Improvements on Bottleneck Matching and Related Problems, Using Geometry

Alon Efrat[*] Alon Itai[†]

Abstract

Let A and B be two sets of n objects in $\mathbb{R}^d$. We propose to use *bottleneck matching* as a convenient way for measuring the resemblance between them, and present several algorithms for computing, as well as approximating, this resemblance. The running time of all these algorithms is close to $O(n^{1.5})$. For instance, if the objects are points in the plane, the running time is $O(n^{1.5}\log n)$.

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

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, 26, 25]). However, measuring the resemblance by the Hausdorff Distance suffers from 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). 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 useless. See Figure above.

In this paper we propose a different measure of similarity: We assume that both images A and B have the same number of objects, and we seek a perfect bipartite matching between A and B, such that the maximal distance between any matched pair of objects is minimized. 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 $\mathit{Match}(A, B)$ 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, 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 relatively to the other, has hardly been investigated, and the best known algorithms are either computationally inefficient [6], or significantly restrict the inputs [7].

For the case where the sets A and B are points in the plane, Vaidya [35] 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 $O(n^2\log^3 n)$-time algorithm for the L_∞ distance. The solution of the Euclidean case has recently been improved by Agarwal, Efrat and Sharir [2] to $O(n^{2+\varepsilon})$ (for any $\varepsilon > 0$). However, the resulting algorithms remain relatively complicated. See also [10] for fast algorithms for other types of graphs

[*]School of Mathematical Sciences, Tel Aviv University, Tel-Aviv 69982, Israel. *alone@cs.tau.ac.il*

[†]Department of Computer Science, Technion – Israel Institute of Technology, Haifa 32000, Israel. *itai@cs.technion.ac.il*

related to geometric configurations.

For computing *Match*(A,B) we use a parametric search technique, reminiscent of the one proposed by Megiddo [33]. We introduce in Section 3 an oracle that determines, for a parameter r, whether $r \leq$ *Match*(A,B). In Section 4 this oracle is used to find the minimal r for which a perfect matching exists. The exact running times depend on the norm and the dimension. When $A, B \subseteq \mathbb{R}^2$ are point-set, and the underlying norm is L_2, (the *planar Euclidean point-sets case*) our algorithm runs in time $O(n^{1.5}\log n)$. In the case that A is a set of n points, B is a set of segments in the plane, and the norm is an arbitrary L_p, or in the case that A and B are sets of points in the plane and the distance is additively weighted, the running time is slightly worse—$O(n^{1.5+\varepsilon})$, for any $\varepsilon > 0$. When the norm is L_∞ and A, B are point-sets in the d-space, (the *L_∞ point-sets d-space case*) the running time is to $O(n^{1.5}\log^{d-1} n)$. These results are listed in Theorem 4.7.

Section 5 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 the constant of proportionality depends on the dimension and on ε. We believe that these schemes are relatively easy to implement, with reasonably small constant of proportionality, and therefore would do reasonably well in practice.

We also show in Section 6 an application of our technique for the following 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. [6] 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 the minimum such ρ in time $O(n^5\log^2 n)$. Finally, we present a scheme to approximate the translation that minimizes this distance among all possible translations.

In Section 7 we discuss two problems strongly related to the matching problem. The first is the *Partial matching* in which we are given A, B (not necessarily of the same cardinality) and a parameter $1 \leq k \leq \min\{|A|,|B|\}$, and we seek a matching of cardinality k for which its longest edge is as short as possible. The second problem is the *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 enough, in the case of points in $\mathbb{R}^3$, this problem is easier to tackle than the problem of finding *Match*(A,B).

2 Matching in 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 adjacent 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 [31, 32] 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 [25]. When a weight is associated with each edge, and we seek a perfect matching for which 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 [30].

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 $d(a,b)$—the distance between $a \in A$ and $b \in B$. The bottleneck matching is, therefore, the matching $M \subseteq E$ that minimizes $\max_{(a,b)\in M} d(a,b)$.

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 $d(a,b) \leq r$. 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, $(v_{2i}, v_{2i+1}) \in M$ and $(v_{2i-1}, v_{2i}) \in E \setminus M$ $(i = 1, ..., t)$. 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 [9] 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] found alternating paths by order of increasing length. Instead of finding the alternating paths one by one, Hopcroft and Karp [25][1] and

[1] A similar algorithm appeared also in [28]

Dinitz [16] find all shortest alternating paths together. We follow Dinitz's terminology.

To find all shortest alternating 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 layer 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 found a maximal set of edge-disjoint alternating paths by conducting a depth-first search of the layer graph. His algorithm requires $O(|E|)$ time to construct the layer graph and to find the alternating paths. For sufficiently large values of r, $G[r]$ contains $\theta(n^2)$ edges, hence his algorithm requires $O(n^2)$ time per layer 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 enables us to find the alternating 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. The data-structure supports the following operations:

- **neighbor**$_r(S, q)$: 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 **neighbor**$_r(S, q) = \emptyset$.
- **delete**$_r(S, s)$: Delete the object s from S.

The exact implementation of $\mathcal{D}_r(\cdot)$ depends on the dimension, the objects of S and the underlying norm. Various implementations will be described in Section 3.3. 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 therefore does not influence the overall complexity.

Let us turn now to the algorithm. 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 layer graph is constructed by the following procedure:

```
procedure ConstructLayerGraph($G, M$)
$L_1 \leftarrow$ exposed vertices of $A$ ;
$i \leftarrow 1$ ; $\mathcal{D} \leftarrow \mathcal{D}_r(B)$;
Repeat forever
  $L_{2i} \leftarrow \emptyset$;
  For each $a \in L_{2i-1}$ Do
    While neighbor$_r(\mathcal{D}, a) \neq \emptyset$
      $b \leftarrow$ neighbor$_r(\mathcal{D}, a)$ ;
          /* Find all $b$'s which are neighbors
             of some $a \in L_{2i-1}$ in $G[r]$ */
      Add $b$ to $L_{2i}$ ;
      delete$_r(\mathcal{D}, b)$;   /* 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 $\mathcal{L}$ is complete;
        Output $\mathcal{L}$ ;
  Else $L_{2i+1} \leftarrow$  all vertices of $A$ adjacent
                   to $L_{2i}$ via edges of $M$.
  $i \leftarrow i + 1$ ;
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 **neighbor**$_r(\mathcal{D}, a)$ 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 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. □

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 from a^+ (b is not exposed since all exposed vertices of $B \cap \mathcal{L}$ 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 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 continue to the next phase. 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 layer graph is again $O(n \cdot T(n))$. By a theorem of Hopcroft and Karp [25], 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. Match(A, B) can be found in time $O(n^{1.5} \cdot T(n))$, where $T(|S|)$ is the time required to perform an operation on $\mathcal{D}_r(S)$.*

3.3 Implementing $\mathcal{D}_r(\cdot)$

The implementation of $\mathcal{D}_r(S)$ depends on the setting: The bounds in this subsection are in the amortized sense, but this does not effect the time bound of the algorithm.

• $A \subseteq \mathbb{R}^2$ is a set of points, and $B \subseteq \mathbb{R}^2$ is one of the following:

(i) B is a set of n disjoint objects, and the underlying norm is L_p, for some $1 \leq p \leq \infty$. The distance from a point $q \in \mathbb{R}^2$ to a segment $b \in B$ is the distance from q to its closest point in b.

(ii) B is a set of n 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 $d(q, b_i) = w_i + ||q - b_i||_2$.

These two cases are handled similarly. For implementing the data-structure $\mathcal{D}_r(B)$ we use the dynamic nearest-neighbor scheme of Agarwal, Efrat and Sharir [2][2], who presented such data structures for both these problems. These data structures enable us to find (efficiently) the closest object of B to the query q, and hence to implement $\mathbf{neighbor}_r(B, q)$, by checking if its distance to q is at most r. These data structures support deletions of objects as well, hence $T(n) = O(n^\varepsilon)$, for any $\varepsilon > 0$.

• A and B are planar point-sets, and the underlying norm is the Euclidean norm. (The same data structure may also be used for L_∞.) Here we maintain a set $\mathbb{S}$ of disks of radius r, centered at the points of S, and $\mathbf{neighbor}_r(S, q)$ is answered by checking if any such disk contains q.

This is done as follows: We divide the plane into a grid Γ of square cells of size r. Let c be such a cell, and let $\mathbb{S}_c = \{s \in \mathbb{S} : s \cap c \neq \emptyset\}$. We store the cells c for which $\mathbb{S}_c$ is not empty in a search tree $\mathcal{T}$, so that given a query point, finding the cell containing it is done in $O(\log n)$. Let $\mathbb{S}_c^i \subseteq \mathbb{S}_c$ consist of those disks of $\mathbb{S}_c$ whose centers inside c. A disk s *lies below* a cell c if the y-coordinate of the center of s is less than the y-coordinate of every point of c. Let $\mathbb{S}_c^b \subseteq \mathbb{S}_c$ denote the set of disks of $\mathbb{S}_c$ below c. The relations *above, left, right* and the sets $\mathbb{S}_c^a$, $\mathbb{S}_c^l$, $\mathbb{S}_c^r$ are defined similarly. Obviously, $\mathbb{S}_c$ equals the disjoint union $\mathbb{S}_c^i \cup \mathbb{S}_c^b \cup \mathbb{S}_c^a \cup \mathbb{S}_c^l \cup \mathbb{S}_c^r$. Note that the total size of all these sets, taken over all cells of Γ is $O(n)$.

Let c be the cell containing the query point q. If $\mathbb{S}_c^i \neq \emptyset$, then $\mathbf{neighbor}_r(S, q)$ returns some disk of $\mathbb{S}_c^i$. Let us explain how to find if any disk of $\mathbb{S}_c^b$ contains q. The other cases are deals similarly. This is done by a data structure reminiscent of the dynamic convex hull of Hershberger and Suri [22]. We construct a binary tree Ψ whose leaves are the elements of $\mathbb{S}_c^b$ sorted by the x-coordinate of their centers, and each internal node ν is associated with the set Ψ_ν of disks in ν's subtree. Each node ν maintains (implicitly) $\mathcal{U}_\nu$, the upper envelope of P_ν. A crucial observation is that for each ν, $\mathcal{U}_{left\_son(\nu)}$ and $\mathcal{U}_{right\_son(\nu)}$ intersect at most once. The coordinates of this point are stored in ν.

In the full version of the paper we explain how the tree Ψ enables us to find whether q is below $U_{root(\Psi)}$ (which happens if and only if q is contained in some disk of $\mathbb{S}_c^b$) in time $O(\log n)$. We also show how to reconstruct $U_{root(\Psi)}$ after the deletion of a disk from $\mathbb{S}_c^b$

[2]see also the full version of [2], in preparation.

in (amortized) time $O(\log n)$. Hence $T(n) = O(\log n)$.

• A and B are sets of n points in $\mathbb{R}^d$, for fixed d, and the underlying norm is L_∞. As in the previous case, we maintain a set of d-dimensional cubes of size $2r$, centered at the points of S. $\mathcal{D}_r(S)$ consists of $d-2$-levels interval-trees (on the projection of the cubes on the first $d-2$ axis), and the two-dimensional data structure of the previous case, built on the projection of the cubes on the last two axis. Hence $T(n) = O(\log^{d-1} n)$. (This is a reminiscent of the orthogonal range trees described in [35]).

4 Computing *Match*(A, B)

In this section we show how we use the oracle of Theorem 3.2 to find r^*—the minimal r for which a perfect matching exists. We first note that r^* is a distance between an object in A and an object in B. Thus, we have a set of n^2 distances, called *critical distances*, on which we can perform binary search. In order to satisfy our time bounds, we do not produce the critical distances explicitly. The techniques by which we generate these values depends on whether we deal with points and/or segments in $\mathbb{R}^2$, or with points in higher dimension $\mathbb{R}^d$.

4.1 The 2-dimensional case

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. Let $A \subseteq \mathbb{R}^2$ be a set of n points and $B \subseteq \mathbb{R}^2$ a set of n objects. For $a_i \in A, b_j \in B$ let d_{ij} denote the distance from a_i to b_j. The *k'th distance selection problem* is to find $d^{(k)}$, the k'th largest value in the sequence d_{ij}, when k is a given parameter.

Lemma 4.1 *(Katz 95 [29]) Let $A, B \subseteq \mathbb{R}^2$ be sets of n points, and $1 \le k \le n^2$ an integer. Then $d^{(k)}$ can be found in time $O(n^{4/3}\log^2 n)$.*

Lemma 4.2 *Let $A \subseteq \mathbb{R}^2$ be a set of n points, and $B \subseteq \mathbb{R}^2$ of n disjoint convex objects of bounded complexity, or a set of n points, and L_p is the underlying norm. for $1 \le p \le \infty$. Let $1 \le k \le n^2$. Then $d^{(k)}$ can be found in time $O(n^{1.5}\log^3 n)$.*

Proof Omitted.

Let us describe the usefulness of Lemmas 4.1 and 4.2. Since r^* is some $d^{(j)}$, we conduct a binary search on $k \in \{1, \ldots, n^2\}$: we use the k'th distance selection algorithm to find $d^{(k)}$ and then use the oracle to decide whether $d^{(k)}$ is too large or too small, and decrease or increase k accordingly. The oracle of Theorem 3.2 is consulted $O(\log n)$ times.

4.2 Arbitrary Dimension

Unfortunately, for $d \ge 3$ we know how to efficiently tackle the matching problem in d-space only when A and B are point-sets, and either in an approximated fashion, which will be described later in Section 5, or when the underlying norm is L_∞. 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_i. We use the oracle to perform a binary search among all such projections on each of the axes. Consider the first axis. Let $a_1, \ldots, a_n$ (resp. $b_1, \ldots, b_n$) be the projection of A (resp. B) on this axis in increasing order. Consider the matrix $D = (d_{ij})$ where $d_{ij} \equiv a_i - b_j$. Note that all rows and all columns of D are sorted. Frederickson and Johnson [21] have shown how to find a critical value in such an (implicitly stored) matrix using $O(\log n)$ calls to the oracle, spending extra time $O(n \log n)$ for each such call (which does not effect the asymptotic running time). Repeating this process for all d axes we have shown

Theorem 4.3 *Let A, B be sets of n points in $\mathbb{R}^d$ $(d \ge 2)$, with L_∞ as the underlying norm. Then after preprocessing of $O(n)$time, Match(A, B) can be found using $O(\log n)$ oracle calls*[1].

[1]: dlog n?

Remark 4.4: Naturally, for $d = 2$ and the L_∞ norm, we use this method instead of the one of Section 4.1 since it is faster and simpler.

4.3 Accelerating the algorithm.

The running time of the algorithm can be improved by a $\log n$ factor in the cases when the time for solving the distance selecting problem is not a barrier; That is, in the cases where A and B are point-sets. This improvement is achieved by combining the oracle phase and the generic part. We describe this idea for the more involved case of planar points with the Euclidean norm. The d-dimensional case (under the L_∞ norm) is handled similarily.

Recall that $d^{(i)}$ is the ith largest distance between $a \in A$ and $b \in B$. We maintain a lower bound, $d^{(\ell)}$ (initially $\ell = 1$), and an upper bound, $d^{(u)}$ (initially $u = n^2$), on the value of r^*. In Section 4.1 and 4.2 we conducted a binary search on the values $d^{(1)}, \ldots, d^{(n^2)}$. However, this introduces a log factor. The purpose of this subsection is to save this factor.

In the course of the algorithm, we maintain a maximum matching M of $G[d^{(\ell)}]$, and use it as an initial matching for $G[d^{(i)}]$, $(\ell < i < u)$. If $d^{(i)} < r^*$, 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 $G[d^{(i)}]$. Otherwise, we can 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 layer graph is constructed.

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

The key observation is that sometimes we can conclude that $G[d^{(i)}]$ does not contain a perfect matching, without even finding the maximum matching. For a partial matching $M \subseteq G$ let $\mathcal{L}(M, G)$ denote the layer graph constructed by the procedure *ConstructLayerGraph*(G, M) of Section 3.1 starting with the matching M. We denote by $|\mathcal{L}(M, G)|$ the number of layers in this graph.

Lemma 4.5 *Let M be a partial matching of G. If $|M| < n - \sqrt{n}$ and $|\mathcal{L}(M, G)| > \sqrt{n}$ then G does not contain a perfect matching.*

Proof: Let M' be a maximum matching of G. $M \oplus M'$ consists of $|M'| - |M|$ vertex disjoint augmenting paths (and some alternating cycles). The length of each augmenting path is at least $|\mathcal{L}(M, G)|$. Since $|\mathcal{L}(M, G)| > \sqrt{n}$, there can be at most $n/|\mathcal{L}(M, G| < \sqrt{n}$ augmenting paths. Hence $|M'| < |M| + \sqrt{n} \leq n$. $\square$

```
$\ell \leftarrow 1$;  $u \leftarrow n^2$;  $M^1 \leftarrow$ empty matching;
While $\ell + 1 < u$ Do
  $step \leftarrow \lceil (u - \ell + 1)/n^{1/7} \rceil$ ;  $i \leftarrow \ell + step$;
  While $i < u$ Do
    Use the distance selection algorithm to find $d^{(i)}$
    $M \leftarrow M^{\ell}$;
    $\mathcal{L} \leftarrow ConstructLayerGraph(M, G[d^{(i)}])$ ;
    While ($\mathcal{L}$ contains exposed vertices of $B$)
      and ( $|\mathcal{L}| \leq \sqrt{n}$  or  $|M| \geq n - \sqrt{n}$ ) Do
        Update $M$ by the procedure of Section 3.2;
        $\mathcal{L} \leftarrow ConstructLayerGraph(M, G[d^{(i)}])$ ;
    End
    If $|M| = n$  Then $u \leftarrow i$ ;
    Else $\ell \leftarrow i$;  $M^{\ell} \leftarrow M$;  $i \leftarrow i + step$ ;
  End
End
```

Lemma 4.6 *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 initally, $u - \ell + 1 = n^2$, 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 (in the planar Euclidean case) time $O(n^{1.5} \log n)$. Note first that we solve the distance selection problem $n^{1/7}$ times. Since this problem can be solved in time $O(n^{4/3} \log^2 n)$ [29], this sums up to time $O(n^{31/21} \log^3 n) = O(n^{1.5})$. The number of times the layer graph is constructed is bounded by the number of times that its construction procedure terminates successfully—($O(\sqrt{n})$, by [25], as described in Section 3.2) and the number 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 to the time spent when the layer graph is of depth smaller than $\sqrt{n}$, and the time when the layer 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 cases there are $O(\sqrt{n})$ paths. Note that there are only 14 phases.

Note again that the very same idea can be implemented for the L_∞ norm, since, using Fredrickson and Johnson algorithm, we are able to find $r^{(i)}$ in time $O(n \log n)$ [21, Thm. ??]. [2]: ?????? ?

4.4 Summary of results

Theorem 4.7 • *Let A and B be two sets of n points in $\mathbb{R}^2$ with L_2 as the underlying norm. Then Match(A, B) can be found in time $O(n^{1.5} \log n)$.*

- *Let $A \subseteq \mathbb{R}^2$ be a set of n points and $B \subseteq \mathbb{R}^2$ a set of n disjoint objects, and the underlying norm is L_p, for some $1 \leq p \leq \infty$. The distance from a point $q \in \mathbb{R}^2$ to a segment $b \in B$ is the distance from q to its closest point in b. Then Match(A, B) can be found in time $O(n^{1.5+\varepsilon})$, for every $\varepsilon > 0$.*

- *Let $A \subseteq \mathbb{R}^2$ be a set of n points and B is a set of n points, and each $b_i \in B$ is associated with a nonnegative weight w_i, so that for a point $q \in \mathbb{R}^2$, we have $d(q, b_i) = w_i + ||q - b_i||_2$. Then Match$(A, B)$ can be found in time $O(n^{1.5+\varepsilon})$, for every $\varepsilon > 0$.*

- *Let A and B be two sets of points in $\mathbb{R}^d$, with L_∞ as the underlying norm. Then Match(A, B) can be found in time $O(n^{1.5} \log^{d-1} n)$.*

5 Approximating *Match*(A, B)

In this section we show an approximation scheme for r^*. To simplify the notation we restrict the discussion to the Euclidean norm. However the same construction works for all L_p.

Theorem 5.1 *Let A and B be sets of n points in $\mathbb{R}^d$, and let $\varepsilon > 0$ be a parameter. Let $r^* \equiv$ Match(A, B) when L_2 is the underlying norm. We can find in time $O(C(\varepsilon, d) \cdot n^{1.5} \log n)$ a Matching M^{app} between A and B satisfying $r^* \leq r^{\text{app}} \leq r^*(1 + \varepsilon)$, when r^{app} is the distance of the furthest matched pair in M^{app}, and $C(\varepsilon, d)$ depends only on ε and d.*

Arya et al. [8] described a data structure for a set of points $S \subseteq \mathbb{R}^d$, that can report in time $O(d(1+1/\varepsilon)^d \log n) = O(\log n)$ an approximated nearest-neighbor of a query point $q \in \mathbb{R}^2$. 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, and ε a pre-determined parameter. The construction takes time $O(n \log n)$. This data structure can also be dynamized so that a deletion takes time $O(\log n)$ [34]. 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 s (the approximated nearest neighbor) and if $||s-q||_2 \leq r \cdot (1+\varepsilon)$, we report that $\mathbf{neighbor}_r(\mathcal{D}_r(B), q)$ is s. 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 procedures as the *approximating oracle.*

Lemma 5.2 *If the approximating oracle finds a perfect matching, then $r(1+\varepsilon) \geq r^*$. Otherwise, $r < r^*$.*

Proof: Sketch: 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 get a graph $G^A \supseteq G[r]$. instead of the graph $G[r]$ G^A contains all the edges of length $\leq r$ and some of the edges of length 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 $\leq (1+\varepsilon)r$, the maximum edge in the matching is bounded by $(1+\varepsilon)r$. □

We find R, an upper bound of r^* such that $R \leq Cr^*$, for a constant C which depends only on the dimension and on ε. Let $r^*_\infty \equiv \mathit{Match}(A, B)$ when using the L_∞-norm. Let us turn for a moment to approximating r^*_∞.

Note that Arya et al.'s data structure [8] can also be used for L_∞ norm. We use it for deducing an approximating oracle for the L_∞ norm. Using this approximating oracle, combined with the procedure of Frederickson and Johnson [21], as described in Section 4.2, and with the improvement described in Section 4.3, we can find in time $O(n^{1.5} \log n)$ a distance r^{app}_∞ for which (by Lemma 5.2) $r^*_\infty \leq r^{\text{app}}_\infty \leq (1+\varepsilon)r^*_\infty$.
Note that $r^*_\infty \leq r^* \leq (\sqrt{d}) \cdot r^*_\infty$. Since a ball of radius $\sqrt{d}r$ fully contains a cube of size $2r$. Substituting, we get

$$r^* \leq (\sqrt{d}) \cdot r^{\text{app}}_\infty \leq (1+\varepsilon)(\sqrt{d}) \cdot r^*_\infty \leq (1+\varepsilon)(\sqrt{d}) \cdot r^* .$$

Let $R = (\sqrt{d}) \cdot r^{\text{app}}_\infty$ and $C = (1+\varepsilon)(\sqrt{d})$. The last equation yields $r^* \leq R \leq Cr^*$ as desired.

Let us divide the interval $[0, R]$ into $4/\varepsilon$ equal subintervals, and let $r_1 < r_2 < \ldots < r_m$ denote their endpoints, for $m = \lceil 4/\varepsilon \rceil$. Use the approximated oracle to perform a binary search among these values, and let r^{app} be the smallest r_i for which the oracle succeeds in finding a perfect matching. As easily observed, $r^* \leq r^{\text{app}} \leq r^*(1+\varepsilon)$, which establishes the proof of Theorem 5.1.

6 The Translation Problem

Let A and B be two sets of n points in $\mathbb{R}^2$. For a translation $\tau \in \mathbb{R}^2$ let $\tau + B$ 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 $\rho^* \equiv \mathit{Match}(A, \tau^* + B)$.

Theorem 6.1 *Given A, B as above, the translation problem can be solved in time $O(n^5 \log^2 n)$.*

Let ρ be a fixed parameter, and let us obtain an oracle that determines if there is a translation τ for which $\mathit{Match}(A, \tau + B) \leq \rho$. This problem was previously investigated by Alt et al. [6] who showed how to solve this problem in time $O(n^6)$. We use our technique to improve the running time of their algorithm to $O(n^5 \log n)$.

Let us briefly describe their algorithm, and refer the reader to their paper for 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 out attention to translations τ which bring some point of A to distance ρ (exactly) of some point $b \in B$.

For $a \in A$ let a^ρ denote the disk of radius ρ (under the underlying norm) centered at a, and let A^ρ denote the union of all disks a^ρ for all $a \in A$. For $a \in A, b \in B$, let C^ρ_{ab} denote all rigid continuous sets of translations that bring a to distance ρ from b.

The algorithm checks for each pair $a \in A$, $b \in B$ if $\mathit{Match}(A, \tau + B) \leq \rho$ for some translation $\tau \in C^\rho_{ab}$. 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 of C^ρ_{ab}. We first evaluate $\mathit{Match}(A, \tau_0 + B)$. If its value is less than or equal to ρ then we are done. Otherwise, we translate B rigidly by all translations of C^ρ_{ab}. During this process, points of B are moved into, or out of disks of A^ρ, implying that edges are inserted into or deleted from the graph $G_\tau[\rho]$. We call such events *critical events.* Because of the convexity, each edge is inserted and deleted at most once.

Hence C^{ρ}_{ab} contains at most n^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 an insertion, the matching increases by at most one augmenting path (containing the new edge). If an edge of the matching is deleted, 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 layer graph.

Alt et al. [6] 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 $O(n^4) \times O(n^2) = O(n^6)$.

We suggest to use the procedure for constructing the layer graph of Section 3.1. This procedure requires only $O(n \log n)$ time for each path. Taken over all $O(n^4)$ critical events, the total time sums to $O(n^5 \log n)$.

Finding ρ^*: In order to find ρ^*, the smallest ρ for which such a translation exists, we use the parametric searching technique of Megiddo [33]. Again, we assume familiarity of the reader with this technique, and refer to [19] for a similar application of this technique. We use the procedure described above as an oracle. However, in contrast to the "traditional" parametric searching technique, we avoid inducing a parallel version of this procedure for a generic algorithm. Instead we show that a simple parallel sorting algorithm will do. Let us describe how we generating critical values of ρ. Let $a \in A, b \in B$. Consider the translations of C_{ab} as defined above, but for the unknown value ρ. Consider the sequence of graphs determined when B is translated along C_{ab}. While changing ρ, this sequence (combinatorially) changes at a set of *critical radii* ρ. Each such critical value is determined for one of the following events, for some $a', a'' \in A$ and $b', b'' \in B$ (not necessarily distinct):

(i) The smallest ρ for which $\tau + b'$ intersects the boundary of a'^{ρ} (for some τ).

(ii) A value of ρ for which there exists τ_0, such that the boundary of a'^{ρ} is intersected by $\tau_0 + b'$, and the boundary of a''^{ρ} is intersected by $\tau_0 + b''$.

Note that ρ^* must be such an event. Note also that the number of events of the second type is $\Omega(n^6)$ in the worst case, hence we cannot generate them all. However, to obtain the combinatorial structure of the graph at ρ^*, it suffices to first to find (in $O(1)$ parallel steps) all events of the first type, and then to sort for each pair a, b the critical values $\tau^{\rho}_{a,b,a',b'}$, (among all values of τ) defined as the translations τ for which $\tau^{\rho}_{a,b,a',b'} + b$ is at distance ρ from a and $\tau^{\rho}_{a,b,a',b'} + b'$ is of distance ρ from a'. We can sort these τ in parallel, using for example, the sorting network of Ajtai et al. [4]. Implementing sequentially this network combining with the acceleration scheme of Cole [15], yields an algorithm that finds ρ^* in time $O(n^4 \log n)$, using $O(\log n)$ calls to the oracle. Hence the total time for finding ρ is $O(n^5 \log^2 n)$ time, thus proving Theorem 6.1.

Approximating the optimal translation. Note that while finding a translation τ^* which minimizes $d(A,\, \tau+B)$ is a non-trivial problem for which only high degree polynomial algorithms are known, it is easy to find translation τ^{app} that brings $d(A,\, \tau+B)$ within a factor of 2 of the optimum—that is $d(A, \tau^{\text{app}} + B) \leq 2\, d(A, \tau^* + B)$. This translation τ^{app} is determined by translating B so that the lower-left corner of its axis-parallel smallest-enclosing rectangle coincides with the lower-left corner of the axis-parallel smallest-enclosing rectangle of A. The proof that the last equation holds follows from the same arguments as in [5], and is easily established. Moreover, if we care for a better approximation to $\rho^* = \mathit{Match}(A, \tau^* + B)$, we use the following approach, borrowing some ideas from [20]. Let $0 < \varepsilon < 1$ be a fixed parameter. Let $\rho^I \equiv d(A, \tau^{\text{app}} + B)$. Surely, τ^* is inside a cube c of size $2\rho^I$ centered at τ^{app}. Consider the grid Γ defined on c, such that each cell of Γ is a cube of size $2\rho^I/\varepsilon$ (so Γ consists of $\lceil\varepsilon\rceil^{-d}$ cells). For each translation τ represented as a vertex of Γ, we approximately evaluate $\mathit{Match}(A,\, \tau+B)$ using the procedure of Theorem 5.1, and choose τ', the best one. The total time of this process is $O(\varepsilon^{-d}(1 + 1/\varepsilon)^d n^{1.5} \log n \log \varepsilon^{-1})$ (since the time of each operation on the approximated data structure is $O((1+1/\varepsilon)^d \log n)$). It can be shown that $\mathit{Match}(A, \tau' + B)$ approximates ρ^* up to a factor of $1 + \varepsilon$.

7 Related Problems

Several related problems are easily tackled by our method.

Partial matching: Let $A, B \subseteq \mathbb{R}^2$ be point-sets (not necessarily with the same cardinality), and let $1 \leq k \leq \min\{|A|, |B|\}$ be an integer. The problem is to find r^*, the smallest r for which a matching of cardinality k exists in $G[r]$. This problem might arise in pattern matching, when we suspect that some of the points are superfluous, and/or that we seek the appearance of a relatively small pattern A inside a large picture B. Using almost the very same ideas that led us to Theorem 4.7, we can solve this problem in $O(|B| \log |B| + |A|^{1.5} \log |B|)$.

For $k < n/2$ the following simple algorithm finds in time ? a matching M^k of size k and whose longest matched edge is at most r^* The algorithm repeated selects the closest pair (a, b) and removes it from the graph. We stop after matching k such pairs. Let r^k denote the distance between the last matched pair. It can be shown that if $k \leq n$ then $r^k \leq r^*$: If $r^k > r^*$ then $G[r^k] \supseteq G[r^*]$, and therefore, $G[r^k]$ contains a perfect matching M. Since each edge of M^k is adjacent to at most two edges of M, $n/2 > k = |M^k| \geq \frac{1}{2}|M| = n/2$.)

Furthermore, if $k = (1-\alpha)n$, then arguments similar to Lemma 4.5 show that the depth of the layer graph is at most $1/\alpha$. Thus for fixed α we need only a constant number of layer graphs, i.e., the algorithm takes time $O(n \log n)$.

Finding the longest perfect matching. Let us describe briefly another set of problems. Let A and B be two sets of n points, 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 $\mathcal{D}_r(B)$ for finding a point of B whose distance from a query point q is at least r, and to delete a point from B. Fortunately, these operations are obtained efficiently in the planar case by maintaining the *Circular Hull* of B — namely the region consists of the intersection of all disks of radius r containing B. Hershberger and Suri [23] 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 $O(n^{1.5} \log n)$.

In our opinion, it is interesting to note that while we failed to find efficient data-structure for implementing $\mathcal{D}_r(\cdot)$ in order to find $Match(A, B)$ in $\mathbb{R}^3$, (i.e. with running time $o(\sqrt{n})$), obtaining such data structure for finding $\overline{Match}(A, B)$ in $\mathbb{R}^3$ is rather easy. Let $\mathcal{B}_r(S)$ denote the set of three-dimensional balls of radius r, centered at the points of S. Note that if some point $s \in S$ is at distance larger than r from a query point q, then $q \notin \cap\mathcal{B}_r(S)$. Agarwal et al. [2] suggested 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 $\cap\mathcal{B}_r(S)$, and to delete a point from S, in time $O(n^\varepsilon)$. To use this structure, we build a binary balanced tree $\mathcal{T}$, whose leaves are the points of S, and each internal node v is associated with S_v, the points associated with the leaves of v's subtree. We also associate with v the data structure $\Xi_v \equiv \Xi(S_v)$. To perform $\mathbf{neighbor}_r(S, q)$ (that is, to find $s \in S$ whose distance from q is larger than r), we use Ξ_v when $v = root(\mathcal{T})$, to find if $q \notin \cap\mathcal{B}(S_v)$, and if so, we recursively check each of its two children to find (at least) one v', such that $q \notin \cap\mathcal{B}(S_{v'})$. We repeat this process until v' is a leaf, where in this case we return the singleton S_v. Deletion is carried out a trivial fashion. Note that both these operations are done in time $O(n^\varepsilon)$, hence when A, B are point-sets in $\mathbb{R}^3$, $\overline{Match}(A, B)$ can be found in time $O(n^{1.5+\varepsilon})$ for any $\varepsilon > 0$.

Acknowledgment.

Discussions with Pankaj K. Agarwal, Esther M. Arkin, Yefim Dinitz, Matya Katz 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 11 Annual Symposium on Computational Geometry*, 1995 39–50.

[3] P. K. Agarwal and J. Matoušek, Ray shooting and parametric search, *SIAM J. Comput.* 22 (1993), 794–806.

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

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

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

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

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

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

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

[11] M.S. Chang, C.Y. Tang and C.T. Lee, Solving the Euclidean Bottleneck Matching Problem by k-Relative Neighborhood Graphs, *Algorithmica* 8 (1992) 177–194.

[12] L.P. Chew and K. Kedem, Improvements on Geometric Pattern Matching Problems, *Scand. Workshop Algorithm Theory — (SWAT), Lecture Notes in Computer Science,* 1992, vol. 621, (O. Nurmi and E. Ukkonen ed.) Springer-Verlag, New York–Berlin–Heidelberg, 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), Lecture Notes in Computer Science* 1995, vol. 979 (P. Spirakis ed.) Springer-Verlag, New York–Berlin–Heidelberg, 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 5 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, M. Sharir and A. Ziv, Computing the smallest k-enclosing circle and related problems. *Computational Geometry: Theory and Applications* 4 (1995), 119–136.

[20] A. Efrat, Finding Approximate Matching of Points under Translation, Manuscript, 1995.

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

[22] J. Hershberger and S. Suri, Applications of a semi-dynamic convex hull algorithm, *Procceedings 2 Scandawian Workshop on Algorithms Theory Lecture Notes in Computer Science* 1990, vol. 447, Springer-Verlag, New York–Berlin–Heidelberg, 380–392.

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

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

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

[26] 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.

[27] H. Imai and Ta. Asano, Efficient algorithms for geometric graph search problems, *SIAM J. Comput.* 15 (1986), 478–494.

[28] 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).

[29] M. Katz, Improved Algorithms in Geometric Optimization via Expanders, *Proc. 4nd Israel Sympos. Theory Computing and Systems*, 1995, 78-87.

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

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

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

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

[34] D. M. Mount, Private communication.

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