Chapter 2

Testing the congruence of point sets in $\mathbb{R}^d$

The simplest point pattern matching problem is to decide whether two patterns are actually the same, up to congruence.

Problem 2.1 (Congruence testing problem for d-dimensional point sets).
Given *two point sets $P, Q \subseteq \mathbb{R}^d$ of n points each.*
Decide, *whether there exists a rigid motion μ such that $\mu(P) = Q$.*

This problem is well-understood in dimensions at most three, where there are good algorithms which reach the asymptotic lower bound of $\Omega(n \log n)$ [18, 21, 22, 37, 39, 48]. For higher dimensions, however, the situation is different: only dimension reduction methods are known, which lead to an $\mathcal{O}(n^{d-2} \log n)$ algorithm by Alt et al. [18], a Monte-Carlo $\mathcal{O}(n^{\lfloor d/2 \rfloor} \log n)$ algorithm by Akutsu [7], an unpublished deterministic algorithm of the same complexity by Matoušek (mentioned in [7]), and the $\mathcal{O}(n^{\lceil d/3 \rceil} \log n)$ algorithm described below (c.f., Theorem 2.2 on page 13).

2.1 The dimension reduction technique

A congruence that maps P to Q consists of two parts: a translation and a rotation. The translational part is easy to determine, since the image of the centroid $c(P)$ of P under that congruence has to be the centroid $c(Q)$ of Q, and the centroid can be determined fast, i.e., in $\mathcal{O}(n)$ time. So each algorithm preprocesses the sets by translating P to $P - c(P)$ and Q to $Q - c(Q)$, and searches for a rotation around 0 that maps one set on the other. Also the distances of the points to the centroid have to be preserved, so we can replace each point by a point on the unit sphere which carries the distance (or an ordered list of distances) of the point (or points) in that direction as a label. So in the following we have two sets P', Q', each consisting of n labeled points (counting multiplicities) on the unit sphere with center 0, and we are looking for a label-preserving rotation that maps P' on Q'.

In the two-dimensional case it is easy to reduce this problem to the classical substring matching problem: The points of P' are cyclically ordered on the circle, and described by the pair of their label and the angle to the next point on the circle (which together is just some symbol in an alphabet). So starting at an arbitrary point and going around the circle once, we can encode P' as a string in that alphabet; and going around the circle twice for Q', we encode that set in another string. Then P and Q are congruent if and only if the string of P' is a substring of the string for Q', which can be tested in $\mathcal{O}(n \log n)$ time.

The algorithm of Alt et al. [18] for the three-dimensional case (similar also Sugihara [48]) involves again the creation of a combinatorial model and solution of the problem on that: Each of the sets consists of n labeled points on the unit sphere, so their convex hull is some polyhedron, which we can describe by its edge graph augmented by edge labels carrying the length of the edge and the angular distance to the next edge around that vertex (i.e., the interior angle of the face at that vertex). This information completely specifies the polyhedron, since, according to Cauchy's rigidity theorem, once the edge lengths and interior angles of each face are prescribed, there is at most one convex realization of a polyhedron. But the isomorphism of (labeled) polyhedral graphs (threeconnected and planar) can be tested in $\mathcal{O}(n \log n)$ time.

For higher dimensions no such direct method is known, but it is possible to reduce one higher-dimensional problem to a number of alternative lower-dimensional ones. The underlying idea is that if we know the correct image point $\mathbf{y} \in Q'$ for some point $\mathbf{x} \in P'$, then the subspace through $\mathbf{x}$ is mapped on the subspace through $\mathbf{y}$ and the same holds for their orthogonal complements. Thus we can orthogonally decompose each point of P' and each point of Q' into a pair of points (one on a line, one on the orthogonal hyperplane), and these have to be mapped on each other by the congruence. But the point on the line is uniquely identified by its signed distance to 0 ($\mathbf{x}$, $\mathbf{y}$ positive), so we can just append this number to the label of the point on the hyperplane (additionally to all previous labels of the original point), and have reduced the original problem to a problem of points with longer labels which lie in a hyperplane. Thus if we know the correct images of k linearly independent points, we can reduce the dimension of the space by k, appending k additional labels (coordinates) to each point.

If we do not know the correct image of any point, the simplest way is to take a fixed $\mathbf{x} \in P'$ and try each of the potential image points $\mathbf{y} \in Q'$ in turn: if we are successful in one of the lower-dimensional problems, we can extend the solution to a solution of the original problem by removing that additional label and adding $\mathbf{x}$ (or $\mathbf{y}$) times that label to each point; if we are not successful for any choice of $\mathbf{y}$, then the sets P and Q are not congruent. This is the algorithm of Alt et al. [18] which runs in time $\mathcal{O}(n^{d-2} \log n)$, reducing one d-dimensional problem of n points to n alternative $(d-1)$-dimensional problems in each step.

If we use the fact that any closest pair in P' has to be mapped on a closest pair in Q', and that the number of closest pairs is only linear in n (for fixed dimension d: since the degree of the graph of closest pairs is bounded by a constant, the 'kissing number' or 'Newton number' which grows exponentially in d [46, 51]), we can reduce the problem in each step by two dimensions to $\mathcal{O}(n)$ alternative $(d-2)$-dimensional problems. This is the

algorithm of Matoušek [7] which runs in $\mathcal{O}(n^{\lfloor d/2 \rfloor} \log n)$ time.

If we try to extend this approach to triples a difficulty arises which is caused by the inner symmetries of the point set: since the point-k-tuples which define the alternative reductions are themselves defined by metric properties, the set of all these alternative reductions will be closed under inner isometries of the point set. But for dimension $d \geq 4$ this set can be quite big, since rotation subgroups in orthogonal planes are possible. So, e.g., in dimension four the set consisting of two regular $(n/2)$-gons with common center 0, but in orthogonal planes, allows $(n^2/4)$ rotations, and any full-dimensional triple of points (that could be used to reduce the dimension by three) will generate an orbit of $\Omega(n^2)$ other possible reductions. Similar constructions work also in higher dimensions. This big number of alternative reductions is not really needed, since they all give the same result, but they have to be recognized.

2.2 A refined approach

The solution to this problem is to recognize whenever the point set lies in orthogonal subspaces (which can be matched independently), and use an extended version of the smallest distance graph in the other cases, in such a way that we get a dimension reduction of three with a linear number of alternatives in each step.

For this purpose we add to each point $\mathbf{x} \in P'$ the antipodal point $-\mathbf{x}$ labeled as 'new', if it is not already in the set P' (in the following always perform the same operations for Q'). Then the smallest distance occurring between two antipodal pairs $p, -p$ and $q, -q$ on the unit sphere is at most $\sqrt{2}$, and $\sqrt{2}$ is reached if and only if p is orthogonal to q. We can extend this from two antipodal pairs to larger point sets by constructing a graph with the extended set as vertices, which has in the beginning the antipodal pairs as connected components. Any graph containing this graph as a subgraph has the property that the smallest distance occuring between points of distinct connected components is at most $\sqrt{2}$, and reaches this value if and only if the connected components lie in orthogonal subspaces. So we start from the antipodal pairs graph, and extend in each step the graph by all those edges between points of distinct components that are of minimum length among all such point pairs. Then we either find after some steps a connected component which spans a subspace of dimension at least three, which allows a dimension reduction, or we find that all the connected components lie in orthogonal two-dimensional subspaces, which can be treated independently. Algorithm *Congruence_Test_* is shown on page 12. In the following, edges between two antipodal points will be called *antipodal edges*, whereas the other edges will be called *strong edges*; two points sharing such an edge will be called *strongly adjacent* or *strong neighbours*. Note that — for the sake of simplicity — we omitted several checks that are performed in the course of the algorithm, e.g., testing whether the two graphs have the same number of connected components all the time, etc. If any of these checks fail, the algorithm gives a negative answer.

In the following two sections we will prove, that the algorithm is correct and that it runs within the claimed time bounds.

Algorithm *Congruence_Test*(P, Q, d)
Input: Two sets P, Q, of n labeled points each, in $\mathbb{R}^d$.
Output: Decides whether there is a labelpreserving congruence that maps P to Q.

```
1.  ▷If the dimension is at most three, use one of the known O(n log n)-algorithms to decide the
    problem.
2.  ▷Preprocess P and Q to P' and Q' by moving the centroid to 0 and projecting each point
    on the unit sphere around 0, with the projection distance appended as an additional label.
3.  ▷Construct the extended sets P'' and Q'' by adding for each point x ∈ P' and each point
    y ∈ Q' the antipodal points −x, −y, labeled as 'new', if they were not already contained
    in the set. Construct the initial graphs for both sets by joining each point to its antipodal
    point.
4.  repeat
5.      if there is only one component left
6.        then (* c.f., Lemma 2.4 *)
7.             ▷Compute for P'' and Q'' the coordinates of the points in the two-dimensional
               linear subspaces spanned by the sets, and apply one of the known O(n log n)-
               algorithms to decide the two-dimensional problem.
8.        else
9.             ▷Determine the smallest distance δ_min between two points of distinct connected
               components.
10.            if δ_min = √2
11.              then (* c.f., Lemma 2.4 *)
12.                   ▷Decompose P'' and Q'' in their connected components, computing
                      the coordinates of the points in the two-dimensional linear subspaces
                      spanned by the components.
13.                   for each matching of the P''-components to the Q''-components
14.                       do
15.                          for each component pair (α, β) in the matching
16.                              do
17.                                ▷Apply one of the known O(n log n)-algorithms for the
                                   two-dimensional case to decide whether α and β are
                                   congruent.
18.                   if there is a matching such that each matched pair is congruent
19.                     then (* c.f., Observation 2.3 *)
20.                          P and Q are congruent
21.                     else
22.                          they are not congruent.
23.              else  (* δ_min < √2 *)
24.                   ▷Add all edges of length δ_min that join points in distinct connected
                      components to the graph.
25.                   if there is a triple T_P = (point, neighbour_1, neighbour_2) of P'' that
                      spans a linear subspace of dimension three
26.                     then
27.                          for all triples T_Q = (point, neighbour_1, neighbour_2) of Q'' that
                             span a linear subspace of dimension three
28.                              do
29.                                ▷ If T_P and T_Q are congruent, perform the dimen-
                                   sion reduction determined by T_P and T_Q, and call re-
                                   cursively Congruence_Test for the (d − 3)-dimensional
                                   problem.
30.                          if one of the potential image triples from Q'' gives congruent
                             sets
31.                             then
32.                                  P and Q are congruent
33.                             else
34.                                  they are not congruent.
35. until a reduction is found, or the set is decomposed in orthogonal subsets.
```

Theorem 2.2 (Correctness and complexity of algorithm *Congruence_Test*). *The algorithm Congruence_Test is correct and runs in* $\mathcal{O}(n^{\lceil d/3 \rceil} \log n)$ *time.*

2.2.1 Correctness

The correctness of the algorithm is obvious if $d \leq 3$; also the claimed time bound follows immediately in that case.

It was already observed that once we end up with a set of connected components of pairwise distance $\sqrt{2}$ we have decomposed the original problem into orthogonal subproblems that can be solved independently:

Observation 2.3. *Let* $P, Q \subset \mathbb{R}^d$ *be two finite* d*–dimensional point sets, and let* $U, V \subseteq \mathbb{R}^d$ *be two linear subspaces. Let* U^{ortho} *denote the orthogonal complement of* U*, and assume that*

1. $P = P_1 \cup P_2$, $P_1 \subseteq U$, $P_2 \subseteq U^{\text{ortho}}$,
2. $Q = Q_1 \cup Q_2$, $Q_1 \subseteq V$, $Q_2 \subseteq V^{\text{ortho}}$,
3. $P_1 = \kappa_1(Q_1)$ *for a congruence* κ_1 *with* $\kappa_1|_{V^{\text{ortho}}} = \mathsf{identity}$*, and*
4. $P_2 = \kappa_2(Q_2)$ *for a congruence* κ_2 *with* $\kappa_2|_V = \mathsf{identity}$.

Then $P = \kappa(Q)$ *for the congruence* $\kappa = \kappa_1 \circ \kappa_2$.

Next we will show that the following invariant holds throughout the algorithm, just before $\delta_{\min}$ is recomputed and edges are added (and components are merged) as long as no dimension reduction is found and $\delta_{\min}$ is smaller than $\sqrt{2}$:

Lemma 2.4 (Invariant of algorithm *Congruence_Test*). *During the execution of algorithm Congruence_Test the following invariant holds at the start of the repeat-loop:* Each connected component consists of points distributed on a great circle of the unit sphere, and therefore is planar.

Proof. We proceed by induction: In the beginning the invariant obviously does hold. Now assume that $\delta_{\min}$ is computed, $\delta_{\min} < \sqrt{2}$ and we add all minimum length edges between vertices in distinct connected components. We show that either a dimension reduction is found (i.e., the graph now contains three points spanning a three-dimensional subspace) or the invariant also holds after the merging step. To this end we look at two connected components c_1 and c_2 that are merged in the merging step. Note that due to the antipodal vertices, the process is symmetric, i.e., if vertex u becomes adjacent to vertex v in the merging process, then also the antipodal vertices u' and v' become adjacent in the same step. By induction there are two great circles C_1 and C_2 containing these two components. We distinguish the following two cases:

The two great circles C_1 and C_2 coincide. In that case the invariant clearly holds after the merging step.

The two great circles C_1 and C_2 differ. In that case these two circles share exactly two points. Let us assume that the merging process establishes a strong adjacency between $p_1 \in c_1$ and a vertex $p_2 \in c_2$.

If $\#c_1 = \#c_2 = 2$ then $\#(c_1 \cup c_2) = 4$ and so either the invariant holds after the merging step or the new graph contains three points spanning a three-dimensional subspace. Therefore we can assume in the following wlog that $\#c_1 \geq 4$, so p_1 has at least one strong neighbor $p_1' \in c_1$.

In case that $p_2 \in c_2 - C_1$ the new graph contains three points spanning a three-dimensional subspace. In case that or all the points in c_2 minimizing the distance to c_1 lie on C_1 we see that either

$\#c_2 = 2$, i.e., $c_2 = C_1 \cap C_2$ and so the invariant holds, or

$\#c_2 > 2$, i.e., p_2 has at least one strong neighbor $p_2 \in c_2 - C_1$ and therefore the new graph contains three points spanning a three-dimensional subspace.

□

The correctness of the algorithm immediately follows from our previous considerations: If we find a dimension reduction at some point, then the correctness follows by induction over the dimension, as was argued in the introduction. If we end up with a single connected component, the problem is actually only two-dimensional by Lemma 2.4, and if we end up with a set of connected components of pairwise distance $\sqrt{2}$ we have decomposed the original problem into two-dimensional orthogonal subproblems according to Lemma 2.4 that can be solved independently by Observation 2.3.

2.2.2 Analysis

To facilitate the analysis of the algorithm, we have to specify the implementation in some more detail. We store the edges of the complete graph on the point set (except the edges connecting antipodal vertices) in a list $\mathcal{L}$, sorted by their length. The initialization time for this list is $\mathcal{O}(n^2 \log n)$. Furthermore we use a union-find data structure $\mathcal{C}$ to store the points in each connected component. This structure is initialized with n sets, each containing two points (namely a point along with its antipodal point, i.e., $\mathcal{C} = \{C_p \mid p \in P\}$ and $C_p = \{p, -p\}$); this initial step takes $\mathcal{O}(n)$ time. Finally we keep the $n \times n$ adjacency matrix G of the modified smallest distance graph. It can be set up in $\mathcal{O}(n^2)$ time as $G_{p,q} = 1$ if $q = -p$ and $G_{p,q} = 0$ otherwise.

Throughout the algorithm for each point $p \in P$ the structure C_p will contain exactly those points that are in the same connected component as p, and $\mathcal{L}$ will contain all point pairs of distance at least $\delta_{\min}$, i.e., $(p, q) \in \mathcal{L} \iff ||p - q|| \geq \delta_{\min}$.

In order to determine the new minimal intercomponent distance in step 9, we consider the length $\delta_{\min}$ of the first element of $\mathcal{L}$. If $\delta_{\min} = \sqrt{2}$ we have decomposed the original problem into orthogonal subproblems and the data structures need not be updated anymore. Otherwise we extract all edges of length $\delta_{\min}$ from $\mathcal{L}$ and store them in a list $\mathcal{L}_{\min}$.

If l denotes the length of this list, the time for this step is bounded by $\mathcal{O}(l)$. For all edges $(p, q) \in \mathcal{L}_{\min}$ we first check whether $C_p = C_q$, i.e., if they join points in the same connected component *before edges are added*. If not, we make them adjacent in G. After that we update $\mathcal{C}$ by merging two different components in case we just added edges between them. The procedure we just described requires at most $2l$ find queries to $\mathcal{C}$ and at most l union operations on $\mathcal{C}$, so the overall time required is of order $\mathcal{O}(l \log^* n)$.

Now, although l can be as large as $\mathcal{O}(n)$ (see below), we see that the amortized cost for determining the minimal intercomponent distances in step 9 and maintaining the data structures $\mathcal{L}$ and $\mathcal{C}$ in step 24 is of order $\mathcal{O}(n^2 \log^* n)$, since each edge is 'touched' at most once in the course of the algorithm.

The task of projecting a set of points to the linear subspace it spans and computing a basis for that space, along with the appropriate coordinate computation can be done in linear time.

The parts of the algorithm that handle the case $d \leq 3$ (step 1), the case of all points lying on a plane (step 7) and the case of orthogonal subproblems (steps 12–22), respectively, are called at most once during the execution of the algorithm. Step 1 as well as step 7 takes $\mathcal{O}(n \log n)$ time. In steps 12–22 there are at most d components in each set, since the components are mutually orthogonal, so we have to try at most $d! = \mathcal{O}(1)$ possibilities for the matching, and for each matching we have to solve at most $d = \mathcal{O}(1)$ two-dimensional subproblems; the total time required for these steps is therefore $\mathcal{O}(n \log n)$, too.

The repeat loop is executed $\mathcal{O}(n^2)$ times and in each step $\mathcal{O}(n)$ edges are added (see below). But since each edge will be used only once in this process, the overall time spent for finding minimum length edges and merging components is bounded by $\mathcal{O}(n^2 \log^* n)$, as argued above.

The number of recursive calls of the algorithm in step 29 is bounded by the number of triples T_Q found in step 27. To prove the $\mathcal{O}(n^{\lceil d/3 \rceil} \log n)$ time bound, we have to show that not too many recursive calls are generated. To be more precise, we prove that steps 24–34 generate only $\mathcal{O}(n)$ possible reductions. For this we look at the last time that $\delta_{\min}$ was recomputed and edges were added. From Lemma 2.4 we know that each connected component was planar before the edges were added, i.e., the points of the component were distributed on a great circle of the unit sphere. Now consider a point p from a component c that is merged with the components $c_1, \ldots, c_k$ $(k \geq 1)$ in the current step. In each component c_i there is a set of points P_i such that $||p - p'|| = \delta_{\min}$ for all $p' \in P_i$ and $||p - p''|| > \delta_{\min}$ for all $p'' \in c_i - P_i$. Furthermore the distances between points from different components are *larger* than $\delta_{\min}$, i.e., for all $i \neq j$ and for all $p' \in P_i$, $p'' \in P_j$ we have that $||p' - p''|| > \delta_{\min}$. If we pick a point p_i from each P_i and place a sphere of radius $\delta_{\min}/2$ around each p_i we get an arrangement of k congruent mutually disjoint spheres touching the sphere of radius $\delta_{\min}/2$ around p. The largest number of spheres in such a packing is called the 'kissing number' or 'Newton number' k_d of dimension d, and is less than $3^d = \mathcal{O}(1)$. This implies that a vertex from a component can only have new neighbours in at most $k_d = \mathcal{O}(1)$ components. Furthermore it has at most two neighbours in each of these components, since a sphere around that point intersects the great circle containing the other connected component in at most two points, unless the center of the

sphere is in the subspace orthogonal to the plane spanned by the circle. But this is not possible, since $\delta_{\min} = \sqrt{2}$ in that case, which we excluded in steps 12–22. We see that at most $\mathcal{O}(n)$ edges were added in this merging step and that each vertex has degree at most $\mathcal{O}(1)$ and therefore the number of triples in the nearest neighbour graph is bounded by $\mathcal{O}(n)$.

The time $T(n, d)$ that algorithm *Congruence_Test* needs to decide the congruence of a d-dimensional n-point set, can be estimated as $T(n, d) \leq c \cdot n^2 \log n + c \cdot nT(n, d-3)$ for a sufficiently large constant $c > 0$, where the first term accounts for the time spent in the loop and for the parts of the algorithm that handle the initialization and the base cases. By iterating this inequality it is easily seen that

$$T(n, d) \leq kc^k n^{k+1} \log n + c^k n^k T(n, d-3k)$$

for all $k \geq 1$. For $k_0 = \lceil \frac{d-3}{3} \rceil$ we have that $d - 3k_0 \leq 3$ and thus $T(n, d-3k_0) \leq c \cdot n \log n$ (if c is sufficiently large), so that

$$T(n, d) \leq 2k_0 c^{k_0+1} n^{k_0+1} \log n.$$

We see that $T(n, d) = \mathcal{O}(n^{\lceil d/3 \rceil} \log n)$, which proves the claim. □