Congruence, Similarity and Symmetries of Geometric Objects[5)]

by

Helmut Alt[1)], Kurt Mehlhorn[2)],
Hubert Wagener[3)], Emo Welzl[4)]

A 02/87

1) FB Mathematik, FU Berlin
2) FB Informatik, Universität des Saarlandes
3) FB Informatik, TU Berlin
4) Institut für Informationsverarbeitung, TU Graz

Abstract: We consider the problem of computing geometric transformations (rotation, translation, reflexion) that map a point set A exactly or approximately into a point set B. We derive efficient algorithms for various cases (Euclidian or maximum metric , translation or rotation or general congruence).

5) Part of this research was supported by the DFG under grants Me 620/6-1 and Al 253/1-1.

1. Introduction

We consider the problem of deciding whether two geometric objects A, B in $\mathbf{R}^d$ are congruent or similar and, if they are, of finding the geometric transformation (rotation, translation, reflexion, stretch) that maps B into A.

Important practical applications are for example: determining how to move an object from one given position into another one (robotics), recognition of patterns and of written text, and determining speed and direction of moving objects out of pictures taken at different times.

The second problem we are considering is finding all the symmetries of a given geometric object, i.e., its symmetry group. Important applications are within pattern recognition and some areas of biology and crystallography.

We consider only the basic case of A and B consisting of n points in $\mathbf{R}^d$. In many cases the algorithms presented here can be extended to more general geometric objects, where the points are connected by lines, curves, or surfaces described, for example, by algebraic equations (cf. also [Ata]).

We assume a model of computation which is able to represent arbitrary real numbers and to perform all the geometric computations involved (determining angles, distances etc.) exactly without roundoff-errors.

By a straightforward reduction of the set equality problem of real numbers it can be shown that each of the problems mentioned before has an $\Omega(n \log n)$ lower bound even in the one-dimensional case ([Atk], [H]). Optimal algorithms in the two-dimensional case have been found by Highnam [H] for symmetry and by Atallah [Ata] for congruence, and in the 3-dimensional case by Atkinson [Atk] for congruence. Here we first give an alternative optimal algorithm for 3-dimensional congruence which then can be used to find the symmetry group of 3-dimensional point sets in $O(n \log n)$ time. Then we extend the result to get $O(n^{d-2} \log n)$ time congruence algorithms for arbitrary dimension $d \geq 3$.

Unfortunately, the problem of deciding congruence, similarity, and symmetry exactly, is not realistic in practice. In fact, small perturbations, which occur in the input data or during the computation because of roundoff-errors usually will destroy these properties, such that, e.g., originally congruent objects will not be recognized as congruent anymore. We therefore introduce the notion of **approximate congruence**, i.e., congruence within a certain tolerance $\epsilon > 0$. The problem of approximate congruence will be dealt with in Section 3.

2. The exact case

In this section we will assume a model of computation with exact real arithmetic and that there are no perturbations in the input data.

We first observe that the similarity problem for n-point sets in any dimension is reducible to the congruence problem in $O(n)$ time. In fact, in $O(n)$ time it is possible to determine the centroids c_A, c_B of the input sets A, B and the maximum distances m_A, m_B of c_A, c_B to the points of A, B, respectively. Then m_A/m_B is

the factor by which B is "stretched" around c_B. Clearly the set B' obtained this way is congruent to A exactly if B is similar to A. Therefore from now on we will not mention similarity anymore, but any result on congruence holds for similarity, as well.

These results are summarized in the following Theorem:

Theorem 1.

a) *For any $d \geq 3$, the congruence of two n-point sets in R^d can be decided in $O(n^{d-2} \log n)$ time.*

b) *The symmetry group of an n-point set in $\mathbb{R}^3$ can be determined in $O(n \log n)$ time.*

Proof of a) *(Congruence)*: In [Atk] an $O(n \log n)$-algorithm for $d = 3$ is given. We give an alternative one which can be applied to prove b).

Given input sets A, B our algorithm consists of the following steps:

Algorithm 1:

1. Determine the centroids c_A, c_B. If $c_A \in A$, then check if $c_B \in B$. If not, give a negative answer, otherwise remove c_A, c_B from A, B, respectively.

2. Project all points of A (B) onto the unit sphere using c_A (c_B) as an origin. Mark the points in the sets A' (B') obtained this way with the distances of the original points from c_A (c_B). Observe, that one point of the sphere can be the image of several points of A (B) and, thus, be marked with several distances. In this case, sort these distances.

3. Construct the convex hulls of A', B'. Observe, that all points in A', B' are extreme and therefore vertices of the convex hulls. Let v be any such vertex, and let $v_0, v_1, \ldots, v_{k-1}$ be the vertices adjacent to v listed in clockwise fashion about v. For $0 \leq i \leq k-1$ let l_i be the length of the arc from v to v_i and φ_i the angle on the sphere between v_i, v, and $v_{(i+1) \bmod k}$ (see Fig. 1).

In addition to the information attached in step 2 mark v with the lexicographically smallest cyclic shift of the sequence $(\varphi_0, l_0), \ldots, (\varphi_{k-1}, l_{k-1})$.

4. The convex hulls together with the labels attached in steps 2 and 3 can be considered as labelled planar graphs ([PS]). It is easy to see that the two input sets are congruent exactly if the two labelled graphs are isomorphic. We decide this isomorphism using the $O(n \log n)$ partitioning algorithm of Hopcroft [AHU, Section 4.13] as follows. Consider a finite automaton whose states are the directed edges (each undirected edge of the planar graph gives rise to two directed edges) of the planar graph and which has input alphabet $\{a, b\}$. If edge (v, w) is the current state and the input symbol is a (b) then the next state is (v, z) where edge (v, z) follows the edge (v, w) in the clockwise order of edges about v ((z, w) where (z, w) follows the edge (v, w) in the clockwise order of edges about w). Finally, let P be the following partition of the edges. Two edges (v, w) and (x, y) belong to the same block of P iff v and x are labelled the same and w and y are labelled the same. The partitioning algorithm computes the coarsest refinement of P which is compatible

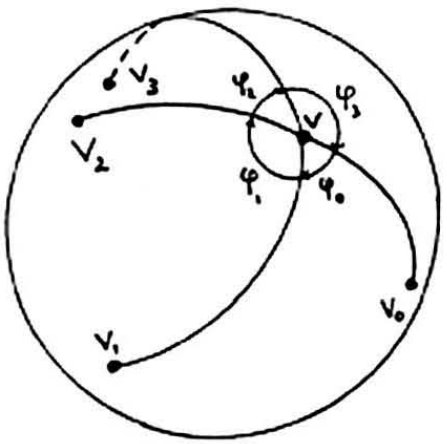


Fig. 1

with the state transitions of the automaton. This partition is clearly identical to the automorphism partition of the union of the two graphs.

Analysis: Step 1 apparently can be done in $O(n)$ time. Constructing the sets A', B' in step 2 takes time $O(n)$. However, sorting of the labels attached to one point may take time $O(n \log n)$. The construction of the convex hull in step 3 can be done in $O(n \log n)$ time ([PS]).

For each vertex v sorting the adjacent vertices $v_0, \ldots, v_{k-1}$ in a clockwise fashion takes time $O(k \log k)$. Also in step 3 the lexicographically smallest cyclic shift can be computed in time $O(k \log k)$ as follows. We first sort the pairs (φ_i, l_i), $0 \leq i < k$, in increasing order and then replace each pair by its rank. This yields a word $w_0 w_1 \ldots w_{k-1}$ with $0 \leq w_i < k$. Let $I = \{i_0 < i_1 < \ldots\}$ be the set of occurrences of letter 0. We next consider the pairs $(i_0, i_1), (i_2, i_3), \ldots$ in turn. For each pair, say (i_{2l}, i_{2l+1}), we compare the subwords of length $i_{2l+1} - i_{2l}$ starting in i_{2l} and i_{2l+1} respectively. If the subword starting at index i_{2l} is not larger lexicographically than the subword starting at i_{2l+1} then we delete i_{2l+1} from I and we delete i_{2l} otherwise. Thus in time $O(k)$ we can reduce the size of I by half and the time bound follows. However, the sum of the runtimes for all vertices does not exceed $O(n \log n)$, since the total number of adjacent vertices considered is twice the number of edges which is $O(n)$.

Algorithm 2:

If $d \leq 3$, Algorithm 1 is applied, otherwise the d-dimensional problem is reduced to n problems of dimension $(d-1)$ in the following way:

1. Construct the labelled sets A', B' as in step 2 of Algorithm 1.

2. For some point $a \in A'$ intersect the d-dimensional unit sphere S_A around c_A with some hyperplane which is orthogonal to the line segment $\overline{c_A a}$. The intersection

will be some $(d-1)$-dimensional sphere S on the surface of S_A. Project each point $b \in A' - \{a\}$ onto S along the arc from b to a on the surface of S_A. This yields a new set A'' (see Fig. 2). To each $x'' \in A''$ attach the following information: From each $x' \in A'$, which was projected onto x'', the label obtained in previous steps of the algorithm and, additionally, the length of the arc from a to x'. If there are several points in A' which are projected onto x'', list their labels sorted with respect to the lengths of the arcs. A'' still contains all the geometric information necessary to identify the original set A.

3. Do the same construction as in step 2 for all points of B, yielding labelled point sets $B_1'', \ldots, B_n''$.

4. The sets A'', B_i'' $(i = 1, \ldots, n)$ are subsets of $(d-1)$-dimensional hyperplanes. Transform them by an isometric mapping into subsets A''', B_i''' $(i = 1, \ldots, n)$ of $\mathbb{R}^{d-1}$ and apply this algorithm recursively to each pair A''', B_i''' $(i = 1, \ldots, n)$. It is easy to see that the original sets A, B are congruent exactly if there is a label preserving congruence from A''' into at least one of the sets $B_1''', \ldots, B_n'''$.

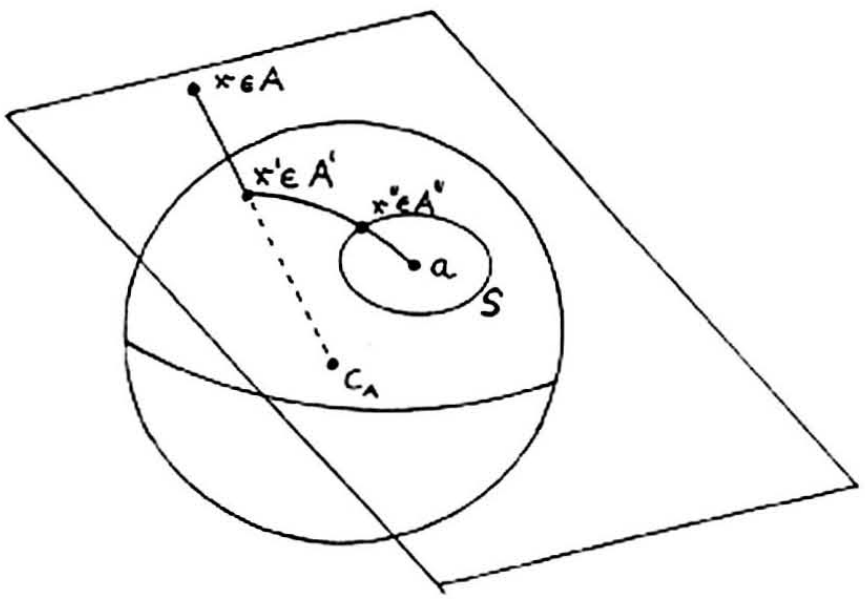


Fig. 2

Analysis: Steps 1 and 2 because of the sorting of labels may take $O(n \log n)$ time. Step 3 takes time $O(n \log n)$ for each B_i'' $(i = 1, \ldots, n)$, i.e., $O(n^2 \log n)$ alltogether. The transformations in step 4 take linear time, the recursive calls take n times the runtime of the $(d-1)$-dimensional problem. This is certainly for $d \geq 4$ the most significant term and since Algorithm 1 runs in time $O(n \log n)$ for $d = 3$, we get a total runtime of $O(n^{d-2} \log n)$.

Proof of b) *(Symmetry)*: Our algorithm for determining the symmetry-group of 3-dimensional point sets makes use of the following theorem by Hensel (1830) which states that there are only finitely many types of such groups:

Theorem 2 Hensel's Theorem (cf. [Ma]).
Any finite symmetry group for a subset of $\mathbb{R}^3$ *is one of the following groups:*

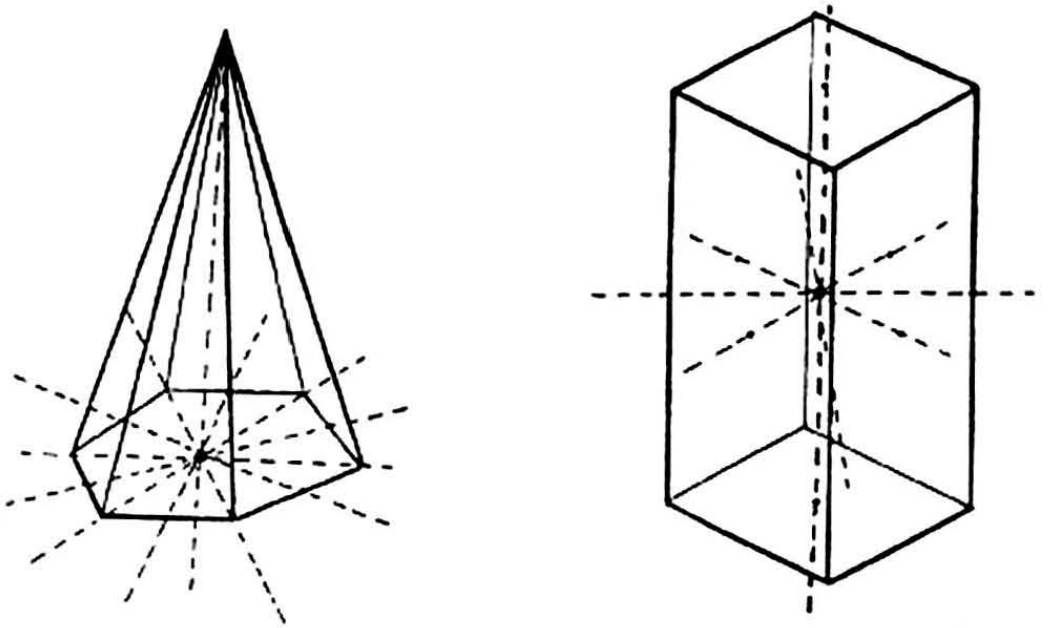
a) *the rotation groups* T, C, I *of the platonic solids tetrahedron, cube, and icosahedron, respectively,*

b) *the cyclic groups* C_n, $n = 1, 2, \ldots$ *and the dihedral groups* D_n, $n = 2, 3, \ldots$,

c) *the groups* $\overline{T}$, $\overline{C}$, $\overline{I}$; $\overline{C_1}$, $\overline{C_2}$, ...; $\overline{D_2}$, $\overline{D_3}$, ..., *where* $\overline{G}$ *means the group generated by* G *and a reflection at some point (inversion),*

d) *the groups* CT, $C_{2n}C_n$, $D_{2n}D_n$, D_nC_n, $n = 2, 3, \ldots$, *where* GH *means* $H \cup (G - H) \circ i$ *where* i *is an inversion.* ∎

For example, the sets in Fig. 3a), b) have rotation groups C_6, D_4 and symmetry groups D_6C_6, $\overline{D_4}$, respectively.



a) 6-pyramid b) 4-prism

Fig. 3

In any case the actual set of transformations that map a set of points into itself is determined by the symmetry group and the set of rotation axes. If the symmetry group is not one of the finitely many derived from the platonic solids, there will be $n + 1$ rotation axes for some $n \in N$. Like in Fig. 3 n of those axes lie in a plane the

$(n+1)$st is perpendicular to that plane. Our algorithm to determine the symmetry group of a set A of n points proceeds as follows:

Algorithm 3:

1. Apply Algorithm 1 to the input pair A, A. Hopcroft's partitioning algorithm partitions the set of points into equivalence classes of "indistinguishable" ones, i.e., points which can be mapped into each other by some congruence- (in this case symmetry-) transformation. In other words, we get the orbits of all points under the symmetry group.

2. First we check if the symmetry group is one of T, C, I, $\overline{T}$, $\overline{C}$, $\overline{I}$, CT. A detailed analysis shows that for each of these symmetry groups the size of the orbit of any point is bounded by some constant (e.g., 48 for $\overline{C}$). In addition the set of rotation axes can be determined uniquely from any nontrivial (i.e., non-singleton) orbit. Therefore, the algorithm for each of the above symmetry groups checks if the orbits satisfy the size bound. If that is the case the set of possible rotation axes is determined from one orbit. As mentioned before this determines the set of transformations. Each of these applied to all points in A and it is checked if the image set is A again. If so, the symmetry group has been found.

3. If step 2 has failed the symmetry group of A must be one of C_m, D_m, $\overline{C_m}$, $\overline{D_m}$, $C_{2m}C_m$, $D_{2m}D_m$, or D_mC_m for some $m \in \mathbb{N}$. It can be shown, that in this case each orbit has size 1, 2, m or $2m$. So, by inspecting one orbit of size > 2 the value of m and, thus, the number of symmetry groups is restricted to constantly many possibilities. The possible rotation axes are determined and all possibilities of groups are tested like in step 2.

Analysis: Step 1 takes $O(n \log n)$ time. Determining the possible sets of transformations by inspecting one orbit in step 2 takes constant time, the test if this set of transformations really maps A into itself takes time $O(n \log n)$ (comparison of 2 n-element sets). Determining the possible transformations in step 3 takes time $O(n)$, testing if they map A into itself takes time $O(n \log n)$. ∎

3. The approximate case

The major drawback of the congruence and symmetry problems considered in the previous section and previous papers ([Ata], [Atk], [H]) is that they are numerically ill-conditioned. In fact, arbitrary small perturbations in the input data will turn pairs of congruent sets into noncongruent ones and destroy symmetries. The symmetry problem even becomes trivial, if we assume that input data are represented by, say, rational cartesian coordinates.

Namely in this case only finitely many different symmetry groups are possible at all, consisting of rotations by multiples of 90-degree angles possibly combined with reflections.

This shows that the exact versions of these problems are unrealistic and numerically intractable. We therefore define the **approximate congruence problem with tolerance** ϵ:

Given two sets A, B of n points each, decide if there exists an isometric mapping which maps in a 1-to-1 fashion the points of B into the ϵ-neighborhood of points of A.

Besides making the original problem numerically tractable, the approximate version is interesting by itself. Namely, many geometric objects occurring in practice (Biology, Crystallography) are not perfectly congruent or symmetric but only within a certain tolerance. In fact the minimum possible tolerance can be considered as a quantitative measure for the degree of symmetry or congruence.

It still could be possible that the algorithms presented in Section 2 can be modified to solve the approximate congruence problem, but they cannot, due to numerical instability. For example, if points of the input sets A, B are close to the centroids c_A, c_B small perturbations will cause large changes in the projected points and change the resulting convex hulls and planar graphs completely. The same holds for the algorithms in previous papers on the subject.

We will next describe a series of results solving various cases of the approximate congruence problem in 2 dimensions. We classify these results according to the following criteria.

- type of congruence:
  - T – translation,
  - R – rotation around a center which is known,
  - I – arbitrary isometric mapping, consisting of a combination of translation, rotation and reflexion;
- knowledge of some bijective labelling $l : B \to A$ telling for any point in B into whose neighborhood it should be mapped:
  - l – known,
  - u – unknown, determine if such a labelling exists;
- specification of ϵ:
  - D – ϵ is given (decision-problem),
  - E – ϵ is given and the set A is ϵ-disjoint, i.e., $U_\epsilon(x) \cap U_\epsilon(y) = \emptyset$ for all $x, y \in A$ where $U_\epsilon(x)$ is the ϵ-neighborhood of x with respect to the underlaying metric,
  - O – find the smallest ϵ such that an approximative congruence exists (optimization-problem);
- metric:
  - e – Euclidean metric,
  - m – maximum metric.

So, for example, $TlOe$ means the problem: Given 2 sets of n points, $A, B \subseteq \mathbb{R}^2$ and a labelling $l : B \to A$, find the smallest $\epsilon > 0$ such that there exists a translation $c : \mathbb{R}^2 \to \mathbb{R}^2$ with $c(x) \in U_\epsilon(l(x))$ for all $x \in B$. We obtained the following upper bounds:

Theorem 3.

Problem	*solvable in time*
$Tl\left\{\begin{matrix}D\\O\end{matrix}\right.\left\{\begin{matrix}e\\m\end{matrix}\right.$	$O(n)$
$TuD\left\{\begin{matrix}e\\m\end{matrix}\right.$	$O(n^6)$
$TuO\left\{\begin{matrix}e\\m\end{matrix}\right.$	$O(n^{6.5}\log n)$
$TuEe$	$O(n\log n)$
$TuEm$	$O(n\log n)$
$RlD\left\{\begin{matrix}e\\m\end{matrix}\right.$	$O(n\log n)$
$RlO\left\{\begin{matrix}e\\m\end{matrix}\right.$	$O(n^2)$
$IlD\left\{\begin{matrix}e\\m\end{matrix}\right.$	$O(n^3\log n)$
$IuD\left\{\begin{matrix}e\\m\end{matrix}\right.$	$O(n^8)$

Proof: Let $A = \{a_1,\ldots,a_n\}$, $B = \{b_1,\ldots,b_n\}$ be such that in the labelled cases $l(b_i) = a_i$ for $1 \le i \le n$. Let C_i be the circle of radius ϵ around a_i, $1 \le i \le n$. (We shall write $C_i(\epsilon)$ if ϵ is not a constant.)

$Tl\left\{\begin{matrix}D\\O\end{matrix}\right.\left\{\begin{matrix}e\\m\end{matrix}\right.$:

Let $r \in \mathbb{R}^2$ be some fixed point, called the reference point. Let K_i, $1 \le i \le n$, be the set of images of r under all translations mapping b_i into $U_\epsilon(a_i)$. It is easy to see that the decision problem has a positive answer, exactly if the intersection

$$K_1 \cap \cdots \cap K_n \neq \emptyset.$$

In fact, each translation mapping the reference point into this intersection, will map each b_i into $U_\epsilon(a_i)$, $1 \le i \le n$. In the Euclidean case each K_i is (the interior of) a circle, so the question, whether the intersection is nonempty is equivalent to the question whether there exists a circle of the same radius, namely ϵ, enclosing the centers. The optimization problem thus is equivalent to finding the smallest enclosing circle of n points, which can be solved in $O(n)$ time by an algorithm due to Megiddo ([Me], cf. also [PS]).

In the case of the maximum metric the K_i $(1 \le i \le n)$ are squares (orthogonal to the axes) instead of circles and there is a straightforward $O(n)$-algorithm to find the smallest enclosing square of n points. Thus we obtain $O(n)$-algorithms for the optimization and, consequently, the decision problem.

$TuD\left\{\begin{matrix}e\\m\end{matrix}\right.$:

Algorithm 4:

We present the algorithm for the Euclidean case but it works for the maximum metric, as well. It uses the following ideas:

1. An easy geometric argument shows that, if there is any solution at all, then there must be one where some point b_j lies directly on the circle C_i of radius ϵ around the assigned point a_i $(1 \le i, j \le n)$. We check this property for all pairs i, j.

For fixed i, j describe any position of b_j on the circle C_i by the polar coordinate φ using a_i as the origin and, for example, the ray parallel to the x-axis in positive direction through b_i as the $\varphi = 0$-ray (see Fig. 4).

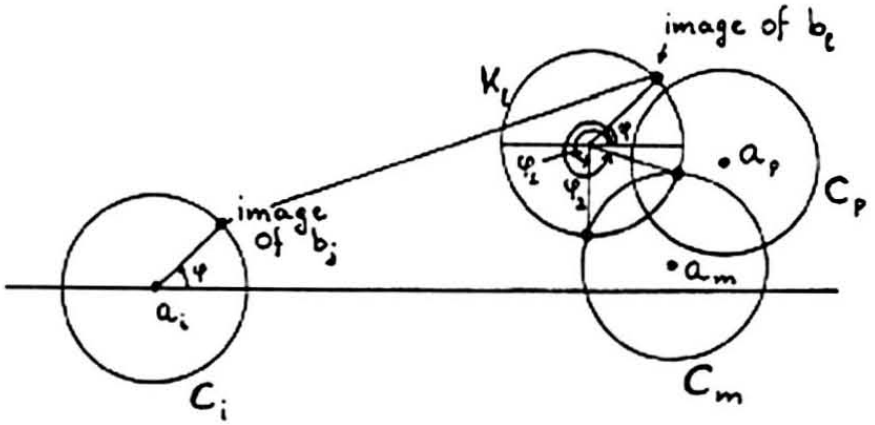


Fig. 4

2. For any $l \neq j$ consider the circle K_l onto which b_l is mapped by translations which map b_j onto C_i. For any $m \neq i$ we determine the set $I_{l,m}$ of angles $\varphi \in [0, 2\pi[$ such that if the image of b_j on C_i has polar coordinate φ, then b_l lies within $U_\epsilon(a_m)$. For example, in Fig. 4 $I_{l,m} = [\varphi_1, \varphi_2]$. In any case $I_{l,m}$ is a (circular) interval of $[0, 2\pi[$ whose endpoints are the cutpoints of K_l and C_m. We determine all these cutpoints and sort them. This gives a partition of the interval $[0, 2\pi[$ into subintervals each of which is the intersection of some of the $I_{l,m}$ $(1 \le l, m \le n;\ l \neq j, m \neq i)$ and the complements of others.

3. We assign to each such subinterval I a bipartite graph $G_I = (V, E_I)$, where V is the disjoint union of two sets $\{u_1, \ldots, u_{i-1}, u_{i+1}, \ldots, u_n\}$ and $\{v_1, \ldots, v_{j-1}, v_{j+1}, \ldots, v_n\}$ of nodes and $E_I = \{(u_l, v_m);\ I \subset I_{l,m}\}$. Now suppose that the graph G_I for some interval I contains a perfect matching, namely edges $(u_{i_1}, v_{j_1}), \ldots, (u_{i_{n-1}}, v_{j_{n-1}})$. This means that $I \subset I_{i_1 j_1} \cap \cdots \cap I_{i_{n-1} j_{n-1}}$. Therefore any translation mapping b_j onto C_i such that the polar coordinate of the image of b_j is in I will map b_{j_k} into $U_\epsilon(a_{i_k})$, $1 \le k \le n-1$, i.e., there exists an assignment which shows that A and B are approximately congruent. First we determine the graph for the subinterval containing $\varphi = 0$. This can be done by checking for each pair l, m, if the image of b_l lies within $U_\epsilon(a_m)$. We then determine the maximum matching of the graph (cf. [M]) and, if it is perfect, give a positive answer.

4. Starting from the maximum matching of the graph for the first subinterval I_1 determined in the previous step we proceed to neighboring intervals using the sorted

sequence of endpoints. Apparently, each time, we proceed from one interval I_k to its neighbor I_{k+1}, an edge has to be deleted from the corresponding graph or a new one has to be added. In the first case, the maximum matching of $G_{I_{k+1}}$ can be determined from the one of G_{I_k} in the following way: If the edge deleted was not part of the maximum matching then it stays the same for $G_{I_{k+1}}$. Otherwise, check if there is an augmenting path, and if so, use it to enlarge the matching. If $G_{I_{k+1}}$ results from G_{I_k} by adding a new edge, determine if an augmenting path exists. If so, use it to enlarge the matching. There is an approximate congruence of A and B exactly if one of the maximum matchings is perfect.

Analysis: Step 1 states that steps 2–4 are repeated n^2 times for all pairs i, j, $1 \le i, j \le n$. In step 2 all cutpoints of circles K_l and C_m, $1 \le l, m \le n$; $l \ne j, m \ne i$ are determined and sorted with respect to their angular polar coordinate. This obviously takes time $O(n^2 \log n)$. In step 3 determining the graph belonging to $\varphi = 0$ takes $O(n^2)$, determining its maximum matching $O(n^{2.5})$ time, cf. [M, Section IV.9.2]. In step 4 we check for each of the $O(n^2)$ subintervals if the corresponding graph together with the previous matching contains an augmenting path, which takes $O(n^2)$ time. So the total runtime of one iteration of step 4 is $O(n^4)$, which asymptotically exceeds the runtimes of steps 2 and 3. The total runtime of the algorithm is thus $O(n^6)$.

$Tu O\left\{\begin{matrix} e \\ m \end{matrix}\right.$:

We extend the technique used in Algorithm 4.

Algorithm 5:

1. An easy geometric argument shows that for the optimal value ϵ_{opt} of ϵ there must be a solution with at least two points b_j and b_l lying on the circles $C_i(\epsilon)$ and $C_k(\epsilon)$ of radius ϵ around the assigned points a_i and a_k respectively. Let $\epsilon(i, j, k, l)$ be the minimal ϵ such that there is a solution with b_j on $C_i(\epsilon)$ and b_l on $C_k(\epsilon)$. We show how to compute $\epsilon(i, j, k, l)$ in time $O(n^{2.5} \log n)$.

2. Let $T : x \mapsto x + t$ be a translation which maps b_j on $C_i(\epsilon)$ and b_l on $C_k(\epsilon)$. Then $|T(b_j) - a_i|^2 = \epsilon^2 = |T(b_l) - a_k|^2$ and hence $|t + b_j - a_i|^2 = \epsilon^2 = |t + b_l - a_k|^2$. Thus there are only two possible values for t, which depend on ϵ. We denote these functions by $t_1(\epsilon)$ and $t_2(\epsilon)$; they have the same domain D. For each function $t \in \{t_1, t_2\}$ separately, we determine the minimum ϵ (if one exists at all) such that $\epsilon \in D$ and a translation by $t(\epsilon)$ constitutes an approximate congruence between B and A. We define for $1 \le p, q \le n$

$$M_{pq} := \{\epsilon \in D;\ b_q + t(\epsilon) \in U_\epsilon(a_p)\}$$

$$\text{and } \epsilon_{pq} := \begin{cases} \min M_{pq} & \text{, if } M_{pq} \ne \emptyset \\ \infty & \text{, otherwise.} \end{cases}$$

The ϵ_{pq} are constants which can be determined from the coordinates of a_p, b_q, a_i, b_j, a_k, b_l.

3. $\epsilon(i,j,k,l)$ is clearly equal to one of the n^2 values ϵ_{pq}. We sort these n^2 values in time $O(n^2 \log n)$ and then determine $\epsilon(i,j,k,l)$ by binary search. Each query of the binary search has a cost of $O(n^{2.5})$ since it is tantamount to deciding the existence of a perfect matching; cf. Algorithm 4.

TuEe:

Algorithm 6:

1. We first want to argue that the labelling used in a solution (if there is any) is unique. Assume o.w., say l_i is a labelling under translation $T_i : x \mapsto x + t_i$, $i = 1, 2$. We may assume w.l.o.g. that $l_1^{-1}(a_j) = b_j$, $l_2^{-1}(a_j) = b_{j+1}$ and $l_2^{-1}(a_k) = b_1$ for $1 \le j \le k$ and some k. Since T_i is a solution with respect to labelling l_i, we have

$$b_j + t_1 \in U_\epsilon(a_j) \quad \text{for } 1 \le j \le k$$
$$b_j + t_2 \in U_\epsilon(a_{j-1}) \quad \text{for } 2 \le j \le k$$
$$b_1 + t_2 \in U_\epsilon(a_k).$$

Let $v = t_1 - t_2$. Then $(a_j - a_{j-1}) \cdot v \ge 0$, as can be seen from Fig. 5. Since $\sum_{j=2}^{k}(a_j - a_{j-1}) + (a_1 - a_k) = 0$ we conclude $v = 0$. This contradicts the fact that $U_\epsilon(a_j) \cap U_\epsilon(a_{j-1}) = \emptyset$ for all j. Thus there is at most one labelling which can be used in an approximate congruence.

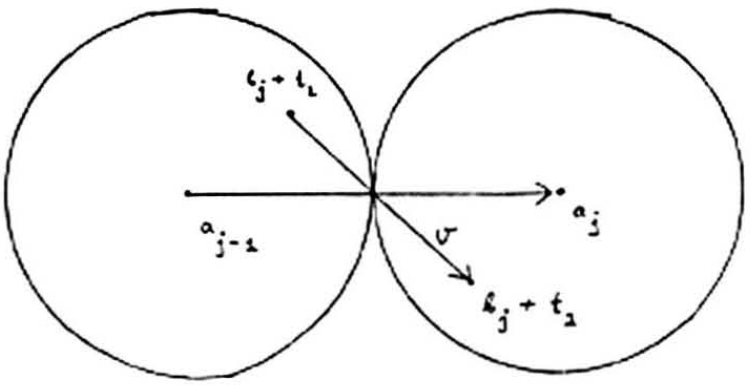


Fig. 5

2. It can easily be shown that any translation which is an approximate congruence between B and A also must map the centroid c_B into $U_\epsilon(c_A)$. We cover $U_\epsilon(c_A)$ by circles which have sufficiently small radius ϵ_0 so that they cannot intersect more than 2 circles of radius ϵ around points of A. Fig. 6 shows that $\epsilon_0 = (\frac{2}{3}\sqrt{3} - 1)\epsilon$ will do.

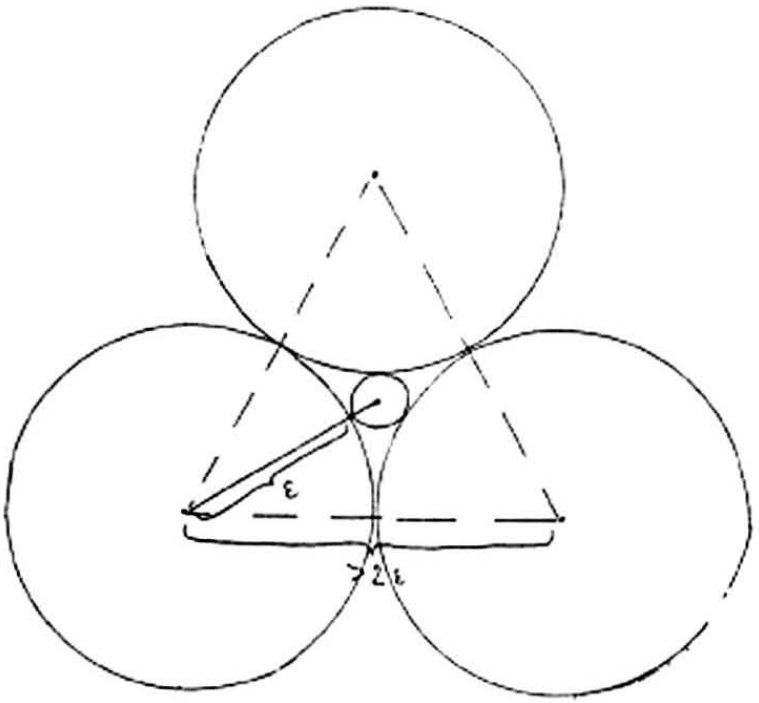


Fig. 6

For each small circle C of this cover (there are constantly many of them) we decide, if there exists a translation mapping c_B into C and satisfying the conditions for an approximate congruence between A and B.

3. Let K_i, $1 \leq i \leq n$, be the set of images of b_i under translations mapping c_B into C, i.e., the circle of radius ϵ_0 around $b_i' := b_i + c_A - c_B$. For the set A we construct the first and second order Voronoi diagrams. Then we locate each b_i' in order to determine the 2 points in A closest to it. For the ones, which are at a distance less than $\epsilon + \epsilon_0$ from b_i', i.e., K_i intersects their ϵ-neighborhoods, we introduce an edge in an undirected graph G of the same form as in Algorithm 4.

4. In the graph G constructed in step 2 any node in the set $\{v_1, \ldots, v_n\}$ has degree ≤ 2. We determine if G has a perfect matching in the following way: If there is any node of degree 0, no perfect matching exists. Otherwise, if there is some node which has degree 1 we add the incident edge e to the matching, and remove e, its endpoints and all other edges incident to them from the graph. We repeat this until there are no more edges of degree 1 left. Now all remaining edges in U and in V must have degree 2 and hence each connected component is a cycle. For each cycle there are only two possible matchings. We check both matchings by the linear time algroithm for the labelled case. By the argument at the beginning of the proof at most one of the labellings can yield an approximate congruence. Once

we have checked each cycle only one labelling is left. We check this labelling by the linear time algorithm.

Analysis: Step 2 states that steps 3–4 have to be executed for each circle in the cover, i.e., a constant number of times. Constructing the Voronoi diagrams and locating the points b_i' in them takes $O(n \log n)$ time (cf. [PS] or [M]). Determining the matchings in step 3 takes $O(n)$, testing the two matchings for a cycle of length c takes time $O(c)$ for a total of $O(n)$ over all cycles. The final test takes also time $O(n)$.

TuEm:

Algorithm 7:

We proceed like in Algorithm 6. Here we have squares instead of circles. Let $\delta = \min_{i \neq j} dist(a_i, a_j) > 2\epsilon$. In step 1 we cover the square of "radius" ϵ and center c_A by small squares of radius $\leq \delta - 2\epsilon$ which can intersect at most one of the ϵ-neighborhoods of the points of A. This makes the matching problem in step 4 even easier. The total runtime is clearly $O(n \log n)$.

$RID \begin{Bmatrix} c \\ m \end{Bmatrix}$:

Algorithm 8: Let c be the center of the rotation and $I_i \subset [0, 2\pi[$ the set of angles under which a rotation around c maps b_i into $U_\epsilon(a_i)$, $1 \leq i \leq n$. Clearly the problem instance has a positive answer exactly if $I_1 \cap \cdots \cap I_n \neq \emptyset$. In the case of the Euclidean metric any I_i is an interval on the unit circle. We sort the boundaries of these intervals ($O(n \log n)$ time). Then we count, how many b_i are within $U_\epsilon(a_i)$, (i.e., within the correct ϵ-neighborhood when a rotation of 0 degrees is performed.) We traverse the sequence of interval boundaries, and increment (decrement) our count by 1 for each left (right) interval boundary. Whenever the count becomes n, we have found a rotation of the desired form.

In case of the maximum metric each I_i consists of at most 4 intervals, namely the set of intervals on a circle, which are within a given square. Again we sort the boundaries of all these intervals and proceed like in the Euclidean case.

The remaining algorithms are described for the Euclidean metric but they work for the maximum metric as well.

$RIO \begin{Bmatrix} c \\ m \end{Bmatrix}$:

Let φ be the angle of a rotation around c satisfying approximate congruence for a minimum value of ϵ. An easy geometric argument shows that in this situation there must be i, j $(1 \leq i, j \leq n)$ such that b_i and b_j are mapped onto the circles C_i, C_j, respectively, of radius ϵ around a_i, a_j. Furthermore, any increment of φ must move one of the images out of and the other one into the corresponding circle (see Fig. 7).

For any pair i, j only one such situation is possible. We denote by $\epsilon_{i,j}$ the corresponding value of ϵ, which can be determined by solving 2 equations with 2

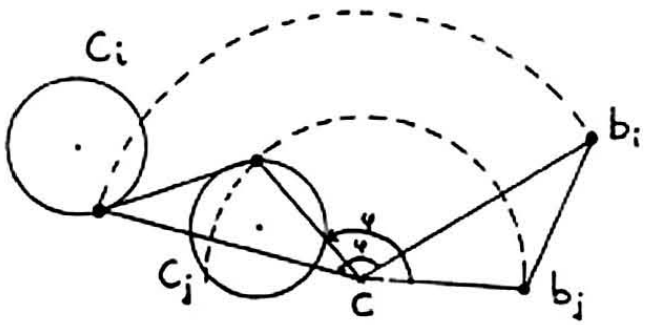


Fig. 7

unknowns (ϵ and φ). We determine the set E of all these values ($1 \leq i, j \leq n$). The solution ϵ of the optimization problem must be one of them, and we determine it by a variant of binary search. First we determine the median ϵ_M of E and apply Algorithm 8 to find out, if there is a solution with tolerance ϵ_M. If so, we apply the algorithm recursively to the set E_1 of elements of E less than ϵ_M, otherwise to the set E_2 of elements greater than ϵ_M until the smallest element of E for which a solution exists, is found. The time to compute the set E is $O(n^2)$. The first step of the binary search takes time $O(n^2)$ to find ϵ_M by fast median finding ([M], Section II.4), $O(n^2)$ to split E with respect to ϵ_M, and $O(n \log n)$ to apply Algorithm 8. It follows, that the total runtime is $O(n^2)$.

$IID \begin{Bmatrix} c \\ m \end{Bmatrix}$:

Any isometric mapping consists either of a rigid motion, i.e., a combination of translation and rotation, or a reflexion (at some straight line in the 2-dimensional case), which can be chosen arbitrarily, followed by a rigid motion. We will give an algorithm which tests if B can be mapped onto A with tolerance ϵ by a rigid motion. We consider both versions of isometric mappings by applying this algorithm once to the pair A, B and once to A, B', where B' is obtained from B by reflecting it at some arbitrary straight line.

A simple geometric argument shows that there exists a rigid motion mapping any b_i into $U_\epsilon(a_i)$, $1 \leq i \leq n$, exactly if there exists a rigid motion of that kind which maps at least 2 points b_i, b_j of B onto the borders of their corresponding ϵ-neighborhoods, i.e., the circles C_i, C_j. Our algorithm tests this property for all $O(n^2)$ possible pairs (i, j). Given such a pair we denote by the parameter $\varphi \in [0, 2\pi[$ the polar coordinate of the image of b_i using a_i as an origin (see Fig. 8).

Clearly, for any value of φ there are at most 2 possible images of b_j on C_j. Once the images of b_i, b_j are fixed, the mapping is uniquely determined. Let b_p be some arbitrary other point of B. When the images of b_i, b_j are moved on the circles C_i, C_j the image b_p describes in cartesian coordinates some algebraic curve K_p (see Fig. 8) which cuts C_p in constantly many (in fact, ≤ 6) points.

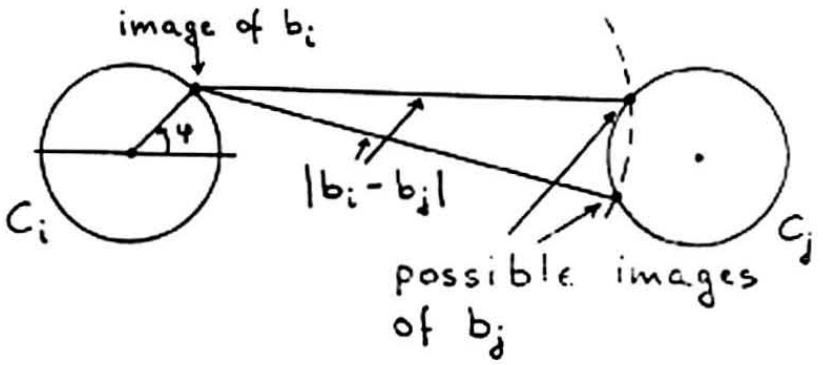


Fig. 8

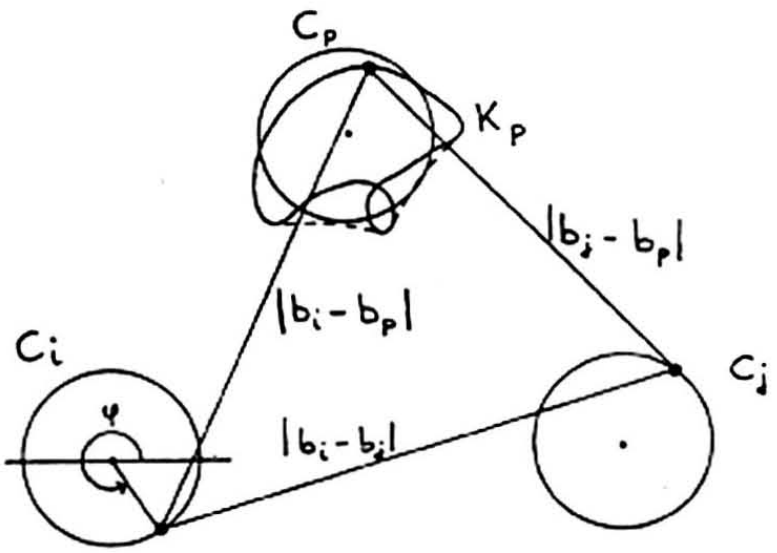


Fig. 9

Thus, we obtain for each $p \neq i, j$ a union I_p of constantly many intervals for φ within which b_p is mapped into C_p. We compute the endpoints of these intervals, sort them ($O(n \log n)$ time), and determine, by scanning and counting, if the intersection of all I_p, $p \neq i, j$ is nonempty.

The algorithm takes time $O(n \log n)$ for each pair i, j, so the total runtime is $O(n^3 \log n)$.

$IuD \begin{Bmatrix} e \\ m \end{Bmatrix}$:

As in the labelled case, if there is a solution, there must be one which maps at least 2 points b_i, b_j onto circles C_k, C_l. We test this property for all quadruples

$i, j, k, l \in \{1, \ldots, n\}$. The coordinate φ is defined like in the labelled case, only that now the image of b_i lies on C_k instead of C_i. The algebraic curve K_p, $1 \leq p \leq n$ of the image of b_p is defined analogously to the labelled case. Let $I_{p,m}$, $1 \leq m \leq n$, be the set of values of φ, such that K_p lies within $U_\epsilon(a_m)$. Like before, $I_{p,m}$ is the union of at most 6 intervals. We sort the boundaries of all the intervals obtained this way and proceed exactly as in Algorithm 4. Whenever we find a perfect matching, the problem has a solution.

Steps 2–4 of Algorithm 4 are executed $O(n^4)$ times, one execution takes $O(n^4)$ time, so we get an upper bound of $O(n^8)$. ∎

4. References

[AHU] Aho, A.V./J.E. Hopcroft/J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974

[Ata] Atallah, M.J., *Checking Similarity of Planar Figures*, International J. Comp. Inf. Science 13 (1984), pp. 279–290

[Ata] Atallah, M.J., *On Symmetry Detection*, IEEE Trans. on Comp. C-34 (1985), 663–666

[Atk] Atkinson, M.D., *An Optimal Algorithm for Geometrical Congruence*, SCS - TR - 31, Carleton University, Ottawa, 1983

[H] Highnam, P.T., *Optimal Algorithms for Finding the Symmetries of a Planar Point Set*, Information Processing Letters 22 (1986), pp. 219–222

[M] Mehlhorn, K., *Data Structures and Algorithms, Vols 1, 2, 3*, Springer-Verlag, 1984

[Ma] Martin, G.E., *Transformation Geometry*, Springer-Verlag, 1982

[Me] Megiddo, N., *Linear Time Algorithm for Linear Programming in* $\mathbb{R}^3$ *and Related Problems*, SIAM J. on Computing 12 (1983), pp. 759–776

[PS] Preparata, F.P. and M.I. Shamos, *Computational Geometry*, Springer-Verlag, 1985