Algorithmica (1995) 13: 387–404

Algorithmica

Point Set Pattern Matching in d-Dimensions

P. J. de Rezende[1] and D. T. Lee[2]

Abstract. In this paper we apply computational geometry techniques to obtain an efficient algorithm for the following point set pattern matching problem. Given a set S of n points and a set P of k points in the d-dimensional Euclidean space, determine whether P matches any k-subset of S, where a match can be any similarity, i.e., the set P is allowed to undergo translation, rotation, reflection, and global scaling. Motivated by the need to traverse the sets in an orderly fashion to shun exponential complexity, we circumvent the lack of a total order for points in high-dimensional spaces by using an extension of one-dimensional sorting to higher dimensions (which we call "circular sorting"). This mechanism enables us to achieve the orderly traversal we sought. An optimal algorithm (in time and space) is described for performing circular sorting in arbitrary dimensions. The time complexity of the resulting algorithm for point set pattern matching is $O(n \log n + kn)$ for dimension one and $O(kn^d)$ for dimension $d \geq 2$.

Key Words. Pattern matching, Circular sorting, Subset similarity.

1. Introduction. Given objects, say, in two-dimensional space, it is the objective of pattern recognition to explore feature extraction, identification, and classification. The inherently geometric flavor of the first two phases invites the use of tools and techniques encountered in the framework of computational geometry.

In many cases the objects in question are either specified directly as sets of points or can be approximated to such sets while maintaining the relevant identifying information intact.

This paper deals with the following problem which is frequently encountered in pattern recognition, image processing, and computer vision. Given a set of points S (the *sampling* set) and a set of points P (the *pattern* set), determine whether P has a *match* in S. Depending on the application, the notion of a match may be restricted to exact match (actual equality of the sets) or made to encompass significantly more flexible transformations such as translation, rotation, scaling, reflection, and subset matching. When a match can be found by applying to P just a translation, a rotation, and possibly a reflection (and no scaling), the match is called a *congruence* and if a scaling transformation is also allowed, it is referred to as a *similarity*.

[1] Departamento de Ciência da Computação, Universidade Estadual de Campinas, 13081 Campinas, SP, Brazil. Supported in part by CNPq—Conselho Nacional de Desenvolvimento Científico e Tecnológico (Brazil) under Grants 200331/79, 300157/90-8, and 500787/91-3.

[2] Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208, USA. Supported in part by the National Science Foundation under Grant CCR 8901815.

Received October 10, 1992; revised July 16, 1993. Communicated by B. Chazelle.

In [3] Atkinson studies the very particular case of the congruence problem under the restrictive assumption that $|S| = |P| = k$ and presents an $O(k \log k)$-time algorithm.[3] This result is extended to an $O(k^{d-2} \log k)$-time algorithm for dimension $d \geq 3$ in [2].

The main difficulty we have to face when attacking the problem under the concept of subset matching, i.e., $|P| = k < n = |S|$, is how to avoid having to look at the exponential number of k-subsets of S $\left(\text{there are } \binom{n}{k} \in O(n^k) \text{ of them}\right)$. This can be achieved in dimension one by sorting the points by x-coordinate and performing a left to right scan.

Sorting points in the real line leads to significant improvements in solutions of an extraordinary number of problems, including many of a geometric nature. Similarly, by linearly sorting points in the plane (or in higher-dimensional space) by coordinate axes, it is possible to develop techniques (such as sweeping) through which we can achieve speedups of orders of magnitude for a variety of problems. Unfortunately, sorting point sets lexicographically by coordinates does not convey enough information to allow us to traverse the set in a way convenient to some applications.

Other methods have been developed for attacking planar geometric problems which involve some flavor of ordering. A prime example is Graham's planar convex hull algorithm which relies on the ordering of the points around one particular point, known to be interior to the hull. Extending this method of lexicographically ordering points in the plane by *polar* coordinates around a fixed point, to one such ordering for *each* point, enough information about the order properties of a set can be captured to lead to a systematic method for its traversal in an orderly fashion.

In [8] Goodman and Pollack develop a procedure of canonically ordering the points of a configuration and present a very elegant characterization of the order-type of a set of points in d-space in terms of a matrix denoted therein by "λ-matrix." This characterization naturally leads to a method of comparing configurations of n points in d-space by verifying whether their λ-matrices agree.

The novel notion of *geometrically sorting* a set of points in d-space (see [8]), remarkable as it is, helps little, however, in developing a procedure for subset matching.

We extend the canonical ordering procedure of [8] to a procedure that we call *circular sorting*, and present an optimal algorithm for carrying it out based on an optimal representation of arrangements of hyperplanes introduced by Edelsbrunner *et al.* in [6].

This paper is organized as follows. In Section 2 we state the point set pattern matching problem and define the notion of match so that instances of the problem of increasing difficulty can be dealt with separately and appropriate algorithms presented. Two simple cases appear in Section 3. The discussion of the one- and

[3] Atkinson's algorithm actually determines *all* congruences (if any) between two given k-sets in three-dimensional space under the same $O(k \log k)$ time bound (which he shows to be optimal).

two-dimensional cases and a brief description of circular sorting in two dimensions are presented in Section 4. In Section 5 we generalize the concept of circular sorting to higher dimensions and present optimal algorithms for its computation. The required material on affine geometry and geometric duality is also presented in the subsections therein. The more laborious algorithms for the high-dimensional case of the point set pattern matching problem comes with Section 6. Lastly, we describe how to extend the notion of match to account for scaling in Section 7. Conclusions and remarks appear in Section 8.

2. Definitions. The model of computation that we assume is the usually adopted model of a random access machine (RAM), similar to that described in [1] with the addition of real number arithmetics. Furthermore, each arithmetic operation (addition, multiplication, and division) is assumed to be performed to arbitrary precision in one time unit. Other primitive operations which may be regarded as unit cost include finding the intersection of two lines and computing the distance between two points. While we sometimes refer to comparing angles between hyperplanes, the actual angles do not need to be computed; instead, equivalent comparisons requiring only the computations of determinants can be employed (see [10]).

Throughout this section, consider the space in which the point sets lie to be the Euclidean space $\mathbb{R}^d$ where $d \geq 1$.

POINT SET PATTERN MATCHING PROBLEM (OR PSPM)
Instance: A (sampling) set S of n points in $\mathbb{R}^d$ and a (pattern) set P of k points in $\mathbb{R}^d$ $(n \geq k)$.
Question: Does a k-subset of S matching P exist?

Note that a naïve solution to this problem (see Figure 1, for $d = 2$) which amounts to testing all $\binom{n}{k}$ possible k-subsets of S against P has time complexity exponential in k.

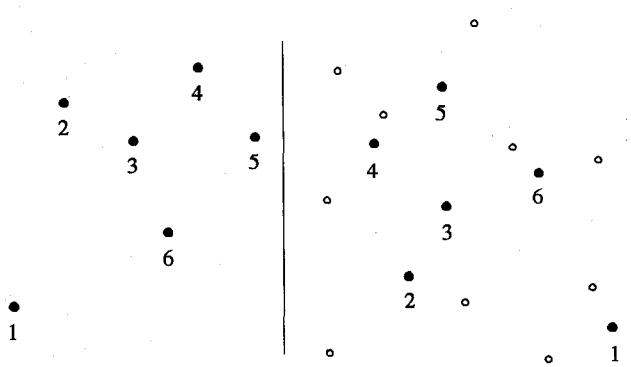


Fig. 1. An instance of PSPM.

To make the statement of the problem precise, we must give a formal definition for the notion of "match." Depending on the application at hand, one of various possibilities is more applicable. It will be made clear in the context of the subsequent sections which of the following notions the term match means.

DEFINITION 1. We employ the generic term match to mean one of the following notions, depending on the context. Given a seat P of k points and a set Q also of k points, P may be:

1. equal to Q,
2. a translation of Q,
3. a rotation of Q,
4. a reflection of Q,
5. an arbitrary isometric mapping of Q (consisting of a combination of translation, rotation, and reflection), or
6. a scaled copy of Q (possibly combined with an isometry).

Whenever it becomes necessary to specify which of these notions is the meaning of match in the statement of PSPM, we write one of PSPM[equality], PSPM[translation], PSPM[rotation], PSPM[reflection], PSPM[isometry], PSPM[scaling], or some combination of these to allow more than one type of transformation.

3. Set Equality, Translation, and Reflection. Note that PSPM[equality] demands determining whether the pattern $P = \{p_1, p_2, \ldots, p_k\}$ is a subset of the sampling $S = \{s_1, s_2, \ldots, s_n\}$. This can be done in $\mathbb{R}^d$ in $\Theta(dn \log n)$ time as follows:

Algorithm PSPM[equality]$_d$.

1. Sort the points in both sets by cartesian coordinates, obtaining d sorted lists.
2. **for** each $p \in P$ **do**
   (a) Determine by binary search in the sorted lists of S whether p is in S.
   (b) **if** p is found to be equal to some $s_i \in S$
   then output the index i and proceed
   else output *failure* and halt.

Step 1 can be done in $\Theta(dn \log n)$ time and step 2 in $O(dk \log n)$ total time. Alternatively, step 2 could be implemented as a simple simultaneous scan of P and S through the sorted lists in $O(d(k + n))$ time, which is an improvement when $k > n/\log n$.

Similarly, if translation is the only transformation allowed, a simple extension of the above procedure constitutes an efficient solution. Namely, for each $s \in S$ check whether the translated set $P + (s - p_1)$ is a subset of S.

Algorithm PSPM[translation]$_d$.

1. Sort the points in both sets by cartesian coordinates.
2. **for** each $s \in S$ **do**
   (a) **for** each $p \in P$ **do**
      i. Determine by binary search in the sorted lists of S whether $p + (s - p_1)$ is in S.
      ii. **if** $p + (s - p_1)$ is found to be equal to some $s_i \in S$
      then output the index i and proceed
      else try again with the next $s \in S$ if there is one, otherwise output *failure* and halt.

The reader can easily verify that the time complexity of this algorithm is $\Theta(dn \log n)$ for step 1 and $O(dnk \log n)$ for step 2. Again, step 2 can alternatively be done in $O(dn(n + k))$ time if we employ scanning instead of binary search in the inner part of step 2 when $k > n/\log n$.

We now concentrate on the nontrivial cases. In Section 4 we see that the above algorithm can be improved to work in time $O(n \log n + kn)$ in dimension one. The method of scanning employed there is modified to work for the higher-dimensional cases where transformations other than translations are allowed.

OBSERVATION 1. *We should observe at this point that if P is an isometric image of Q, then an isometry φ: $\mathbb{R}^d \to \mathbb{R}^d$ consisting of the combination of exactly one translation T, one rotation R, and possibly one reflection F exists such that $\varphi(P) = Q$. Furthermore, if a reflection is involved, then for any given reflection F' a translation T' and a rotation R' exist such that $\varphi = T' \circ R' \circ F'$. This leads to a fortunate simplification of future complexity arguments involving reflection transformations. That is, to decide whether a set P matches any subset of S, when any isometric transformation is allowed, it suffices to solve two simpler problems. Namely, decide whether*:

1. *P matches any subset of S through a combination of one translation and one rotation.*
2. *A reflected copy of P (with respect to some fixed reflection) matches any subset of S through a combination of one translation and one rotation.*

NOTATION. Throughout the remaining sections, unless noted otherwise, we employ the term match of two sets P and Q to mean that P is a combination of a translation and a rotation of Q (or simply a translation in the case of dimension one).

We leave scaling to be considered in Section 7. By doing this, we can concentrate on presenting the algorithms without being concerned with additional details.

4. PSPM in One and Two Dimensions

4.1. PSPM in One Dimension. In this section we study the PSPM problem defined in Section 2 for the Euclidean line $\mathbb{R}^1$. The reason for presenting the rather

simple one-dimensional case is twofold. Firstly, it provides a more straightforward way of introducing the basic scanning scheme and, secondly, the forthcoming one-dimensional algorithm is essentially the inner loop of the algorithm for the higher-dimensional cases.

Throughout this section let S be an n-point set in the real line and let P be a k-point set also in the real line (with $n \geq k$). Denote $S = \{s_i | i = 1, 2, \ldots, n\}$ and $P = \{p_i | i = 1, 2, \ldots, k\}$.

We now present an algorithm which finds a k-subset M of S matching P if one exists. The outcome of the algorithm is the set of indices $M = \{m_i | 1 \leq i \leq k\} \subset \{1, 2, \ldots, n\}$ so that each $s_{m_i} \in S$ is the matching point corresponding to $p_i \in P$. If no match exists, the empty set is returned.

The algorithm proceeds as follows. After sorting both sets, it scans P and S from left to right defining the m_i's on the fly. As long as a match is found, it proceeds looking for matches for the next points of P. If a match of $\{p_1, \ldots, p_i\}$ cannot be extended to p_{i+1}, it advances the attempted matches $\{m_1, m_2, \ldots, m_i\}$ of $\{p_1, p_2, \ldots, p_i\}$ to their successors in S and proceeds from the largest index $j \leq i$ for which the new match is valid. The algorithm succeeds if P is scanned up to p_k and fails if m_1 is advanced beyond $n - k + 1$. See Figure 2 for an illustration.

Algorithm PSPM_1.

1. Sort S and P, so that $s_i \leq s_{i+1}$ for $1 \leq i \leq n-1$ and $p_i \leq p_{i+1}$ for $1 \leq i \leq k-1$.
2. Initialize $m_i = i$ for $1 \leq i \leq k$.
3. [The first point always matches].
   Let $i = 1$.
4. **while** $i < k$ **do**
   4.1. [Get the next point in P.]
   Let $i = i + 1$.
   4.2. [Guarantee that m_i is ahead of m_{i-1} in case of an earlier failure.]
   Let $m_i = \mathbf{max}\{m_i, m_{i-1} + 1\}$.
   4.3. **while** $d(p_1, p_i) > d(s_{m_1}, s_{m_i})$ **and** $m_i < n - k + i$ **do** [Must try further.]
   Advance m_i.
   4.4. **if** $d(p_1, p_i) \neq d(s_{m_1}, s_{m_i})$
   then if $m_i \geq n - k + i$ **then** output *failure* and halt
   else advance m_1 and let $i = 1$.
5. Output $(\{m_i | 1 \leq i \leq k\})$ [A match was found.]

To see that the algorithm always halts, we show that either i reaches k or m_1 reaches $n - k + 1$. First note that in the loop of step 4, m_1 never backtracks and that although i is never decremented, it may be reset to 1, in step 4.4. Whenever this happens, m_1 is advanced and hence loop 4 always terminates.

To establish the correctness of the algorithm, first suppose it returns $\{m_i | 1 \leq i \leq k\}$. It follows that in step 5 $i = k$ and, hence, i must have been incremented from 1 to k in step 4.1 (possibly after being reset to 1 in step 4.4) and

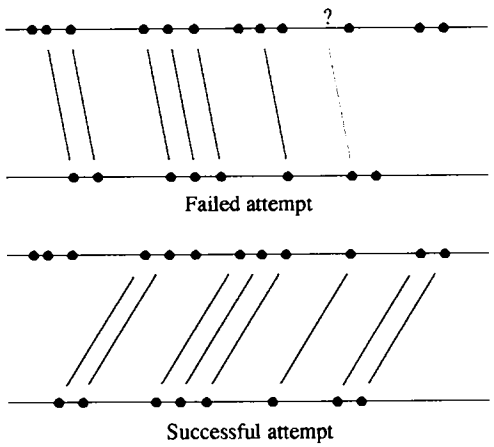


Fig. 2. Illustrating PSPM_1.

therefore the following must hold:

$$d(p_1, p_i) = d(s_{m_1}, s_{m_i})$$

for $i = 1, 2, \ldots, k$; i.e., $\{s_{m_i} | 1 \leq i \leq k\}$ is indeed a match. Conversely, suppose there is a match and let $\mu_1, \ldots, \mu_k$ be the indices of the leftmost one. It is easy to see from the argument above that m_1 will eventually advance to μ_1. The result now follows from a simple induction argument (on k).

The time complexity of the algorithm follows from the following observation. Given an i between 1 and k, m_i is advanced to each s_l for $l = i, \ldots, n - k + i$ at most once and never backtracks. Hence, the total time complexity is bounded by $O(n \log n)$ for sorting plus $k(n - k) \in O(kn)$ for traversing the sets.

Therefore, we have proved:

THEOREM 1. *Given a set P of k points and a set S of $n \geq k$ points in the real line, the Point Set Pattern Matching problem can be solved in $O(n \log n + kn)$ time.*

4.2. Circular Sorting in Two Dimensions. While the lack of a total ordering for points in dimensions higher than one precludes us from extending the notion of linear sorting, a number of plausible generalizations have appeared in the literature (e.g., [8]). Consider, for instance, the following one. Given a set of n points $S = \{s_1, s_2, \ldots, s_n\}$ in the plane, find the ordering of the points in $S \backslash \{s_i\}$ around s_i for each $i = 1, 2, \ldots, n$. The ordering is obtained by lexicographical sorting the points by their polar angles around s_i in the counterclockwise direction and their distances from s_i. See Figure 3. While this notion has been referred to as two-dimensional sorting, we prefer the more descriptive term *circular sorting—in two dimensions*—which we later generalize to higher dimensions.

This two-dimensional problem is frequently encountered in computational geometry and has received some attention. The simplest algorithm for circular sorting a planar set S of n points consists of performing, for each $s \in S$, the sorting of the points of $S \backslash \{s\}$ around s lexicographically by polar coordinates (θ_i, r_i), where

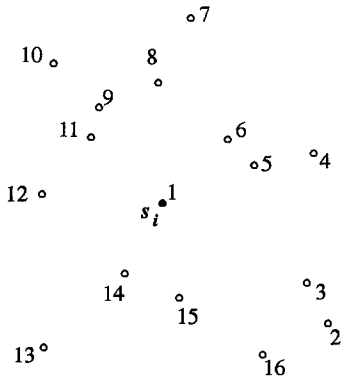


Fig. 3. The circular list around s_i.

θ_i is the polar angle of point p_i with respect to s, and r_i is the distance. However, this approach can be no faster than $O(n \log n)$ time per point. An elegant and optimal solution to the circular sorting problem in the plane was developed by Lee and Ching [9]. It employs the notion of geometric duality and arrangement of lines in the plane to compute the $O(n^2)$ entries of the ordered lists in $\Theta(n^2)$ total time. For details, the reader is referred to [9].

4.3. PSPM in Two Dimensions. The beauty of the results which use circular sorting as a generalization of sorting on the real line is that once we know how to solve a problem for a subset of $\mathbb{R}$ using a sweep technique, we are likely to solve that same problem in higher dimensions. It amounts to wrapping the algorithm for dimension one into $d-1$ layers of **for** loops spanning over the cardinality n of the input set. Each of these layers adds a factor of n (corresponding to one additional degree of freedom from one of the added dimensions) to the time complexity of the one-dimensional solution. Therefore, the complexity analysis is usually also straightforward.

For conciseness, we introduce the following notation. For each $s_j \in S$ the corresponding circular list of points obtained by the circular sorting of S is denoted $S_j = \{s_j = s_{j,1}, s_{j,2}, \ldots, s_{j,n}\}$. Corresponding to each of these lists, we introduce an ordering relation for the points of S as follows. Since these lists are circular, the ordering relations will include the specification of a starting point. For each $j = 1, 2, \ldots, n$, define $<_j^h$ by $s_i <_j^h s_l$ if and only if s_i precedes s_l in the sorted list S_j viewed as starting with the elements $s_{j,1} = s_j$ and $s_{j,2} = s_h$. See Figure 4. Furthermore, define the function $\mathbf{max}_j^h$ which returns the *index* of the maximum of its arguments *with respect to the ordering relation* $<_j^h$. Let $\mathcal{N}_j(s_i, s_l)$ denote the number of points in S_j between s_i and s_l in the counterclockwise direction.

Let s_{i1}, s_{i2}, s_{i3} and p_{l_1}, p_{l_2}, p_{l_3} be points. We denote by $\measuredangle(p_{l_1}, p_{l_2}, p_{l_3})$ the angle at p_{l_1} from p_{l_2} to p_{l_3}. Furthermore, if $d(s_{i_1}, s_{i_2}) = d(p_{l_1}, p_{l_2})$ we also write:

1. $(p_{l_1}, p_{l_2}, p_{l_3}) \approx (s_{i_1}, s_{i_2}, s_{i_3})$ if $\measuredangle(p_{l_1}, p_{l_2}, p_{l_3}) = \measuredangle(s_{i_1}, s_{i_2}, s_{i_3})$ and $d(p_{l_1}, p_{l_3}) = d(s_{i_1}, s_{i_3})$.[4]

[4] That is, the triangles $\Delta(p_{l_1}, p_{l_2}, p_{l_3})$ and $\Delta(s_{i_1}, s_{i_2}, s_{i_3})$ are congruent.

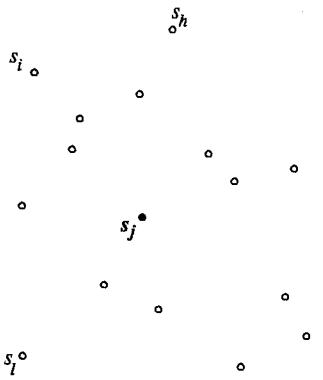


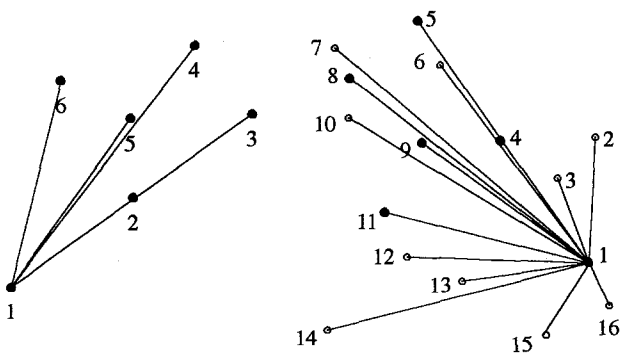
Fig. 4. Ordering relation $s_i <_j^h s_l$.

2. $(p_{l_1}, p_{l_2}, p_{l_3}) \prec (s_{i_1}, s_{i_2}, s_{i_3})$ if either $\measuredangle(p_{l_1}, p_{l_2}, p_{l_3}) < \measuredangle(s_{i_1}, s_{i_2}, s_{i_3})$ or $(\measuredangle(p_{l_1}, p_{l_2}, p_{l_3}) = \measuredangle(s_{i_1}, s_{i_2}, s_{i_3})$ and $d(p_{l_1}, p_{l_3}) < d(s_{i_1}, s_{i_3}))$.
3. $(p_{l_1}, p_{l_2}, p_{l_3}) \succ (s_{i_1}, s_{i_2}, s_{i_3})$ otherwise.

We now proceed to present the algorithm for the two-dimensional case of the PSPM problem. The algorithm works as follows. For each $s_j \in S$, attempt to find a match for P while anchoring p_1 at s_j. The attempt follows essentially the same procedure as algorithm PSPM_1 since it suffices to traverse the linear lists of points around p_1 and s_j. (See Figure 5.)

Algorithm PSPM_2

1. Circular sort the set S and sort the points of P lexicographically by polar coordinates (angle, norm) around the point p_1.
2. Let $j = 0$.
3. **repeat**
   3.1. [Anchor p_1 at s_j. (Recall that $S_j = \{s_j = s_{j,1}, s_{j,2}, \ldots, s_{j,n}\}$).]
   Let $j = j + 1$.



P=$\{p_1, p_2, p_3, p_4, p_5, p_6\}$ Match for P: $\{s_1, s_4, s_5, s_8, s_9, s_{11}\}$

Fig. 5. Illustrating PSPM_2.

3.2. [Assume each point is unique within its set.]
Initialize $m_i = i$ for $1 \leq i \leq k$.
3.3. [The first point always matches.]
Let $i = 1$.
3.4. **while** $i < k$ **and** $m_2 > 1$ **do**
3.4.1. [Get the next point in P.]
Let $i = i + 1$.
3.4.2. [To guarantee that m_i is ahead of m_{i-1} in case of an earlier failure.]
Let $m_i = \mathbf{max}_j^2\{m_i, m_{i-1} + 1\}$.
3.4.3. **while** $(\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_i) \succ (\boldsymbol{s}_{j,m_1}, \boldsymbol{s}_{j,m_2}, \boldsymbol{s}_{j,m_i})$ **and**
$\mathcal{N}_j(\boldsymbol{s}_{j,m_1}, \boldsymbol{s}_{j,m_2}) > k - i$ **do** [Must try further.]
Advance m_i.
3.4.4. [Is $\boldsymbol{s}_{j,m_i}$ a match to $\boldsymbol{p}_i$?]
Let success $= ((\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_i) \approx (\boldsymbol{s}_{j,m_1}, \boldsymbol{s}_{j,m_2}, \boldsymbol{s}_{j,m_i}))$.
3.4.5. **if not** success
then let $i = 1$ and advance m_2. [Restart from m_2.]
until $j = n$ **or** success
4. **if** success **then return** $(\{m_i | 1 \leq i \leq k\})$ [A match was found.]
else return $(\{\})$. [A match was found.]

The analysis of the algorithm is very similar to that of the one-dimensional case. Essentially, the only difference in this case is the outer loop whose purpose is to make the scanning of S through each of the n ordered lists produced by the circular sorting. It is immediate to see that the algorithm always halts and that if it returns a k-subset of S, that set must indeed be a match for P.

To see that if a match exists it will be found, let $\boldsymbol{s}_{j_1}, \ldots, \boldsymbol{s}_{j_k} \in S$ be a match for P with $\boldsymbol{s}_j$, corresponding to $\boldsymbol{p}_i$ for $i = 1, \ldots, k$. Assume that $\boldsymbol{s}_{j_1}, \ldots, \boldsymbol{s}_{j_k}$ is the first match (with respect to the ordering relation $<_{j_1}^2$) that can be found by traversing the list S_{j_1}. Unless another match for P is found first, at some point we will have $j = j_1$. This means that eventually $\boldsymbol{s}_{j,m_1} = \boldsymbol{s}_{j,1} = \boldsymbol{s}_{j_1}$. Once this happens, we are essentially dealing with just one of the ordered lists for S, namely, S_{j_1}. The argument, which is by (tedious but straightforward) induction on k, is basically one-dimensional and the result easily follows.

For the time-complexity analysis, first note that step 1 can be done in $O(n^2 + k \log k)$ time. Secondly, in the worst case, the point $\boldsymbol{p}_1 \in P$ may have to be anchored at each $\boldsymbol{s} \in S$. In each of these cases, since S_j is a circular list, m_i for $2 \leq i \leq k$ can advance to all of the remaining $n - 1$ indices of points of S at most once and never backtracks. Hence, the time complexity is bounded by $O(n^2 + k \log k + nk(n-1)) \subset O(kn^2)$.

Therefore we have established:

THEOREM 2. *Given a set P of k points and a set S of $n \geq k$ points in the plane, the PSPM problem can be solved in $O(kn^2)$ time.*

Since we are dealing here with PSPM[rotation + translation], we can obtain the isometry that maps P into S as follows. Once a match is found, the point of

S matching p_1 determines the translation transformation and the point matching p_2 subsequently determines the rotation transformation.

5. Circular Sorting in Higher Dimensions. Our aim in this section is to recall a generalization of circular sorting to arbitrary dimensions introduced in [8] and to present an optimal algorithm for its computation. The techniques we use are generalizations of those employed in [9], some of which had been developed by Edelsbrunner *et al.* in [6]. The time bound of our algorithm for circular sorting a set of n points in d-dimensional space will be shown to be $\Theta(n^d)$.

In [8] Goodman and Pollack present a notion of multidimensional sorting which they show to characterize a set with respect to its order type. Their characterization, called the λ-matrix, was later shown to be computable in $\Theta(n^d)$ time in [6]; see also [7]. The interested reader can verify, however, that while being computable in the same amount of time as circular sorting, the λ-matrix does not contain enough information about the spatial structure of the set to lead to a fast traversal scheme needed here.

5.1. Affine Geometry and Geometric Duality. In this section we very briefly discourse on the subject of geometric duality in preparation for studying circular sorting in high dimensions. For further details, the reader is referred to [4]–[6] and [9].

If A_1 and A_2 are affine spaces (possibly of different dimensions) we write $A_1 \| A_2$ to denote that A_1 and A_2 are parallel.

DEFINITION 2. If $U \subset \mathbb{R}^d$ is any subset, its affine span $\langle U \rangle$ is defined by

$$\langle U \rangle = \left\{ \sum_{i=1}^{k} a_i \boldsymbol{u}_i \,\middle|\, \boldsymbol{u}_i \in U, \sum_{i=1}^{k} a_i = 1 \right\}.$$

Denote $\mathbb{R}^d \backslash \{\mathbf{0}\}$ by $\mathbb{R}^d_*$, where $\mathbf{0}$ is the origin of $\mathbb{R}^d$. Let $\mathscr{H}^d$ be the set of all hyperplanes of $\mathbb{R}^d$ not containing $\mathbf{0}$.

A point $\boldsymbol{u} = (u_1, u_2, \ldots, u_d) \in \mathbb{R}^d_*$ determines a unique hyperplane $u^\perp$ given by $\{\boldsymbol{x} \in \mathbb{R}^d | \boldsymbol{u} \cdot \boldsymbol{x} + 1 = 0\}$, where $\boldsymbol{u} \cdot \boldsymbol{x} \overset{\text{def}}{=} \sum_{i=1}^{d} u_i x_i$ is the scalar product. Geometrically, $u^\perp$ can be characterized by the following three conditions:

1. The vector $\overrightarrow{\mathbf{0}\boldsymbol{u}}$ is normal to $\boldsymbol{u}^\perp$,
2. The distance $d(\mathbf{0}, \boldsymbol{u}^\perp)$ from $\boldsymbol{u}^\perp$ to $\mathbf{0}$ is $|\boldsymbol{u}|^{-1}$.
3. The point $\boldsymbol{u}$ is contained in the same d-dimensional half-space of $\mathbb{R}^d \backslash \{\boldsymbol{u}^\perp\}$ as $\mathbf{0}$.

Consider now the following map, called the *geometric duality transform*:

$$\Delta: \mathbb{R}^d_* \to \mathscr{H}^d$$
$$\boldsymbol{u} \mapsto \boldsymbol{u}^\perp.$$

We call $\boldsymbol{u}^\perp$ the *geometric dual* (or simply *dual*) of $\boldsymbol{u}$. This map is easily seen to be a bijection and for this reason we write, without confusion, $\Delta^{-1}(H) = H^\perp$ for any $H \in \mathscr{H}^d$. Hence, $\boldsymbol{u} = (\boldsymbol{u}^\perp)^\perp$ for all $\boldsymbol{u} \in \mathbb{R}^d_*$.

We make use of properties of the geometric duality transform stated in the following lemmas whose proofs, being straightforward, are left to the reader.

LEMMA 1. *Let $\boldsymbol{p}_1, \boldsymbol{p}_2$ be points in $\mathbb{R}^d_*$. Then $\boldsymbol{p}_2 \in \langle \mathbf{0}, \boldsymbol{p}_1 \rangle$ if and only if $\boldsymbol{p}_1^\perp \| \boldsymbol{p}_2^\perp$.*

LEMMA 2. *Let $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_{d-1}$ be affinely independent points in $\mathbb{R}^d_*$ such that $\mathbf{0} \notin \langle p_1, p_2, \ldots, p_{d-1} \rangle$ and let*

$$l = \bigcap_{1 \le j \le d-1} \boldsymbol{p}_j^\perp$$

Then:

1. *l is a line.*
2. *$\boldsymbol{p} \in \boldsymbol{p}_1^\perp$ if and only if $\boldsymbol{p}_1 \in \boldsymbol{p}^\perp$.*
3. *$\bigcap_{\boldsymbol{p} \in \langle \boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_{d-1} \rangle} \boldsymbol{p}^\perp = l$.*
4. *$\boldsymbol{p} \in \langle \mathbf{0}, \boldsymbol{p}_1, \boldsymbol{p}_2, \cdots, \boldsymbol{p}_{d-1} \rangle \backslash \langle \boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_{d-1} \rangle$ if and only if $l \cap \boldsymbol{p}^\perp = \{\}$.*
5. *Let $\boldsymbol{p} \notin \langle \mathbf{0}, \boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_{d-1} \rangle$. Then*

$$\langle \boldsymbol{p}, \boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_{d-1} \rangle^\perp = \boldsymbol{p}^\perp \cap l.$$

In a more verbose way, this lemma states that:

1. l has dimension one.
2. A point $\boldsymbol{p}$ lies on a hyperplane $P \in \mathscr{H}^d$ if and only if $P^\perp$ lies on $\boldsymbol{p}^\perp$.
3. The intersection of $\boldsymbol{p}^\perp$ for all $\boldsymbol{p}$ in the $(d-2)$-flat $\langle \boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_{d-1} \rangle$ is the line l.
4. The hyperplane $\langle \mathbf{0}, \boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_{d-1} \rangle$ defined by $\mathbf{0}, \boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_{d-1}$ is the locus of points whose geometric duals are parallel to l.
5. The dual of the hyperplane through d points is the intersection point of the dual (hyperplanes) of those points.

LEMMA 3. *Let L be a line in $\mathbb{R}^d$ not passing through the origin and let $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ be two distinct points on L. Then the dual of a point $\boldsymbol{p}$ moving on L is a hyperplane $\boldsymbol{p}^\perp$ which rotates around the $(d-2)$-flat $\boldsymbol{p}_1^\perp \cap p_2^\perp$.*

5.2. Circular Sorting in d-Dimensions. Given a set of n points $S = \{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_n\} \subset \mathbb{R}^d$, consider the problem of finding, for each affinely independent $(d-1)$-tuple $T = (\boldsymbol{s}_{i_1}, \boldsymbol{s}_{i_2}, \ldots, \boldsymbol{s}_{i_{d-1}})$ of points of S, the ordering of the points in $S \backslash T$ around the $(d-2)$-dimensional affine subspace $\langle T \rangle$. The ordering can be viewed as that in which the points of $S \backslash T$ are encountered by a half-hyperplane anchored at $\langle T \rangle$ as it rotates around $\langle T \rangle$. Whenever two or more distinct points of $S \backslash T$ lie on a half-hyperplane containing $\langle T \rangle$, their ordering is obtained within the affine

subspace of dimension $d-1$ spanned by T and (any two of) those points. This is called the *circular sorting* problem in d-dimensional space.

Note that while the ordering so defined has a recursive flavor, we show that the degenerate cases are dealt with in a more efficient nonrecursive way.

The direct approach for solving this problem amounts to spending at least $O(n^d \log n)$ time, i.e., $O(n \log n)$ sorting time for each of $\binom{n}{d-1}$ $(d-1)$-tuples.

Let us now consider applying the duality transformation to an instance of the problem. Given a set $S = \{s_1, s_2, \ldots, s_n\} \subset \mathbb{R}^d_*$ the set of dual hyperplanes of its points determines a partition of the d-dimensional Euclidean space called an *arrangement*. A representation of this arrangement, referred to as a *cell complex*, in the form of an efficient data structure can be constructed in $\Theta(n^d)$ time [6]. It consists of vertices, edges, and faces (k-flats for $0 \le k \le d$) together with their incidence relations, which are determined by the intersections of the dual hyperplanes.

We now describe a mechanism for extracting the circular sorted lists from this arrangement and show that the time needed is proportional to the size of the lists and hence optimal.

We assume without loss of generality that (some $(d+1)$-subset of) the set S affinely spans $\mathbb{R}^d$. Otherwise, the circular sorting is performed in the Euclidean space spanned by S.

Let $s_{i_1}, s_{i_2}, \ldots, s_{i_{d-1}}$ be any $d-1$ affinely independent points in S. It suffices to describe how the sorted list of $n-(d-1)$ points in $S \backslash \{s_{i_1}, s_{i_2}, \ldots, s_{i_{d-1}}\}$ can be constructed in $O(n)$ time after the cell complex defined by the dual hyperplanes has been obtained.

Assume for simplicity that the coordinates of all points in S are positive. Furthermore, we place the origin of the coordinate axes so that it is not contained in any hyperplane defined by any d points of S. This can be achieved in $O(n^d)$ time by placing the origin on the dth coordinate axes below the lowest of its intersections with all the hyperplanes spanned by any d points.

Let $\pi(s_{i_1}, s_{i_2}, \ldots, s_{i_{d-1}})$ be the hyperplane containing the $(d-2)$-flat $\langle s_{i_1}, s_{i_2}, \ldots, s_{i_{d-1}} \rangle$ and perpendicular to the hyperplane $\langle \mathbf{0}, s_{i_1}, s_{i_2}, \ldots, s_{i_{d-1}} \rangle$. This pair of hyperplanes divides the Euclidean d-space into four quadrants I, II, III, and IV and hence the points in $S \backslash \langle s_{i_1}, s_{i_2}, \ldots, s_{i_{d-1}} \rangle$ are divided into four subsets. By Lemma 3, if the hyperplane $\langle \mathbf{0}, s_{i_1}, s_{i_2}, \ldots, s_{i_{d-1}} \rangle$ rotates around $\langle s_{i_1}, s_{i_2}, \ldots, s_{i_{d-1}} \rangle$ its dual will move along

$$L_{i_1, i_2, \ldots, i_{d-1}} = \bigcap_{1 \le j \le d-1} s_{i_j}^{\perp}$$

from infinity to infinity.

By Lemma 2(5), the dual of each point $s \notin \langle \mathbf{0}, s_{i_1}, s_{i_2}, \ldots, s_{i_{d-1}} \rangle$ intersects $L_{i_1, i_2, \ldots, i_{d-1}}$ in exactly one point. If we visit the intersection points on $L_{i_1, i_2, \ldots, i_{d-1}}$, we can determine the ordering of the points in $S \backslash \{s_{i_1}, s_{i_2}, \ldots, s_{d-1}\}$. To do this, we maintain four lists denoted $S_{i_1, i_2, \ldots, i_{d-1}}(q)$ for $q = \text{I, II, III, IV}$, each corresponding to one of the four quadrants.

The lists are initialized to be empty. To each intersection point $\boldsymbol{u}$ which is traversed, there corresponds (at least) one intersecting hyperplane. In case there is exactly one dual hyperplane through $\boldsymbol{u}$, say $\boldsymbol{s}_l^{\perp}$, we determine the quadrant in which $\boldsymbol{s}_l$ lies and append it to the corresponding list.

We now deal with the case of multiple intersecting hyperplanes. It follows from Lemma 2(5) that this case happens whenever a hyperplane H *not* through the origin containing $\langle \boldsymbol{s}_{i_1}, \boldsymbol{s}_{i_2}, \ldots, \boldsymbol{s}_{i_{d-1}} \rangle$ passes through two or more other points. Suppose $\boldsymbol{s}_{l_1}, \boldsymbol{s}_{l_2}, \ldots, \boldsymbol{s}_{l_k}$ are on H, for some $k > 1$. Then the dual hyperplanes $\boldsymbol{s}_{l_r}^{\perp}$, $r = 1, 2, \ldots, k$, will intersect $L_{i_1, i_2, \ldots, i_{d-1}}$ at a point $\boldsymbol{u} = H^{\perp}$ (see Lemma 2(5) again). We order these points around $\boldsymbol{s}_{i_1}, \boldsymbol{s}_{i_2}, \ldots, \boldsymbol{s}_{d-2}$ within the hyperplane H in the form of circular sorting in $d-1$ dimensions.

Denote by $L_{i_1, i_2, \ldots, i_{d-2}, l_r}$ the intersection line of $\bigcap_{1 \le j \le d-2} \boldsymbol{s}_{i_j}^{\perp}$ and $\boldsymbol{s}_{l_r}^{\perp}$ for $r = 1, 2, \ldots, k$. (Note that $L_{i_1, i_2, \ldots, i_{d-1}}$ is like one of these.) Since these lines are all contained in the (*two-dimensional*) *plane* $\bigcap_{1 \le j \le d-2} \boldsymbol{s}_{i_j}^{\perp}$, it makes sense to talk about their counterclockwise order around $\boldsymbol{u}$. So, assume that $L_{i_1, i_2, \ldots, i_{d-1}}$ and $L_{i_1, i_2, \ldots, i_{d-2}, l_r}$ occur in counterclockwise order around $\boldsymbol{u}$ such that $L_{i_1, i_2, \ldots, i_{d-2}, l_t}$ follows $L_{i_1, i_2, \ldots, i_{d-2}, l_{t-1}}$, for $t = 1, 2, \ldots, k+1$, where $L_{i_1, i_2, \ldots, i_{d-2}, l_0} = L_{i_1, i_2, \ldots, i_{d-2}, l_{k+1}} = L_{i_1, i_2, \ldots, i_{d-1}}$. For each r from 1 to k, the point $\boldsymbol{s}_{l_r}$ will be appended to the list $S_{i_1, i_2, \ldots, i_{d-1}}(q)$ depending on which quadrant q (of I, II, III, IV) $\boldsymbol{s}_{l_r}$ lies on. Note that only two quadrants are involved each time, namely, I and III or II and IV.

The only case not yet considered is that of the points lying on the plane $\langle \boldsymbol{0}, \boldsymbol{s}_{i_1}, \boldsymbol{s}_{i_2}, \ldots, \boldsymbol{s}_{i_{d-1}} \rangle$. However, this case cannot occur because of the placement of the origin chosen *a priori*.

Hence, the following has been established.

THEOREM 3. *Given a seat* $S = \{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_n\}$ *of points in the d-dimensional Euclidean space, the circular sorting problem, i.e., determining the ordering of the points in* $S \backslash \{\boldsymbol{s}_{i_1}, \boldsymbol{s}_{i_2}, \ldots, \boldsymbol{s}_{i_{d-1}}\}$ *around the* $(d-2)$*-flat* $\langle \boldsymbol{s}_{i_1}, \boldsymbol{s}_{i_2}, \ldots, \boldsymbol{s}_{i_{d-1}} \rangle$, *for* $i_1, i_2, \ldots, i_{d-1} = 1, 2, \ldots, n$, *can be solved in* $\Theta(n^d)$ *time.*

6. PSPM in Higher Dimensions. Let S be a set of n points in $\mathbb{R}^d$. Assume that the origin $\boldsymbol{0}$ is not in S and that the points of S affinely span[5] $\mathbb{R}^d$.

We now extend to higher dimensions, a notation introduced before for dimension two. Our aim in doing so is to attain enough similarity between the two cases so as to substantially simplify the description and the complexity analysis of the algorithm.

For each affinely independent $(d-1)$-tuple $\mathscr{T} = (\boldsymbol{s}_{j_1}, \boldsymbol{s}_{j_2}, \ldots, \boldsymbol{s}_{j_{d-1}})$ of points of S the corresponding circular list of points obtained by the circular sorting of S is denoted by $S_{\mathscr{T}}$. Denote the ith element of this list by $S_{\mathscr{T}}(i)$. Corresponding to each of these lists, we introduce an ordering relation for the points of S as follows. Since these lists are circular, the ordering relations include the specification

[5] If the space spanned by the points of S has dimension $d' < d$, we perform a change of coordinates so as to work on $\mathbb{R}^{d'}$.

of a starting point. Define $<^h_{\mathscr{T}}$ by $s_i <^h_{\mathscr{T}} s_l$ if and only if s_i precedes s_l in the sorted list $S_{\mathscr{T}}$ viewed as starting with the elements $S_{\mathscr{T}}(l) = s_{j_i}$ for $i = 1, 2, \ldots, d-1$, and $S_{\mathscr{T}}(d) = s_h$. Furthermore, define the function $\mathbf{max}^h_{\mathscr{T}}$ which returns the *index* of the maximum of its arguments *with respect to the ordering relation* $<^h_{\mathscr{T}}$. Let $\mathscr{N}_{\mathscr{T}}(s_i, s_l)$ denote the number of points in $S_{\mathscr{T}}$ between s_i and s_l.

Lastly, given two $(d+1)$-tuples of affinely independent points, $(s_{i_1}, s_{i_2}, \ldots, s_{i_{d+1}})$ and $(p_{l_1}, p_{l_2}, \ldots, p_{l_{d+1}})$ such that the $(d-1)$-simplex[6] $[s_{i_1}, s_{i_2}, \ldots, s_{i_d}]$ is congruent to the $(d-1)$-simplex $[p_{l_1}, p_{l_2}, \ldots, p_{l_d}]$ we define $(p_{l_1}, p_{l_2}, \ldots, p_{l_{d+1}}) \approx (s_{i_1}, s_{i_2}, \ldots, s_{i_{d+1}})$ as follows.

Denote by $\measuredangle(p_{l_1}, p_{l_2}, \ldots, p_{l_{d+1}})$ the angle between the hyperplanes $\langle p_{l_1}, p_{l_2}, \ldots, p_{l_{d-1}} p_{l_d}\rangle$ and $\langle p_{l_1}, p_{l_2}, \ldots, p_{l_{d-1}} p_{l_{d+1}}\rangle$. We recursively define:

1. $(p_{l_1}, p_{l_2}, \ldots, p_{l_{d+1}}) \approx (s_{i_1}, s_{i_2}, \ldots, s_{i_{d+1}})$

   if $\begin{cases} \measuredangle(p_{l_1}, p_{l_2}, \ldots, p_{l_{d+1}}) = \measuredangle(s_{i_1}, s_{i_2}, \ldots, s_{i_{d+1}}) \text{ and} \\ (p_{l_1}, p_{l_2}, \ldots, p_{l_{d-1}}, p_{l_{d+1}}) \approx (s_{i_1}, s_{i_2}, \ldots, s_{i_{d-1}}, s_{i_{d+1}}).^7 \end{cases}$

2. $(p_{l_1}, p_{l_2}, \ldots, p_{l_{d+1}}) \prec (s_{i_1}, s_{i_2}, \ldots, s_{i_{d+1}})$

   if $\begin{cases} \text{either} \\ \measuredangle(p_{l_1}, p_{l_2}, \ldots, p_{l_{d+1}}) < \measuredangle(s_{i_1}, s_{i_2}, \ldots, s_{i_{d+1}}) \\ \text{or} \\ \measuredangle(p_{l_1}, p_{l_2}, \ldots, p_{l_{d+1}}) = \measuredangle(s_{i_1}, s_{i_2}, \ldots, s_{i_{d+1}}) \text{ and} \\ (p_{l_1}, p_{l_2}, \ldots, p_{l_{d-1}}, p_{l_{d+1}}) \prec (s_{i_1}, s_{i_2}, \ldots, s_{i_{d-1}}, s_{i_{d+1}}). \end{cases}$

3. $(p_{l_1}, p_{l_2}, \ldots, p_{l_{d+1}}) \succ (s_{i_1}, s_{i_2}, \ldots, s_{i_{d+1}})$ otherwise.

We now proceed to present the algorithm for the d-dimensional case of the PSPM problem.

Algorithm PSPM$_d$

1. Sort S in the sense of d-dimensional circular sorting described in Section 5. Fix $\mathscr{P}$ as an affinely independent $(d-1)$-tuple of points in P. Without loss of generality, assume $\mathscr{P} = \{p_1, p_2, \ldots, p_{d-1}\}$. Sort the points of P circularly around the $(d-2)$-flat $\langle \mathscr{P} \rangle$, only.
2. Of all affinely independent $(d-1)$-tuples of S, let $\mathscr{T}_j$ for $j = 1, 2, \ldots, N$ denote all those which are congruent to $\mathscr{P}$.
   Let $j = 0$.
3. **repeat**
   3.1. [Anchor $\mathscr{P}$ at $\mathscr{T}_j$.]
   Let $j = j + 1$.
   3.2. [Assume each point is unique within its set.]
   Initialize $m_i = i$ for $1 \le i \le k$.

---

[6] A d-simplex in d-dimensional space is the convex hull of an affinely independent set of $d+1$ points. For example, a triangle in the plane, a tetrahedron in Euclidean space, etc.

[7] That is, the d-simplices $\Delta(p_{l_1}, p_{l_2}, \ldots, p_{l_{d+1}})$ and $\Delta(s_{i_1}, s_{i_2}, \ldots, s_{i_{d+1}})$ are congruent.

3.3. [The points of $\mathscr{P}$ are already known to match.]
Let $i = d - 1$.

3.4. **while** $i < k$ **and** $m_d > 1$ **do**

3.4.1. Get the next point in P.]
Let $i = i + 1$.

3.4.2. [To guarantee that m_i is ahead of m_{i-1} in case of an earlier failure.]
Let $m_i = \mathbf{max}^d_{\mathscr{T}_j}\{m_i, m_{i-1} + 1\}$.

3.4.3. **while** $(p_1, p_2, \ldots, p_{d-1}, p_i) \succ (S_{\mathscr{T}_j}(m_1), S_{\mathscr{T}_j}(m_2), \ldots, S_{\mathscr{T}_j}(m_{d-1}), S_{\mathscr{T}_j}(m_i))$.
and $\mathscr{N}_{\mathscr{T}_j}(S_{\mathscr{T}_j}(m_i), S_{\mathscr{T}_j}(m_d)) > k - i$ **do** [Must try further.]
Advance m_i.

3.4.4. [Is $S_{\mathscr{T}_j}(m_i)$ a match to p_i?]
Let success =
$((p_1, p_2, \ldots, p_{d-1}, p_i) \approx (S_{\mathscr{T}_j}(m_1), S_{\mathscr{T}_j}(m_2), \ldots, S_{\mathscr{T}_j}(m_{d-1}), S_{\mathscr{T}_j}(m_i)))$.

3.4.5. **if not** success
then let $i = d - 1$ and advance m_d, [Restart from m_d.]

until $j = N$ **or** success

4. **if** success **then return** $(\{m_i | 1 \le i \le k\})$ [A match was found.]
else return $(\{\})$. [A match was not found.]

The analysis of the algorithm is very similar, although much more arduous due to the complexity of the required notation, to that of the two-dimensional case. Essentially, in this case the outer loop was modified so that the distinguished $(d-1)$-tuple $\mathscr{P}$ can be made to anchor on each and every one of the N $(d-1)$-tuples of S congruent to $\mathscr{P}$. It can be verified that the algorithm always halts and that if it returns a k-subset of S, that set must indeed be a match for P.

As before, to see that if a match exists it will be found, let $s_{j_1}, \ldots, s_{j_k} \in S$ be a match for P with s_{j_i} corresponding to p_i for $i = 1, \ldots, k$. Clearly, $\mathscr{U} = \{s_{j_1}, \ldots, s_{j_{d-1}}\}$ is affinely independent (since it is a match for $\mathscr{P}$). Let $\mathscr{T}_{j_1} = \mathscr{U}$. Assume that $s_{j_1}, \ldots, s_{j_k}$ is the first match (with respect to the ordering relation $<^d_{\mathscr{T}_{j_1}}$) that can be found by traversing the list $S_{\mathscr{T}_{j_1}}$. Unless another match for P is found first, we will eventually have $j = j_1$. This means that $\mathscr{P}$ is eventually mapped to $\mathscr{T}_{j_1} = \mathscr{U}$. Once this happens, we are essentially dealing with just one of the ordered lists for S, namely, $S_{\mathscr{T}_{i_1}}$. The argument, which is again by induction on k, is basically one-dimensional and the result follows.

To analyze the time complexity, first note that $N \in O(n^{d-1})$ since there can be up to $\binom{n}{d-1}$ $(d-1)$-tuples which are congruent to $\mathscr{P}$.[8] For each one of these, the algorithm behaves as in the one-dimensional case in the sense that it has to traverse one totally ordered list of length n. The complexity of each of these

[8] It is immaterial whether all of the N $(d-1)$-tuples $\mathscr{T}_j$ congruent to $\mathscr{P}$ are precomputed in step 2 or (more naturally) are computed one at a time within the outer **repeat** loop (in steps 3.1–3.3).

traversals is, as we have seen before, $O(kn)$. Hence, we have a total time complexity of $O(kn^d)$.

Therefore, we have established:

THEOREM 4. *Given a set P of k points and a set S of $n \geq k$ points in the d-dimensional Euclidean space, the PSPM problem can be solved in $O(kn^d)$ time.*

7. Scaling. We now consider the possibility of the sets of points undergoing global scaling. Actually, it suffices to consider that *one* of the sets be scaled. There are two cases to consider. The first is when the scaling factor is known *a priori*. This, which is the simpler case, imposes a minor change in the algorithms of no relevance to the time complexities. It amounts to, whenever comparing the distance between two points of the pattern with that of two points of the sampling set, multiplying the appropriate one of the two by the scaling factor.

The second case, in which the scaling factor is *not* known *a priori*, imposes that one of the sets, say, the pattern, be allowed to undergo arbitrary scaling (shrinking or stretching). This can be easily incorporated into our algorithms. Whenever the first two points $\boldsymbol{p}_1, \boldsymbol{p}_2$ of the pattern have to be matched against *any* two points $\boldsymbol{s}_i, \boldsymbol{s}_j$ of the sampling set, they should be considered a match with the scaling factor equal to the quotient of their corresponding distances:

$$\frac{d(\boldsymbol{p}_1, \boldsymbol{p}_2)}{d(\boldsymbol{s}_i, \boldsymbol{s}_j)}.$$

This establishes the value of the scaling factor to be employed for all the other points. As far as the time complexities of the algorithms for dimensions $d \geq 2$ are concerned, there is no change in the worst-case performance since each pair of points from the sampling set is already compared with $\boldsymbol{p}_1, \boldsymbol{p}_2$ in the case of isometries. (A factor of n is imposed to the complexity of the one-dimensional case, though.)

8. Concluding Remarks. Regarding the PSPM problem, some related questions not studied in this paper can be formulated.

For instance, consider the case in which not all points in the pattern set find a match in the sampling set. Two problems can be posed: one where the maximum number of mismatches is part of the input and another where that is not the case. In the latter, a measure of "goodness" needs to be introduced, such as the ratio between the number of points in the pattern and the number of points which actually find match. Another measure of goodness may be incorporated to determine the total deviation of an approximate match, or the maximum such deviation.

Also, it remains an open question whether the PSPM problem in the real line can be done in time $o(k(n-k))$ after sorting. If this was possible, the higher-dimensional cases are likely to be improved as well.

Acknowledgments. The authors are grateful to Bernard Chazelle for bringing to their attention the problem, which he called the *constellation problem*, and to the referees for helpful suggestions on how the paper could be improved.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA.

[2] H. Alt, K. Mehlhorn, H. Wagener, and E. Welzl, Congruence, Similarity and Symmetries of Geometric Objects, *Discrete Comput. Geom.*, **3** (1988), 237–256.

[3] M. D. Atkinson, An Optimal Algorithm for Geometrical Congruence, *J. Algorithms*, **8** (1987), 159–172.

[4] B. M. Chazelle, L. J. Guibas, and D. T. Lee, The Power of Geometric Duality, *BIT*, **25** (1985), 76–90.

[5] P. J. de Rezende, Point Set Pattern Matching in d-Dimensions, Ph.D. Dissertation, Northwestern University, Evanston, IL, 1988.

[6] H. Edelsbrunner, J. O'Rourke, and R. Seidel, Constructing Arrangements of Lines and Hyperplanes with Applications, *SIAM J. Comput.*, **15** (1986), 341–363.

[7] H. Edelsbrunner, R. Seidel, and M. Sharir, On the Zone Theorem for Hyperplane Arrangements, *SIAM J. Comput.*, to appear.

[8] J. E. Goodman and R. Pollack, Multidimensional Sorting, *SIAM J. Comput.*, **12** (1983), 484–507.

[9] D. T. Lee and Y. T. Ching, The Power of Geometric Duality Revisited, *Inform. Process. Lett.*, **21** (1985), 117–122.

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