ is associated with a non-negative weight w_i, and the distance from a point $q \in \mathbb{R}^2$, to b_i is $||q - b_i|| + w_i$. Then in both cases $\mathit{Match}(A, B)$ can be found in time $O(n^{1.5+\varepsilon})$.*

7 Approximating the Matching

In this section we present an approximation scheme for $r^* = \mathit{Match}(A, B)$ in any dimension. Heffernan and Schirra [28] and Heffernan [29], gave an approximation technique whose running time depends on r^*. We describe an improved technique for finding an approximation to $\mathit{Match}(A, B)$ for A, B point sets in $\mathbb{R}^d$ where the underlying norm is any Minkowski norm L_p.

Definition 7.1 *Let A and B be sets of n objects in $\mathbb{R}^d$, and $\varepsilon > 0$ a parameter. A perfect matching M^{ε} between A and B is an ε-approximating matching if*

$$r^* \leq \max(M^{\varepsilon}(A, B)) \leq r^*(1 + \varepsilon),$$

where $r^* = \mathit{Match}(A, B)$.

We use the data structure of Arya et al. [7] who described a data structure for a set of points $S \subseteq \mathbb{R}^d$, that can report in time $O(f(d, \varepsilon) \log n)$ (for $f(\varepsilon, d) = d(1 + 1/\varepsilon)^d$) an approximated nearest-neighbor of a query point $q \in \mathbb{R}^d$. That is, a point $s \in S$ for which $||q - s||_p \leq (1+\varepsilon) \cdot ||q - s'||_p$, where s' is the closest point of S to q. This data structure can also be dynamized so that a deletion takes time $O(f(d, \varepsilon) \log n)$ — see Bespamyatnikh [9].

Let $\mathcal{D}_r(\cdot)$ denote this data structure. To implement $\mathbf{neighbor}_r(\mathcal{D}_r(B), q)$, we consult $\mathcal{D}_r(B)$ to find b (the approximated nearest neighbor) and if $||b - q||_p \leq r \cdot (1 + \varepsilon)$, we report that $\mathbf{neighbor}_r(\mathcal{D}_r(B), q)$ is b. Otherwise, report that $\mathbf{neighbor}_r(\mathcal{D}_r(B), q) = \emptyset$.

Our approximation scheme consists of applying the procedure of Theorem 3.2, with the approximating data-structure replacing the exact one. Let us refer to this procedure as the *approximating oracle.*

Lemma 7.2 *If for parameter r the approximating oracle returns a positive answer then $r(1+\varepsilon) \geq r^*$. Otherwise, $r < r^*$.*

Proof: Note that if $H \supseteq G$ then any matching in G is also a matching of H. Moreover, if M is a matching of G which can be increased by an augmenting path, then for any matching of size $|M|$ of H there exists an augmenting path in H.

When applying the approximate distance query, we use a graph $G^A \supseteq G[r]$ instead of the graph $G[r]$. The graph G^A contains all the edges of length $\leq r$ and some of the edges whose length is between r and $(1+\varepsilon)r$, but no longer edges, i.e., $G[r] \subseteq G^A \subseteq G[(1+\varepsilon)r]$. Thus if $G[r]$ has a perfect matching, so does G^A. Since all the edges of G^A have length $\leq (1+\varepsilon)r$, if the approximating oracle returns a negative answer, then $G[r]$ does not have a perfect matching and thus $r < r^*$. In case of a positive answer, since all the edges of G^A have length $\leq (1+\varepsilon)r$, the length of the maximum edge in the matching is bounded by $(1+\varepsilon)r$. □

To find an approximate matching, we have to search among the n^2 distances. Even though we do not know how to efficiently solve the k-th bi-chromatic distance selection problem for all dimensions and every norm, in Theorem 6.5 we showed an $O(n \log^d n)$ solution for L_∞ in $\mathbb{R}^d$. We will now use this solution to find an approximation for the L_p-norm for all p.

Theorem 7.3 *Let A and B be sets of n points in $\mathbb{R}^d$, and let $\varepsilon > 0$ be a parameter. Let $r_p^* = Match(A,B)$ where L_p $(1 \leq p \leq \infty)$ is the underlying norm. We can find in time $O(f(\varepsilon,d) \cdot n^{1.5} \log n)$ a matching ${M^\varepsilon}_p$ between A and B satisfying*

$$r_p^* \;\leq\; {M^\varepsilon}_p(A,B) \;\leq\; (1+\varepsilon) r_p^* \;,$$

where $f(\varepsilon,d) = d(1+1/\varepsilon)^d$.

Proof: We first prove the theorem for the L_∞ norm. In this case, we use the techniques of Theorem 6.5 with Lemma 7.2 instead of the exact oracle of Theorem 6.4 to find a matching ${M^\varepsilon}_\infty(A,B)$ that satisfies

$$r_\infty^* \leq {M^\varepsilon}_\infty(A,B) \leq (1+\varepsilon) r_\infty^* \;.$$

Now we use the relationship between Minkowski norms:

$$d\,||x||_\infty \;\geq\; ||x||_1 \;\geq\; ||x||_2 \;\geq\; \cdots \geq ||x||_\infty \;,$$

where $||x||_p$ denotes the length of the vector x using the L_p-norm. Thus $d r_\infty^*$ is an upper bound on $r_p^* = {M^\varepsilon}_p(A,B)$, and $d(1+\varepsilon) r_\infty^\varepsilon$ is an upper bound on r_p^ε. Likewise, $r_\infty^\varepsilon/(1+\varepsilon)$ is a lower bound on r_p^ε. Thus r_p^ε belongs to the interval $[r_\infty^\varepsilon/(1+\varepsilon), d(1+\varepsilon) r_\infty^\varepsilon]$.

We now conduct a binary search on this interval, using the approximating oracle for L_p with $\varepsilon/2$. After $2 + \log d + \log \varepsilon^{-1}$ iterations, the length of the remaining interval is less than $r_p^* \varepsilon/2$. Choosing r_p^ε as the left boundary of this interval satisfies the requirement of the theorem. □

8 The Translation Problem

8.1 The translation oracle

Let A and B be two sets of n points in $\mathbb{R}^2$ and $\tau \in \mathbb{R}^2$. Let $\tau+B = \{\tau+b \,|\, b \in B\}$ denote the set B translated by τ. The *translation problem* is to find τ^*, a translation τ that minimizes $\mathit{Match}(A, \tau + B)$. Let $\rho^* = \mathit{Match}(A, \tau^* + B)$. The *translation oracle* receives $\rho > 0$ as input and determines, whether there exists a translation τ for which $\mathit{Match}(A, \tau + B) \leq \rho$. In other words, the translation oracle determines whether $\rho^* \leq \rho$. Alt et al. [5] presented an $O(n^6)$ time algorithm for the translation oracle. Using our technique we improve the running time of their algorithm to $O(n^5 \log n)$. In Section 8.2 we use this oracle to solve the translation problem.

Let us briefly describe the algorithm of [5], and refer the reader to that paper for further details: If for a translation τ, $\mathit{Match}(A,\ \tau + B) \leq \rho$ then there also exists a translation τ' and a pair of points $a \in A,\ b \in B$ such that $\mathit{Match}(A,\ \tau' + B) = \rho$ and the distance from a to $\tau + b$ is exactly ρ. Hence we can limit our attention to translations τ that bring some point of A to distance ρ (exactly) from some point $b \in B$.

For $a \in A$ let a^ρ denote the disk of radius ρ (in the underlying norm) centered at a, and let A^ρ denote the set $\{a^\rho \,|\, a \in A\}$. For $a \in A, b \in B$, let $\mathit{circ}[a, b, \rho]$ denote the set of translations that bring a to distance ρ from b; this is a circle of radius ρ centered at $b - a$. The algorithm checks for each pair $a \in A$, $b \in B$ if $\mathit{Match}(A, \tau+B) \leq \rho$ for some translation $\tau \in \mathit{circ}[a, b, \rho]$. That is, if there exists a perfect matching in the graph $G_\tau[\rho]$ determined by A and $\tau+B$. Let τ_0 be a fixed translation in $\mathit{circ}[a, b, \rho]$. We first construct $\mathit{Match}(A, \tau_0+B)$. If its value is less than or equal to ρ then we are done. Otherwise, we translate B rigidly by all translations of $\mathit{circ}[a, b, \rho]$. During this process, images of points of B are moved into, or out of disks of A^ρ, implying that edges are inserted into or deleted from the graph $G_\tau[\rho]$. While the image of b revolves around a, each image of a point $b' \in B$ travels along a circle of radius ρ, and enters and exits each disk $(a')^\rho \in A^\rho)$ at most once. Hence, each edge is *born* (inserted to $G_\tau[\rho]$) and dies (deleted from $G_\tau[\rho]$) at most once. The birth/death events are called *critical events*. Therefore, $\mathit{circ}[a, b, \rho]$ contains at most $2n^2$ such critical events. After each critical event, we might need to re-compute $\mathit{Match}(A,\ \tau + B)$. Each critical event adds or deletes a single edge: In the case of a birth, the matching increases by at most one edge. Therefore, we look for an augmenting path which contains the new edge. If an edge of the matching dies, we need to search for a single augmenting path. Thus in order to update the matching, we need to find a single augmenting path in $G_\tau[\rho]$, for which we need only one layered graph.

Alt et al. [5] use standard graph theoretical techniques to find the path, and hence spend $O(n^2)$ time for each critical event. Summed over all pairs $a \in A, b \in B$, the total number of critical events encountered in the course of the algorithm is $O(n^4)$, so the total time spent by the algorithm of Alt et al. is $O(n^4) \times O(n^2) = O(n^6)$.

Instead, we use the procedure of Section 3.1 and Section 5.1 to construct the layered graph. This procedure requires only $O(n \log n)$ time for each augmenting path. Taken over all $O(n^4)$ critical events, the total time sums to $O(n^5 \log n)$. Hence we have proved:

Theorem 8.1 *Given A, B and ρ as above, we can decide in time $O(n^5 \log n)$ whether there* *exists a translation τ for which $Match(A, \tau + B) \leq \rho$.*

8.2 Finding the optimal translation

Alt et al. [5] found the translation itself in time $O(n^6 \log n)$. For pedagogical reasons, we first develop an inefficient polynomial algorithm to find an optimal translation. Then we examine several parallel versions of the algorithm, to which we apply the parametric search paradigm of Megiddo [41] to improve the time to $O(n^5 \log^2 n)$, i.e., the complexity of our final algorithm is only a $\log n$ factor more than that of the translation

We start by introducing a polynomial sized set of *critical radii* which contains ρ^*. Given $a \in A, b \in B$, $\rho > 0$ and $0 \leq \theta < 2\pi$, let $G_{ab}[\rho, \theta]$ denote the graph $G[\rho]$ when b is translated to $a+(\rho\cos\theta, \rho\sin\theta)$. As b revolves around a, θ increases from 0 to 2π and $G_{ab}[\rho, \theta]$ evolves: At some angle θ a vertex $b' \in B$ enters $(a')^\rho \in A^\rho$ and the edge (a', b') is *born*. At some other angle b' leaves $(a')^\rho$ and the edge (a', b') *dies*. The optimal translation τ^* occurs when some edge (a', b') is born in $G_{ab}[\rho^*, \theta]$.

Let us examine more closely the birth angle of an edge $e = (a', b')$ as ρ grows. It is easy to see that the birth occurs when b' lies on the perpendicular bisector of a' and $\beta' = b' - b + a$. Also, the *life arc of* $e' = (a', b')$—the circular arc between the birth and death of e'—is less than half the circle (i.e., less than 180 degrees).

A critical radius of the first type occurs when the life arc degenerates to a point, i.e., when the circles of radius ρ centered at a' and at β' are tangent to each other, i.e., when b' is midway between a' and β'. This occurs when the value of ρ equals $dist(a', \beta')/2$, which we will denote by $\rho_{ab}(e')$. Let $\mathcal{R}^1_{ab} = \{\rho_{ab}(e') \mid e' \in A \times B\}$ denote the set of all such critical radii.

Next we examine another type of critical radii, that might occur for a pair of edges e', e''. At some value ${\rho^{(s)}}_{ab}(e', e'')$ the birth angle of e' coincides with the death angle of e''.

Given e', e'', the values of $\rho_{ab}(e', e'')$ and of ${\rho^{(s)}}_{ab}(e', e'')$, are solutions of a quadratic equation which has at most two solutions. These solutions can be computed in constant time. Let $\mathcal{R}^2_{ab} = \{{\rho^{(s)}}_{ab}(e', e'') | e', e'' \in A \times B\}$.

The optimal translation occurs at an angle θ at which the graph $G_{ab}[\rho^*, \theta]$ changes, i.e., either some edge is added (in which case $\rho^* \in \mathcal{R}^1_{ab}$), or two edges that did not coexist for smaller ρ now belong to the same graph (in which case $\rho^* \in \mathcal{R}^2_{ab}$). Let $\mathcal{R} = \bigcup_{ab} \mathcal{R}^1_{ab} \cup \bigcup_{ab} \mathcal{R}^2_{ab}$ be the set of *critical radii*. From the above discussion $\rho^* \in \mathcal{R}$. Since $|\mathcal{R}^1_{ab}| < n^2$ and $|\mathcal{R}^2_{ab}| < n^4$, $|\mathcal{R}| = O(n^6)$.

Our first algorithm constructs $\mathcal{R}$, sorts it, and then uses the translation oracle to conduct a binary search to find ρ^*. Finding $\mathcal{R}$ requires time $O(n^6)$, sorting it $O(n^6 \log n)$, and the binary search requires $O(\log n)$ oracle calls, each of which requires $O(n^5 \log n)$ time. Thus the entire algorithm requires $O(n^6 \log n)$ time.

Our second algorithm does not construct $\mathcal{R}$ explicitly. Instead, for each a, b we sort the birth and death angles of edges at ρ^*. For that purpose we need to know which edges exist at ρ^*, and for all pairs e', e'' that exist at ρ^*, check whether the death of e' precedes the birth of e'' or vice versa. The difficulty is that we do not yet know ρ^*. However, we may use the translation oracle.

To check whether $e' = (a', b')$ exists at ρ^*, we note that if e' exists at r then it exists for all $r' > r$. We, therefore, compute $\rho_{ab}(e')$ and ask the translation oracle if $\rho_{ab}(e') \leq \rho^*$. If it is then e' exists at ρ^*.

To check whether the death of e' precedes the birth of e'', we apply the translation oracle at $\rho_{ab}(e', e'')$. If it answers that $\rho^* < \rho_{ab}(e', e'')$ then at ρ^* the edge e' died before e'' was born. Otherwise, at ρ^* the edge e' died after e'' was born, i.e., their life arcs are not disjoint. Let us call this test the *overlap oracle*.

Since for each a, b there are at most $2n^2$ birth/death events, there could be a total of about n^4 such events, and sorting them might require $\Theta(n^4 \log n)$ comparisons, i.e., $\Omega(n^4 \log n)$ oracle calls—far too many.

We use the parametric search paradigm of Megiddo [41] to reduce the number of calls to the overlap oracle. Again, we assume that the reader is familiar with this technique, and refer to Efrat et al. [23] for a similar application. To this end, we consider a parallel algorithm in which for each pair a, b the sorting is performed by a separate processor. We now describe an efficient sequential simulation. Let us consider each processor's first call to the overlap oracle. The parallel algorithm performs all these calls in parallel. The sequential simulation answers all the comparisons by first sorting the radii $\rho(e'_1, e''_1) \ldots, \rho(e'_{n^2}, e''_{n^2})$ and then performing a binary search, to find the smallest $\rho(e'_i, e''_i)$ that satisfies the translation oracle. We need therefore time $O(n^2 \log n)$ for the sort and $O(\log n)$ calls to translation oracle—a total of $O(n^5 \log^2 n)$ time for each parallel comparison step. However, the entire algorithm involves $n^2 \log n$ such steps, i.e., a total of $O(n^7 \log^3 n)$—again far too much.

To get a good algorithm, we increase the degree of parallelism. Each pair a, b conducts its sort in parallel using the depth $O(\log n)$ parallel AKS sorting network of Ajtai et al. [3] that sorts $O(n^2)$ critical radii with $O(n^2 \log n)$ comparisons.

In each parallel step each of the n^2 networks performs n^2 comparisons, thus a total of $O(n^4)$ comparisons are conducted in parallel. The sequential simulation sorts all these n^4 radii, then performs a binary search calling the translation oracle $O(\log n)$ times. Now we may deduce in constant time whether a critical radius is smaller than ρ^*—thus answering all the overlap oracles in time $O(n^4 \log n + n^5 \log^2 n)$. Since the combined network has depth $\log n$, the total number of oracle calls is $O(\log^2 n)$. Thus in time $O(n^5 \log^3 n)$ we have ordered all the critical events at ρ^*. Since a critical event happens at ρ^*, the value of ρ^* is the minimal translation oracle call that returned a positive answer.

Cole [15] studied parallel sorts on sorting networks and showed how to reorder the comparisons, so as to save a $\log n$ factor. His technique is applicable in any setting where one uses the AKS sorting network as a generic algorithm. Using this technique, the number of calls to the translation oracle is reduced from $\log^2 n$ to $\log n$. See [15] for more details. We summarize these results:

Theorem 8.2 *Given A, B as above, the translation problem can be solved in time $O(n^5 \log^2 n)$.*

8.3 Approximating the optimal translation

We next note that while finding a translation τ^* which minimizes $Match(A, \tau + B)$ is a non-trivial problem for which only high degree polynomial algorithms are known, and only in the plane, it is easy to find a translation that brings $Match(A, \tau + B)$ within a factor of $1 + diam(p,d)$ of the optimum, where $diam(p,d) = ||(1,\ldots,1)||_p$ is the diameter of the d-dimensional unit cube in the underlying norm norm L_p, $(1 \leq p \leq \infty)$, i.e., for finite p, $diam(p,d) = \sqrt[p]{d}$ and $diam(\infty,d) = 1$.

For a point $s \in \mathbb{R}^d$ let s_i denote the i-th coordinate of s. For a set of points $S \subseteq \mathbb{R}^d$ let $LL(S)$ denote the point in $\mathbb{R}^d$ whose i-th coordinate is equal to the minimum among the values of the i-th coordinate of all the points of S for each $i = 1,\ldots d$. In the plane, $LL(S)$ is the lower-left corner of the smallest axis-parallel rectangle that encloses S, and analogously in higher dimensions. Henceforth, we assume, with no loss of generality, that $LL(A)$ coincides with the origin. Thereby, for all $a \in A$ and $i = 1,\ldots,d,$ we have $a_i \geq 0$.

We can identify a translation τ of B with the image of $LL(B)$. Let τ^0 be the translation that maps $LL(B)$ to the origin and let $\delta = \tau^* - \tau^0$.

Lemma 8.3 *For $i = 1,\ldots,d$*

$$|\delta_i| \leq \rho^* \ .$$

Proof: Let M^* be the optimum matching for $(A, B + \tau^*)$.

Case 1 $\delta_i < 0$: Consider the point $b' \in B$ for which $(b')_i$ is minimal. Since $(b')_i = (LL(B))_i$ we have $(b' + \tau^0)_i = 0$. Let a' be the point matched to b' in M^*, i.e., $(a', b') \in M^*$. Since we chose the origin to be $LL(A)$, $(a')_i \geq 0$,

$$\begin{aligned} \rho^* &\geq dist(a', b' + \tau^*) = dist(a', b' + \tau^0 + \delta) \geq \left|(a')_i - (b' + \tau^0 + \delta)_i\right| \\ &= |(a')_i - \delta_i| = (a')_i + |\delta_i| \geq |\delta_i| \ . \end{aligned}$$

Case 2 $\delta_i > 0$: Choose $a'' \in A$ to be a point such that $(a'')_i = 0$ (this is possible since the origin was chosen to be $LL(A)$). Now choose $b'' \in B$ such that $(a'', b'') \in M^*$. Since $(a'')_i = 0$ and $(b'' + \tau^0)_i \geq 0$,

$$\begin{aligned} \rho^* &\geq dist(a'', b'' + \tau^*) = dist(a'', b'' + \tau^0 + \delta) \geq \left|(a'')_i - (b'' + \tau^0 + \delta)_i\right| \\ &\geq \left|(b'' + \tau^0)_i + \delta_i\right| = (b'' + \tau^0)_i + \delta_i \geq \delta_i = |\delta_i| \ . \end{aligned}$$

□

The following theorem is reminiscent of of a similar result of Alt at al. [4].

Theorem 8.4 *Let $1 \leq p \leq \infty$ and $diam(p,d)$ as above. Then τ^0 satisfies*

$$Match(A, B + \tau^0) \leq (1 + diam(p,d))\, Match(A, \tau^* + B) \ .$$

Proof: Let $(a,b) \in M^*$ be a pair for which $dist(a, b + \tau^0)$ is maximum. Consider the matching M^* for A and $B + \tau^0$. Since $Match(A, B + \tau^0)$ is the minimum value over all matchings for A and $B + \tau^0$, $\rho^0 \leq dist(a, b + \tau^0)$. Therefore,

$$\begin{aligned} \rho^0 &\leq dist(a, b + \tau^0) = dist(a, b + \tau^* + \delta) \leq ||a - (b + \tau^* + \delta)|| \\ &\leq ||a - (b + \tau^*)|| + ||\delta|| = \rho^* + ||(\pm\rho^*,\ldots,\pm\rho^*)|| = (1 + diam(p,d))\, \rho^* \ . \end{aligned}$$

□

If we care for a better approximation for $\rho^* = \mathit{Match}(A, \tau^* + B)$, we use the following approach, borrowing some ideas from Efrat [24]. Let $\mathit{cube}[r]$ denote the axis-parallel cube of edge lengths $2r$ centered at the origin. Lemma 8.3 states that the optimal translation τ^* brings $LL(B)$ to a point in $\mathit{cube}[\rho^*]$.

Let $0 < \varepsilon < 1$, let τ^0 be the translation of Theorem 8.4 and $\rho^0 = \mathit{Match}(A, \tau^0 + B)$. Consider a grid Γ centered at the origin with cell size $\gamma = \varepsilon\rho^0/\left(2\mathit{diam}(p,d)\right)$. The distance of any point of $\mathit{cube}[\rho^0]$ to its closest grid point of Γ is at most $\gamma \mathit{diam}(p,d) = \frac{\varepsilon}{2}\rho^0$. Let $T(\Gamma)$ be the set of all translations that bring $LL(B)$ to some grid point of Γ in $\mathit{cube}[\rho^0]$. Since this cube properly contains $\mathit{cube}[\rho^*]$, Lemma 8.3 implies that distance of the optimal translation τ^* to some grid point of Γ is at most $\frac{\varepsilon}{2}\rho^0$.

For each translation $\tau \in T(\Gamma)$ we approximately evaluate $\mathit{Match}(A,\ \tau + B)$ using the procedure of Theorem 7.3, and choose τ^ε to be the best one.

Consider any two points $a \in A$, $b \in B$.

$$\begin{aligned} \mathit{dist}(a, \tau^\varepsilon + b) &\leq \mathit{dist}(a, \tau^* + b) + \mathit{dist}(\tau^* + b, \tau^\varepsilon + b) = \mathit{dist}(a, \tau^* + b) + \mathit{dist}(\tau^*, \tau^\varepsilon) \\ &\leq \mathit{dist}(a, \tau^* + b) + \frac{\varepsilon}{2}\rho^0 \ . \end{aligned}$$

Thus, if $\rho^* = \mathit{dist}(a^*, \tau^* + b^*)$, then for any two points a, b matched by the approximated match at τ^ε, we have

$$\mathit{dist}(a, \tau^\varepsilon + b) \leq \mathit{dist}(a, \tau^* + b) + \frac{\varepsilon}{2}\rho^0 \leq \mathit{dist}(a^*, \tau^* + b^*) + \frac{\varepsilon}{2}2\rho^* = \rho^*(1+\varepsilon) \ .$$

The number of grid points of Γ is

$$|\Gamma| = \left(1 + \frac{2\rho^0}{\gamma}\right)^d = \left(1 + \frac{2\rho^0}{\varepsilon\rho^0/(2\mathit{diam}(p,d))}\right)^d = \left(1 + \frac{4}{\varepsilon}\mathit{diam}(p,d)\right)^d \ .$$

By Theorem 7.3, finding the approximate matching for each grid point requires time $O(d(1+1/\varepsilon)^d \cdot n^{1.5} \log n \log \varepsilon^{-1})$. Hence we have:

Theorem 8.5 *Let A and B be sets of n points in $\mathbb{R}^d$ and L_p ($1 \leq p \leq \infty$) the underlying norm. Then there exists an algorithm that for all $\varepsilon > 0$ finds in time*

$$O\left(d\left(1 + \frac{1}{\varepsilon}\right)^d \left(1 + \frac{4}{\varepsilon}\mathrm{diam}(p,d)\right)^d \cdot n^{1.5} \log n \log \varepsilon^{-1}\right)$$

a translation τ^ε such that

$$\rho^\varepsilon = \mathit{Match}(A, B + \tau^\varepsilon) \leq (1+\varepsilon)\rho^* \ ,$$

where ρ^* is value of the matching at the optimal translation.

For constant d the time is $O(\varepsilon^{-2d} \cdot n^{1.5} \log n \log \varepsilon^{-1})$.

9 Related problems to the bottleneck matching

Several related problems are easily tackled by our method.

9.1 Partial matching

Let A and B be sets of objects (not necessarily with the same cardinality), and let $1 \le p \le \min\{|A|, |B|\}$ be an integer. The problem is to find r^p, the smallest r for which a matching of cardinality p exists in $G[r]$. This problem might arise in pattern matching, when we suspect that some of the points are superfluous, or we seek the appearance of a relatively small pattern A inside a large picture B.

To find whether $r < r^p$ we use the methods of Section 3. There we increased the matching incrementally, so after matching p pairs, we can answer whether $r < r^p$. The time spent by this procedure is $O(|A|^{1.5} \cdot T(|B|))$, where $T(|S|)$ is the time required to perform an operation on $\mathcal{D}_r(S)$.

To find r^p, we need to be able to solve the k-bi-chromatic distance selection problem efficiently. Here too the methods of Section 4 can be applied.

9.2 Finding a batch of partial matchings

When the number of points is not known in advance, we can further modify the algorithm, so for every $1 \le m \le n$ in time $O(n^{1.5+\varepsilon} + \eta n^{1+\varepsilon})$ we find a batch of values $r^*_m, r^*_{m+1}, \ldots, r^*_{m+\eta}$. The proposed procedure is faster than separately finding for each $i = m, \ldots, m+\eta$ the value of $Match^*_i$—the best partial match on i points This is achieved as follows: We first find $Match^*_m$ using the procedure described in Section 9.1 above. Next we find η augmenting paths, such that each such path augments $Match^*_{m+i}$ to $Match^*_{m+i+1}$, for $i = 0, \ldots, \eta - 1$. An augmenting path is found as follows. Let $A_0 \subseteq A$ and $B_0 \subseteq B$ be the exposed vertices of A and B in $Match^*_{m+i}$. We maintain a forest of augmenting trees containing A_0. Let $A_1 \subseteq A$ be all nodes reachable from A_0 via an augmenting path of one of the trees in the forest, and let $\bar{B} \subseteq B$ be all nodes *not* in any tree of the forest. At each step of the algorithm we add to the forest the edge (a, b) such that

$$dist(a, b) \;=\; \min\{dist(a, b) \mid a \in A_1,\ b \in \bar{B}\}$$

If b is an exposed vertex ($b \in B_0$) an augmenting path has been found. Otherwise, $(b, a') \in Match^*_{m+i}$ for some $a' \notin A_1$. We add (a, b) and (b, a') to the forest.

Adding an edge to the forest is done in $O(n^{\varepsilon})$ time by inserting the vertex a' to A_1 and deleting b from $\bar{B}$, using the procedure of Agarwal et al. [2] for maintaining the closest bi-chromatic pair.

Since each vertex can be added to forest only once, updating $Match^*_{m+i}$ to $Match^*_{m+i+1}$ requires $O(n)$ update operations, i.e., a total of $O(n^{1+\varepsilon})$ time. To update the forest we need to delete the tree whose root $a_0 \in A_0$ was matched. This also involves at most $2n$ update operations per augmenting path. Thus the time required for the entire batch $Match^*_m, \ldots, Match^*_{m+\eta}$ is $O(n^{1.5} \log n + \eta n^{1+\varepsilon})$.

9.3 Finding the longest perfect matching

Let us describe briefly another set of problems. Let A and B be two sets of n points, $r > 0$ and let $\overline{G}[r]$ denote that graph on $A \cup B$ whose edges are pairs of points of distance *at least*

r. The problem is to find $\overline{Match}(A, B)$, the largest r for which a perfect matching exists in $\overline{G}[r]$ (in this scenario, this problem is the *dual* of finding $Match(A, B)$). Surely, our basic scheme will do here as well, provided we obtain a data-structure $\mathcal{D}_r(B)$ that allows:

(i) finding a point of B whose distance from a query point q is at least r,

(ii) deleting a point from B.

Fortunately, these operations can be done efficiently in the Euclidean planar case by maintaining the *Circular Hull* of B — namely the region consisting of the intersection of all disks of radius r containing B. Hershberger and Suri [31] showed how both these operations can be handled in (amortized) time $O(\log n)$. Hence $\overline{Match}(A, B)$ can be found in this scenario in time $O(n^{1.5} \log n)$.

Recall that finding $Match(A, B)$, when A and B are point-sets in $\mathbb{R}^3$ can be done in time $O(n^{11/6+\varepsilon})$ (Theorem 6.3). It is surprising, in our opinion, that $\overline{Match}(A, B)$ can be found in this setting much faster; we describe only the data structure and use the same oracle and generic algorithms used in the proof of Theorem 6.3.

Let $S \subseteq B$, and fix a parameter r. Trivially, if some point $b \in S$ can be matched in $\overline{G}[r]$ to a point $a \in A$, then $a \notin \bigcap_{b_i \in S} b_i^r$, where b^r is the three-dimensional ball of radius r centered at b. Agarwal et al. [2] proposed a data structure $\Xi(S)$ for a set of congruent three-dimensional balls. This data structure enables us to determine whether a point is in $\bigcap_{s \in S} s^r$, and to delete a ball in time $O(n^\varepsilon)$. The data structure may be constructed in time $O(n^{1+\varepsilon})$. To use this structure, we build a balanced binary tree $\mathcal{T}$, whose leaves are the points of S, and each internal node v is associated with S_v, the set of balls whose centers are associated with the leaves of v's subtree. We also associate with v the data structure $\Xi_v = \Xi(S_v)$. To perform $\textbf{neighbor}_r(\mathcal{D}_r(S), q)$ (that is, to find $s \in S$ such that q does not lie in s^r), we use Ξ_v where $v = root(\mathcal{T})$, to find if $q \notin \bigcap_{s \in S_v} s^r$, and if so, we recursively check each of its two children to find (at least) one v', such that $q \notin \bigcap_{s \in S_{v'}} s^r$. We repeat this process until v' is a leaf, and then return the singleton v'. Deletion is carried out in a trivial fashion. Note that both these operations are done in time $O(n^\varepsilon)$, so, by plugging this data structure into the oracle of Section 3 we get the following theorem:

Theorem 9.1 *Let A and B be two sets of n points in $\mathbb{R}^3$. Then $\overline{Match}(A, B)$, the longest perfect matching, can be found in time $O(n^{1.5+\varepsilon})$ for any $\varepsilon > 0$.*

9.4 Computing a most uniform matching

The following problem has applications in pattern matching [20]. Let A and B be two sets of n points in the plane. We seek $Match^*_U$, a matching $Match$ that minimizes the difference $\max(Match) - \min(Match)$. Let $G[r, r']$ denote the bipartite graph whose set of vertices is $A \cup B$, and there is an edge between $a \in A$ and $b \in B$ iff $r \le ||a - b|| \le r'$, where $||a - b||$ is the Euclidean distance between a and b. Recall that $dist^{(1)}, \ldots, dist^{(n^2)}$ denote the n^2 distances between points of A and points of B, in increasing order. We refer to them as *critical distances*, and we assume, for simplicity of exposition, that they are all distinct. We seek $1 \le i < j \le n^2$ such that $G[dist^{(i)}, dist^{(j)}]$ contains a perfect matching $Match$, and the difference $dist^{(j)} - dist^{(i)}$ is as small as possible; $Match$ is then the desired matching. Our algorithm maintains a maximum matching in $G[dist^{(i)}, dist^{(j)}]$. We start with $G[dist^{(i)}, dist^{(j)}]$ for $i = j = 1$, and with the matching consisting of the single edge whose corresponding distance is $dist^{(1)}$. The top level of the algorithm consists of the following loop. If there is no perfect matching in $G[dist^{(i)}, dist^{(j)}]$ we increase j by one, else we increase i by one; in either case we compute a maximum matching in the new graph, and repeat. Increasing j adds a single edge to the graph, and we check whether the size of the maximum matching increases by one. Increasing i deletes a single edge from the graph, and, if this edge was in the current maximum matching, we must check whether the size of the maximum matching remains as before (or decreases by one). Both these checks are done by trying to compute an augmenting path for the current matching using a slightly simpler version of the procedure of Section 3.1, as we did in Section 8. (In the latter check, we do this after deleting the edge corresponding to the distance $dist^{(i)}$ from the current matching.) If such a path exists then the answer is positive and we update the current maximum matching; otherwise, the answer is negative. If a perfect matching was found, then we compare the appropriate difference, i.e., either $dist^{(j+1)} - dist^{(i)}$ or $dist^{(j)} - dist^{(i+1)}$, with the difference corresponding to the best perfect matching found so far. Clearly, the most uniform matching will be discovered in this way, and the number of times we need to compute an augmenting path is $O(n^2)$.

As in Section 3.1 we need a data structure $\mathcal{D}_{r,r'}(P)$ over a set of points $P \subseteq B$, supporting the following operations, in amortized time $O(n^{1/3} \log n)$.

- $\mathbf{neighbor}_{r,r'}(\mathcal{D}_{r,r'}(P), q)$: For a query point q, return a point $p \in P$ whose distance from q is between r and r'. If no such p exists, then $\mathbf{neighbor}_{r,r'}(\mathcal{D}_{r,r'}(P), q) = \emptyset$.
- $\mathbf{delete}_{r,r'}(\mathcal{D}_{r,r'}(P), p)$: Delete the point p from P.

We show below that such a data structure can be constructed in time $O(n^{4/3} \log n)$, and that an augmenting path can be computed within the same time bound. Since we repeat this process $O(n^2)$ times, we obtain an $O(n^{10/3} \log n)$-time algorithm for computing a most uniform matching. The data structure is based on the following theorem.

Theorem 9.2 *(Katz and Sharir [36]) Let $\mathcal{M}$ be a set of m congruent annuli and A a set of n points in the plane. One can compute the set of pairs*

$$\mathcal{Z} = \{\ (c, a)\ \mid\ c \in \mathcal{M},\ a \in A,\ \text{and } a \text{ lies in } c\ \}$$

as a collection $\{\mathcal{M}_u \times A_u\}_u$ of complete edge-disjoint bipartite graphs, in time and space $O((m^{2/3}n^{2/3} + m + n) \log m)$. (That is, for each annulus-point pair $(c, a) \in \mathcal{Z}$, there exists a single graph $\mathcal{M}_u \times A_u$ such that $c \in \mathcal{M}_u$ and $a \in A_u$, and for each graph $\mathcal{M}_u \times A_u$ and for each $c \in \mathcal{M}_u$, $a \in A_u$, the pair (c, a) is in $\mathcal{Z}$.) The number of graphs is $O(m^{2/3}n^{2/3} + m + n)$, and $\sum_u |A_u|$, $\sum_u |\mathcal{M}_u| = O((m^{2/3}n^{2/3} + m + n) \log m)$.

For each point $b \in B$ we draw the annulus of radii r and r' that is centered at b. Let $\mathcal{M}$ be the set of these annuli. Clearly, $r \le ||q - b|| \le r'$ for a point q iff q lies inside the annulus associated with b. We apply Theorem 9.2 to the sets A and $\mathcal{M}$ and obtain in time $O(n^{4/3} \log n)$ a collection of $O(n^{4/3})$ bipartite graphs $H_u = \mathcal{M}_u \times \mathcal{A}_u$ such that $\sum_u |A_u|, \sum_u |\mathcal{M}_u| = O(n^{4/3} \log n)$.

The operations are implemented as follows:

neighbor$_{r,r'}(\mathcal{D}_{r,r'}(B), a)$**:** Find any bipartite graph H_u such that $a \in A_u$, and return any $b \in \mathcal{M}_u$.

delete$_{r,r'}(\mathcal{D}_{r,r'}(B), b)$**:** For each graph H_u such that $b \in \mathcal{M}_u$, remove b from $\mathcal{M}_u$, and, if after the removal $\mathcal{M}_u = \emptyset$, remove the entire graph H_u (i.e., remove the points in A_u).

Each graph H_u is represented by two lists A_u and $\mathcal{M}_u$. In addition, for each $a \in A$ we maintain a list L_a of the occurrences of a in the lists A_u. All lists are doubly linked to enable deletions, and there is a pointer from the occurrence of a in a list A_u back to the entry in L_a which points to this occurrence. Similar lists L_b ($b \in B$) are constructed according to the $\mathcal{M}_u$ lists. Once the complete bipartite graphs have been constructed, the implementation of **neighbor**$_{r,r'}$ and **delete**$_{r,r'}$ is a matter of list processing.

Since each occurrence of a or b in $\{H_u\}$ is removed only once, the time needed for n **neighbor**$_{r,r'}$ and **delete**$_{r,r'}$ operations is $O(n^{4/3} \log n)$, as asserted.

Remark 9.3: Note that if the underlying norm is L_∞, we can find a most uniform matching in time $O(n^3 \log^d n)$, for any fixed $d \geq 2$.

This is done by constructing a d-level orthogonal range tree for the set B. The points of B lying at distance between r and r' of a query point q, lie in a region that is defined as the difference between two concentric cubes; namely, the cube centered at q with edge length $2r'$ and the cube centered at q with edge length $2r$. This region can be partitioned into $2d$ disjoint axis-aligned boxes, on each of which a query can be performed. Details are standard and hence omitted.

Summarizing, we have:

Theorem 9.4 *Let A and B be two sets of n points. It is possible to compute a most uniform matching in time $O(n^{10/3} \log n)$ when the points are in $\mathbb{R}^2$ and the underlying norm is L_2, or in time $O(n^3 \log^d n)$ when the points are in $\mathbb{R}^d$, $d \geq 2$, and the underlying norm is L_∞.*

Acknowledgments

Discussions with Pankaj K. Agarwal, Esther M. Arkin, Yefim Dinitz, Rafi Hassin and Micha Sharir contributed significantly to this paper, and we are grateful to them.

References

[1] P.K. Agarwal, B. Aronov, M. Sharir and S. Suri, Selecting distances in the plane, *Algorithmica* 9 (1993), 495–514.

[2] P.K. Agarwal, A. Efrat and M. Sharir, Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications, *Proceedings 11th Annual Symposium on Computational Geometry*, 1995, 39–50.

[3] M. Ajtai, J. Komlós and E. Szemerédi, Sorting in $c \log n$ parallel steps, *Combinatorica* 3 (1983), 1–19.

[4] H. Alt, B. Behrends, and J. Blömer, Approximate matching of polygonal shapes, *Proceedings 7th Annual Symposium on Computational Geometry*, 1991, 186–193.

[5] H. Alt, K. Mehlhorn, H. Wagener and E. Welzl, Congruence, similarity and symmetries of geometric objects, *Discrete and Computational Geometry* 3 (1988), 237–256.

[6] E.M. Arkin, K. Kedem, J.S.B. Mitchell, J. Sprinzak and M. Werman, Matching points into noise regions: combinatorial bounds and algorithms, *Proceedings 2th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1991, 42–51.

[7] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman and A. Wu, An optimal algorithm for approximate nearest neighbor searching, *Proceedings 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1994, 573–582.

[8] M. de Berg, M. van Krevekd, M. Overmars, O. Schwarzkopf, Computational Geometry, Algorithms and Applications, Springer-Verlag, 1997.

[9] S.N. Bespamyatnikh. Dynamic algorithms for approximate neighbor searching. *Proceedings 8th Candian Conf. Computational Geometry*, 1996, 252–257.

[10] C. Berge, Two Theorems in Graph Theory, *Proc. Natl. Acad. Sci. U.S.* 43 (1957), 842–844.

[11] S. Buss and P. Yianilos, Linear and $O(n \log n)$ time minimum-cost matching algorithms for quasi-convex tours, *Proceedings 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1994, 65–76.

[12] L.P. Chew and K. Kedem, Improvements on geometric pattern matching problems, *Proceedings 3th Scand. Workshop on Algorithms Theory*, LNCS Vol. 621, 1992, 318–325.

[13] L.P. Chew, D. Dor, A. Efrat and K. Kedem, Geometric pattern matching in d-dimensional space, *Third European Symposium on Algorithms (ESA)*, LNCS Vol. 979, 1995, 264–279.

[14] L.P. Chew, M.T. Goodrich, D.P. Huttenlocher, K. Kedem, J.M. Kleinberg and D. Kravets, Geometric pattern matching under Euclidean motion, *Proceedings 5th Candian Conf. Computational Geometry*, 1993, 151–156.

[15] R. Cole, Slowing down sorting networks to obtain faster sorting algorithms, *J. ACM* 34 (1987), 200–208.

[16] E.A. Dinitz, Algorithm for solution of a problem of maximum flow in a network with power estimation, *Soviet Math Dokl.* 11, (1970) 248–264.

[17] H. Edelsbrunner, L. Guibas and J. Stolfi, Optimal point location in a monotone subdivision, *SIAM J. Comput.* 15 (1986), 317–340.

[18] J. Edmonds and R.M. Karp, Theoretical improvements in algorithmic efficiency for network flow problems, *J. ACM* 19 (1972), 248–264.

[19] A. Efrat, Geometric Location Optimization, Ph.D. Dissertion, Tel-Aviv University, 1998.

[20] A. Efrat and A. Itai, Improvements on bottleneck matching and related problems using geometry, *Proceedings 12th Annual Symposium on Computational Geometry*, 1996, 301–310.

[21] A. Efrat and M.J. Katz, Computing Fair and Bottleneck Matchings in Geometric Graphs, *Proc. 7th Int. Symp. on Algorithms and Computation (ISAAC)*, 1996, 115–125.

[22] A. Efrat, M.J. Katz, F. Nielsen, and M. Sharir, Dynamic Data Structures for Fat Objects and their Applications, *Proceedings 5th Workshop on Algorithms and Data Structures*, 1997 LNCS Vol. 1272, 297–306.

[23] A. Efrat, M. Sharir and A. Ziv, Computing the smallest k-enclosing circle and related problems, *Computational Geometry: Theory and Applications* 4 (1995), 119–136.

[24] A. Efrat, Finding approximate matching of points under translation, Manuscript, 1995.

[25] G.N. Frederickson and D.B. Johnson, Generalized selection and ranking sorted matrices, *SIAM J. Computing* 13 (1984), 14–30.

[26] S.K. Gupta and A.P. Punnen, Minimum deviation problems, *Oper. Res. Lett.* 7 (1988), 201–204.

[27] D. Halperin and M. Sharir, New bounds for lower envelopes in three dimensions, with applications to visibility of terrains, *Discrete and Computational Geometry* 12 (1994), 313–326.

[28] P.J. Heffernan and S. Schirra, Approximate decision algorithms for point set congruence, *SIAM J. Computing* 8 (1992), 93–101.

[29] P.J. Heffernan, Generalized approximate algorithms for point set congruence, *Proceedings 3th Workshop on Algorithms and Data Structures*, 1993, LNCS Vol. 709, 373–384.

[30] J. Hershberger and S. Suri, Applications of a semi-dynamic convex hull algorithm, *Proceedings 2th Scand. Workshop on Algorithms Theory*, 1990, LNCS Vol447, 380–392.

[31] J. Hershberger and S. Suri, Finding tailored partitions, *J. Algorithms* 12 (1991), 431–463.

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

[33] D.P. Huttenlocher and K. Kedem, Efficiently computing the Hausdorff distance for point sets under translation, *Proceedings 6th Annual Symposium on Computational Geometry*, 1990, 340–349.

[34] D.P. Huttenlocher, K. Kedem and M. Sharir, The upper envelope of Voronoi surfaces and its applications, *Discrete and Computational Geometry* 9 (1993), 267–291.

[35] A.V. Karzanov. Exact complexity bound for a max-flow algorithm applied to "representatives" problem. *Voprosy Kibernetiki (Proc. of the Seminar on Combinatorial Mathematics, Moscow*, January 1971). Akad. Nauk SSSR, Scientific Council on the Complex Problem "Kibernetika", 1973, 66–70 (in Russian).

[36] M.J. Katz and M. Sharir, An expander-based approach to geometric optimization, *SIAM J. Computing* 26 (1997), 1384–1408.

[37] H. Kuhn, The Hungarian method for the assignment problem, *Naval Research Logistics Quarterly* 2 (1955), 83–97.

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

[39] L. Lovasz and M.D. Plummer, Matching Theory, *Annals of Discrete Mathematics; 29 North-Holland Mathematics Studies* 121 (1986).

[40] S. Martello, W.R. Pulleyblank, P. Toth and D. de Werra, Balanced optimization problems, *Operations Research Letters* 3 (1984), 275–278.

[41] N. Megiddo, Applying parallel computation algorithms in the design of serial algorithms, *J. ACM* 30 (1983), 852–865.

[42] M.H. Overmars and J. van Leeuwen, Maintenance of configurations in the plane, *J. Computer and Systems Sciences* 23 (1981), 166–204.

[43] M. Sharir, Almost tight upper bounds for lower envelopes in higher dimensions, *Discrete and Computational Geometry* 12 (1994), 327–345.

[44] M. Sharir, A Near-Linear Algorithm for the Planar 2-Center Problem, *Discrete and Computational Geometry* 18 (1997), 125–134.

[45] R.E. Tarjan, *Data Structures and Network Algorithms,* SIAM, Philadelphia, PA (1983).

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