Even faster point set pattern matching in 3-d

Laurence Boxer
Niagara University and State University of New York at Buffalo. *

Robert Haralick
University of Washington. †

Abstract

Recent papers concerned with the Point Set Pattern Matching Problem (PSPM) (finding all congruent copies of a pattern in a sample set) in Euclidean 3-space, R^3, have given algorithms with running times that have decreased as known output bounds for the problem have decreased. In this paper, a recent result of [2] is used to show that the volume of the output is $O(kn^{1.8}[\log^* n]^{O(1)})$, where k is the size of the pattern and n is the size of the sample set. This is a smaller output bound than was previously known, and makes possible faster running times than were previously known for this problem. We show that with mild additional restrictions on the input (roughly, that the sample set is not too dense or the pattern set is not too small relative to the sample set), our solutions have running times bounded by the time to sort the volume of data given by our output bounds, on sequential computers and on a coarse-grained multicomputer (CGM). Solutions to related problems are also discussed.

Keywords: Point Set Pattern Matching, congruence, similarity, analysis of algorithms

1 Introduction

"Point Set Pattern Matching" is a geometric pattern recognition problem in which we seek to identify a (all) representation(s) of a pattern set P in a sampling set S, where P and S are subsets of a Euclidean space R^d. Variations of the problem may require a "representation" of P in S to be P itself, as a subset of S; a translation of P; a rotation of P; congruent, or similar, to P; approximately congruent to P; a maximal cardinality subset of P that is congruent or similar to a subset of S; etc. Among the papers in which such variants of the Point Set Pattern Matching problem have been studied are [2, 3, 4, 5, 6, 7, 9, 10, 11, 14, 16, 17, 19, 20].

For the exact matching (*i.e.*, finding all congruent copies of) version of the PSPM Problem, the fastest sequential algorithms in the literature are the following.

*Department of Computer and Information Sciences, Niagara University, Niagara University, NY 14109, USA. e-mail: boxer@niagara.edu. Research supported by grant from Niagara University Research Council.

†Department of Electrical Engineering, FT-10 University of Washington, Seattle, Washington 98195, USA. e-mail: haralick@ee.washington.edu

- For $d = 2$, the algorithm of [16].
- For $d = 1$ and for $d > 3$, the algorithms of [14].
- For $d = 3$, the algorithm of [5] has a running time of $O(kn^{2.5}[\lambda_6(n)/n]^{0.25} \log n)$. A faster result is claimed in [6], but it depends on Lemma 3.2 of [6], the proof of which is incorrect (we thank Peter Braß for pointing out the error).

In the current paper, we show that a result of [2] yields a smaller upper bound for the volume of output to the PSPM Problem in R^3 than was previously known. This allows us to produce an algorithm faster than that of [5], the best previous algorithm for the problem (also faster, in some cases, than the questionable time claimed in [6]). We further show that under mild additional restrictions on the input, our algorithm runs within a factor of $\log n$ of the output bound.

An implementation of our algorithm is given for the coarse-grained multicomputer (CGM) model of computation. We also give sequential and CGM solutions to the *all similarities* version of the PSPM Problem in R^3.

The paper is organized as follows.

- In section 2, we define our problem, introduce relevant combinatorial results, and give an efficient algorithm for the removal of duplicate entries from a list of lists.
- In section 3, we give bounds on the worst case output for the problem.
- In section 4, we present an algorithm for Point Set Pattern Matching in R^3 that can be used for both the all-congruences and the all-similarities version of the problem. The algorithm is stated in a high-level, architecture-independent fashion.
- In section 5, we analyze our algorithm for the sequential (RAM) model of computation, for both the all-congruences and the all-similarities versions of the problem.
- In section 6, we analyze our algorithm for the CGM model of computation, for both the all-congruences and the all-similarities versions of the problem.
- In section 7, we give some concluding remarks.

A preliminary version of this paper appeared in [7]. The current paper includes several improvements on [7].

2 Preliminaries

2.1 The problem

Let P and S be finite subsets of R^3, $|P| = k \leq n = |S|$. We wish to identify every subset P' of S such that P' is congruent to P. Alternately, we may wish to identify every subset P' of S such that

P' is similar to P. We will abuse notation by saying $(x_1, \ldots, x_k)$ is congruent (respectively, similar) to $(y_1, \ldots, y_k)$ when we properly would say $\{x_1, \ldots, x_k\}$ is congruent (respectively, similar) to $\{y_1, \ldots, y_k\}$.

We assume the real RAM model of computation, as described in [18]. In section 6, in which we use parallel computers, we assume every processor conforms to this model. In order to make exact matches, we assume trigonometric functions and square roots (for computation of the Euclidean distance) can be computed exactly in $\Theta(1)$ time.

2.2 Davenport-Schinzel sequences

Let $t_2(k)$ be the tower function defined for nonnegative integers by

$$t_2(k) = \begin{cases} 1 & \text{if } k = 0; \\ 2^{t_2(m)} & \text{if } k = m + 1. \end{cases}$$

Recall $\log^* n$ is the minimal k such that $t_2(k) \geq n$.

A *Davenport-Schinzel sequence* is a sequence of symbols in which long subsequences of alternating symbols are not permitted. In particular, let $A = \{a_1, \ldots, a_n\}$ be an alphabet of n distinct symbols and let $\lambda_s(n)$ be the maximum length of a sequence e of members of A such that e has

- no pair of consecutive members that are identical, and
- no subsequence of length $s + 2$ consisting of two alternating members of A.

Let $\alpha(n)$ be the extremely slowly growing inverse of Ackermann's function. As a case of a more general result of [1], we have the following.

Proposition 2.1 $\lambda_6(n) = \Theta(n \cdot 2^{\Theta([\alpha(n)]^2)}) = O(n \log^* n)$. ■

2.3 Unit distances in Euclidean spaces

For $j \in \{2, 3\}$, let $D_j(n)$ be the maximal number, over all sets $S \subset R^j$ such that $|S| = n$, of line segments of unit length with endpoints in S.

Proposition 2.2

- [15, 21] *There is a constant $c > 0$ such that*

$$n^{1+c/\log\log n} \leq D_2(n) = O(n^{4/3}).$$

- [12] $D_3(n) = O(n^{1.5}[\frac{\lambda_6(n)}{n}]^{0.25})$. ■

We note also that $D_2(n) \leq D_3(n)$.

2.4 Density properties

We will show that minor modifications of our general algorithm make it possible to solve the PSPM in R^3 more efficiently if we can assume the sample set S is not too dense. In this paper, we assume the use of the Euclidean metric for R^3: for $\{p_0, p_1\} \subset R^3$, $p_i = (x_i, y_i, z_i)$,

$$d(p_0, p_1) \;=\; [(x_1 - x_0)^2 \;+\; (y_1 - y_0)^2 \;+\; (z_1 - z_0)^2]^{0.5}.$$

The *diameter*, denoted $diam(X)$, of a set of points X in a Euclidean space is the maximal distance with respect to the Euclidean metric between members of X. In particular,

$$diam(\{p_0, p_1, p_2\}) \;=\; \max\{d(p_0, p_1), d(p_0, p_2), d(p_1, p_2)\}.$$

We denote by $\delta_2(n)$ and $\delta_3(n)$ functions that we think of as density measures for the set S that are determined by P. These functions are used as asymptotic bounds on the density of S with respect to a coordinate (arbitrarily chosen as the $x-$coordinate) and a closest pair (respectively, a minimal-diameter triangle) with endpoints in P. We assume our density measures satisfy the following properties.

- **(Segment Density Property)** Let $\{p_r, p_s\} \subset P$ be a closest (with respect to the Euclidean metric) pair in P, *i.e.*, a pair of distinct members of P such that $d_2(P) \;=\; d(p_r, p_s)$, where we define

  $$d_2(P) \;=\; \min\{d(a, b) \mid a, b \text{ are distinct members of } P\}.$$

  Then for each i such that $0 \le i < n$,

  $$|\{j \;:\; |x(s_i) - x(s_j)| \le d_2(P)\}| \;=\; O(\delta_2(n)),$$

  where $x(p)$ is the x-coordinate of p.

- **(Triangle Density Property)** Let $\{p_u, p_v, p_w\} \subset P$ be a minimal-diameter (with respect to the Euclidean metric) non-collinear triple in P, *i.e.*, a non-collinear set such that $d_3(P) \;=\; diam(\{p_u, p_v, p_w\})$, where we define

  $$d_3(P) \;=\; \min\{diam(\{a, b, c\}) \mid a, b, \text{ and } c \text{ are distinct non-collinear members of } P\}.$$

  Then for each i such that $0 \le i < n$,

  $$|\{j \;:\; |x(s_i) - x(s_j)| \le d_3(P)\}| \;=\; O(\delta_3(n)).$$

Notice that these properties are always true for $\delta_2(n) \;=\; \delta_3(n) \;=\; n$. Therefore, we will assume that $\delta_2(n) \;=\; O(n)$, $\delta_3(n) \;=\; O(n)$.

2.5 Elimination of duplicates

In a list of sets of the same size, duplicates can be eliminated efficiently by the algorithm of the following proposition.

Proposition 2.3 *Let X be a well-ordered set such that for $\{x_0, x_1\} \subset X$, it can be decided in $\Theta(1)$ time if $x_0 < x_1$ or if $x_0 = x_1$. Let W be a list of n sets such that each member of W is a subset of X with cardinality k. Then duplicate members can be eliminated from the listing of W in $O(kn \log n)$ time.*

Proof: We give the following algorithm.

1. Sort all the members of W individually by lexicographic order. The time required by these sorts is $\Theta(kn \log k)$.
2. Sort W by lexicographic order. This takes $O(kn \log n)$ time.
3. Perform a prefix operation to eliminate each member of W that is equal to a predecessor on the ordered list W. This takes $O(kn)$ time.

The algorithm takes $O(kn \log n)$ time altogether. ■

3 Output bounds

In this section, we obtain an upper bound for the amount of data produced in solution to the PSPM Problem in R^3.

Proposition 3.1 [2]. Let P be a triangle and let S be subset of R^3, with $n = |S|$. Then a listing of all triangles P' with vertices in S such that

- P' is congruent to P, has $O(n^{1.8}[log^* n]^{O(1)})$ output.
- P' is similar to P, has $O(n^{2.2})$ output. ■

The result below for the general case for congruences improves upon the output bound of [5].

Proposition 3.2 *Let P and S be finite subsets of R^3, with $|P| = k \leq n = |S|$. Then a listing of all subsets P' of S such that P' is congruent to P has*

- $O(kn^{1.5}[\frac{\lambda_6(n)}{n}]^{0.25})$ *output if P is a collinear set;*
- $O(kn^{1.8}[\log^* n]^{O(1)})$ *output in general.*

A listing of all subsets P' of S such that P' is similar to P has

- $O(kn^2)$ *output if* P *is a collinear set;*
- $O(kn^{2.2})$ *output in general.*

Proof: Let $P = \{p_0, p_1, \ldots, p_{k-1}\}$. The collinear case for congruences was given in [5]. Therefore, we assume without loss of generality that p_0, p_1, and p_2 are not collinear. By Proposition 3.1, there are $O(n^{1.8}[log^* n]^{O(1)})$ triangles with vertices in S that are congruent to $\{p_0, p_1, p_2\}$. For each such triangle there is at most a constant number of $(k-3)$-tuples of members of S that complete a congruence with P. Hence a listing of all such congruences has $O(kn^{1.8}[log^* n]^{O(1)})$ output.

For similarities, we reason as follows. If P is a collinear set, then for each of the $\Theta(n^2)$ segments (s_i, s_j) with endpoints in S, there are at most two $(k-2)$-tuples $(s_{i_2}, \ldots, s_{i_{k-1}})$ of members of S that complete a similarity with P determined by matching (s_i, s_j) with (p_0, p_1). Hence in this case, a listing of all subsets of S that are similar to P has $O(kn^2)$ output. In the general case, the asserted volume of output, $O(kn^{2.2})$, for subsets P' of S that are similar to P is easily derived from Proposition 3.1 by mimicking the argument given above for congruences. ■

The following gives a lower bound for the volume of the worst case output for the *congruent* version of the PSPM Problem in R^3. The assertion is incorrectly stated in [6], where the proof given is valid for the corrected statement. We thank Peter Braß for pointing out this error.

Proposition 3.3 [6] *The worst case volume of output in listing all solutions to the congruent version of the PSPM Problem in* R^3 *is* $\Omega(\max\{k^2 D_2(\frac{n}{k}),\ D_3(n)\})$. ■

4 Our algorithm

In this section, we give our algorithm for the Point Set Pattern Matching Problem in R^3. In sections 5 and 6, we analyze its running time for both the *all congruences* and *all similarities* versions of the problem, on sequential and coarse-grained parallel computers.

Let $S = \{s_0, s_1, \ldots, s_{n-1}\}$. Let $P = \{p_0, p_1, \ldots, p_{k-1}\}$. Without loss of generality, $i \neq j$ implies $p_i \neq p_j$ and $s_i \neq s_j$.

In the following, we use "match" to mean either "is congruent to" or "is similar to," according to whether we are working with the congruence or similarity version of the PSPM Problem.

1. Determine whether or not P is a collinear set. If so, solve the problem by an algorithm designed for this special case. Otherwise, note an index r such that $\{p_0, p_1, p_r\}$ is not a collinear set.

2. Find each triple (s_i, s_j, s_m) of vertices in S that matches (p_0, p_1, p_r). The details of how this is done differ somewhat between the *congruent* and *similar* versions of the PSPM Problem; they are explained in sections 5 and 6.

3. If $k > 3$, proceed as follows. For each triple (s_i, s_j, s_m) of vertices in S that matches (p_0, p_1, p_r), determine the unique rigid transformation f_{ijm} of R^3 such that $f_{ijm}(p_0) = s_i$, $f_{ijm}(p_1) = s_j$, and $f_{ijm}(p_r) = s_m$.

4. For each of the rigid transformations f_{ijm} of R^3 determined above, compute the set

$$V_{ijm} = \{f_{ijm}(p_q) \mid 2 \leq q \leq k-1,\ q \neq r\}$$

and, for each of its members $f_{ijm}(p_q)$, determine via a search of S which, if any, member of S that $f_{ijm}(p_q)$ equals. If $f_{ijm}(p_q) \notin S$, discard the triple (i, j, m). If $V_{ijm} \subset S$, then $f_{ijm}(P) \subset S$.

5. Among the sets $f_{ijm}(P)$ that match P, there may be duplicate sets determined by distinct f_{ijm}. If desired, we may eliminate such duplication via the sequential algorithm of Proposition 2.3, or its parallel analog, Proposition 6.6.

5 Sequential analysis

In this section, we discuss implementation of the algorithm presented above on a sequential computer, for both the all congruences version and the all similarities version of the PSPM Problem in R^3.

5.1 All congruences

We start with the special case of a collinear pattern set.

Theorem 5.1 *Let P and S be finite subsets of R^3. Let $|P| = k \leq n = |S|$. Suppose P is a collinear set. Then every subset P' of S such that P' is congruent to P can be identified on a sequential computer in*

- $O(n^2 + kD_3(n)\log n) = O(n^2 + kn^{1.5}[\frac{\lambda_6(n)}{n}]^{0.25}\log n)$ *time, in general;*
- $O(kn\delta_2(n)\log n) = O(kn^{1.5}[\frac{\lambda_6(n)}{n}]^{0.25}\log n)$ *time, if* $\delta_2(n) = O(D_3(n)/n)$.

Proof: The general case was shown in [5]. Therefore, we assume $\delta_2(n) = O(D_3(n)/n)$. We give the following algorithm for this case.

1. Apply a minimum operation to each of the $\Theta(k^2)$ pairs of distinct points in P to obtain $d_2(P)$, the length of the shortest line segment with endpoints in P. The time required is $\Theta(k^2) = O(kn)$. In $\Theta(1)$ additional time, we may swap at most two pairs of points so that $d(p_0, p_1) = d_2(P)$.

2. Sort S with respect to lexicographic order. This takes $\Theta(n \log n)$ time.

3. Use a parallel prefix and a parallel postfix operation so that each $s_i \in S$ knows the range S^i of members of S that differ from s_i with respect to the x-coordinate by at most $d_2(P)$. This takes $\Theta(n)$ time. Notice $|S^i| = O(\delta_2(n))$.

4. Note that if $d(s_i, s_j) = d_2(P)$, then $s_j \in S^i$. For each $s_i \in S$, scan the corresponding subrange $S^i \subset S$ to determine which $s_j \in S$ satisfy $d(s_i, s_j) = d_2(P)$. The time required is $O(n\delta_2(n))$, and the number of pairs satisfying $d(s_i, s_j) = d_2(P)$ is $O(n\delta_2(n))$.

5. For each pair (s_i, s_j) satisfying $d(s_i, s_j) = d_2(P)$ (hence matching (p_0, p_1)) and $m \in \{2, 3, \ldots, k-1\}$, use a binary search of S to determine if there exists $s_{q(m)} \in S$ such that $(s_i, s_j, s_{q(m)})$ matches (p_0, p_1, p_m). If a search fails, discard (s_i, s_j); otherwise, $(s_i, s_j, s_{q(2)}, \ldots, s_{q(k-1)})$ matches P. These operations require $O(kn\delta_2(n)\log n)$ time.

6. It may happen that the same match of P is discovered more than once in the previous step. If desired, duplicate matches of P may be eliminated via the algorithm of Proposition 2.3 in $O(kn\delta_2(n)\log n)$ time.

Thus, our algorithm uses $O(kn\delta_2(n)\log n)$ time. Since we assumed $\delta_2(n) = O(D_3(n)/n)$, it follows that the running time is $O(kD_3(n)\log n) = O(kn^{1.5}[\frac{\lambda_6(n)}{n}]^{0.25}\log n)$, which is within a factor of $\log n$ of the output bound of Proposition 3.2. ■

Theorem 5.2 *Let P and S be finite subsets of R^3. Let $|P| = k \le n = |S|$. Then every subset P' of S such that P' is congruent to P can be identified on a sequential computer in*

- $O(n^2 + kD_3(n)\log n) = O(n^2 + kn^{1.5}[\frac{\lambda_6(n)}{n}]^{0.25}\log n)$ *time, if P is a collinear set;*
- $O(kn\delta_2(n)\log n) = O(kn^{1.5}[\frac{\lambda_6(n)}{n}]^{0.25}\log n)$ *time, if P is a collinear set and $\delta_2(n) = O(D_3(n)/n)$;*
- $O(\min\{\delta_3(n)D_3(n), n[\delta_3(n)]^2\} + kn^{1.8}\log n[\log^* n]^{O(1)}) =$
$$O(n^{2.5}[\frac{\lambda_6(n)}{n}]^{0.25} + kn^{1.8}\log n[\log^* n]^{O(1)}) \textit{ time,}$$
in general;
- $O(kn[\delta_3(n)]^2\log n) = O(kn^{1.8}\log n[\log^* n]^{O(1)})$ *time, if $k^2 = O(n[\delta_3(n)]^2\log n)$ and $\delta_3(n) = O(n^{0.4}[\log^* n]^{O(1)})$.*

Proof: We consider the steps of the algorithm given in section 4.

1. It may be decided whether or not P is a collinear set in $O(k)$ time by a simple scan of P, and if P is collinear, the assertions for collinear P follow from Theorem 5.1. Without loss of generality, we proceed assuming that we have determined that p_0, p_1, and p_2, are members of P that are not collinear. If we are assuming $k^2 = O(n[\delta_3(n)]^2\log n)$ and $\delta_3(n) = O(n^{0.4}[\log^* n]^{O(1)})$, we further assume (by swapping $\Theta(1)$ pairs of points) that (p_0, p_1, p_2) is a minimal-diameter triple of non-collinear members of P. This assumption costs $\Theta(k^3)$ time, via application of a minimum operation to the $\Theta(k^3)$ triples of members of P. Note in the general case, we skip this $\Theta(k^3)$-time minimum operation. Let $d_3 = diam(\{p_0, p_1, p_2\})$ (this computation takes $\Theta(1)$ time).

2. Finding the triples (s_i, s_j, s_k) that match (p_0, p_1, p_2) is done as follows.

   - Sort S with respect to lexicographic order. This takes $\Theta(n \log n)$ time.
   - Use a parallel prefix and a parallel postfix operation so that each $s_i \in S$ knows the range S^i of members of S that differ from s_i with respect to the x-coordinate by at most d_3. This takes $\Theta(n)$ time.

     Let $\sigma = \max\{|S^i|\}_{i=1}^n$. We need not compute σ as part of our algorithm, as we use σ only in our analysis. Note we have

     $$\sigma = \begin{cases} O(n) & \text{in general;} \\ O(\delta_3(n)) & \text{if } k^2 = O(n[\delta_3(n)]^2 \log n) \text{ and } \delta_3(n) = O(n^{0.4}[\log^* n]^{O(1)}). \end{cases}$$

   - For each $s_i \in X$, scan the corresponding subrange $S^i \subset S$ to determine which $s_j \in S$ satisfy $d(s_i, s_j) = d_3$. The time required is $O(n\sigma)$. The number of pairs satisfying $d(s_i, s_j) = d_3$ is $O(\min\{D_3(n), n\sigma\})$.
   - For each pair (s_i, s_j) such that $d(s_i, s_j) = d_3$, scan the subrange $S^i \subset S$ for members s_q such that (s_i, s_j, s_q) matches (p_0, p_1, p_2). For one pair (s_i, s_j), this takes $O(\sigma)$ time, so the total time required is $O(\min\{\sigma D_3(n), n\sigma^2\})$.

   Let $\Delta(n)$ be the number of triples (s_i, s_j, s_m) of members of S such that (s_i, s_j, s_m) is congruent to (p_0, p_1, p_2). It follows from Proposition 3.1 that

   $$\Delta(n) = O(\min\{n^{1.8}[\log^* n]^{O(1)}, n\sigma^2\}). \tag{1}$$

3. For each triple (s_i, s_j, s_m) of vertices in S that matches (p_0, p_1, p_2), determining the transformation f_{ijm} requires $\Theta(1)$ time. Altogether, the time required is $\Theta(\Delta(n))$.

4. Computing every V_{ijm} and, for each of its members $f_{ijm}(p_q)$, determining via a binary search of S which, if any, member of S that $f_{ijm}(p_q)$ equals, requires, altogether, $O(k\Delta(n) \log n)$ time.

5. Elimination of duplicate matches of P is done via the algorithm of Proposition 2.3. The time required is $O(k\Delta(n) \log n)$.

Thus, the time required by the algorithm is, in general,

$$O(\min\{\sigma D_3(n), n\sigma^2\} + k\Delta(n) \log n), \tag{2}$$

which simplifies as

$$O(n^{2.5}[\frac{\lambda_6(n)}{n}]^{0.25} + kn^{1.8} \log n [\log^* n]^{O(1)}).$$

When $k^2 = O(n[\delta_3(n)]^2 \log n)$ and $\delta_3(n) = O(n^{0.4}[\log^* n]^{O(1)})$, we have the additional $\Theta(k^3)$-time minimum step; also, in this case, $\sigma = O(\delta_3(n))$, so the expression (2) for the time required must be modified as

$$O(k^3 + n[\delta_3(n)]^2 + k\Delta(n)\log n) = O(kn[\delta_3(n)]^2 \log n) = O(kn^{1.8}\log n[\log^* n]^{O(1)}),$$

which is within a factor of $\log n$ of our output bound. ■

Notice that if $k = \Omega(\frac{n^{0.7}}{\log n})$, the running time for the general case of the problem also simplifies as $O(kn^{1.8}\log n[\log^* n]^{O(1)})$, within a factor of $\log n$ of our output bound.

5.2 All similarities

In this section, we discuss the *similarities* version of the PSPM Problem for a sequential computer. Again, we start with the special case of a collinear pattern set.

Proposition 5.3 *Suppose P is a collinear set. Then every subset P' of S such that P' is similar to P can be identified on a sequential computer in $O(kn^2 \log n)$ time.*

Proof: We give the following algorithm.

1. Sort S by lexicographic order. This takes $\Theta(n \log n)$ time.
2. For each pair (s_i, s_j) of distinct points in S, determine a rigid transformation f_{ij} of R^3 such that $f_{ij}(p_0) = s_i$, $f_{ij}(p_1) = s_j$. This takes $\Theta(n^2)$ time.
3. For each transformation f_{ij} and for each $m \in \{2, \ldots, k-1\}$, use a binary search to determine if $f_{ij}(p_m) \in S$. If for each m, $f_{ij}(p_m) \in S$, then $f_{ij}(P)$ is a subset of S that is similar to P; otherwise, $f_{ij}(P)$ is not similar to P. These operations take $O(kn^2 \log n)$ time.
4. A subset of S that is similar to P may have been discovered by more than one f_{ij}. If desired, duplicates may be removed from the list of similarities by the algorithm of Proposition 2.3 in $O(kn^2 \log n)$ time.

Our algorithm uses $O(kn^2 \log n)$ time. ■

Theorem 5.4 *Let P and S be finite subsets of R^3. Let $|P| = k \leq n = |S|$. Then every subset P' of S such that P' is similar to P can be identified on a sequential computer in*

- *$O(kn^2 \log n)$ time if P is a collinear set;*
- *$O(n^3 + kn^{2.2} \log n)$ time, in general.*

Proof: We consider the steps of the algorithm given in section 4. The analysis follows.

1. We can determine whether or not P is collinear in $O(k)$ time by a simple scan of P. If P is collinear, solve the problem via the algorithm of Proposition 5.3 in $O(kn^2 \log n)$ time. Thus, the collinear case requires $O(kn^2 \log n)$ time, which is within a factor of $\log n$ of the output bound for this case.

   If P is not collinear, we note an index r such that p_0, p_1, and p_r are not collinear, and we proceed as follows. Without loss of generality, $r = 2$ (since otherwise, we may use a $\Theta(1)$-time swap).

2. Finding the matching triples can be done in $\Theta(n^3)$ time by forming all triples of members of S and, for each triple, determining if it is similar to (p_0, p_1, p_2). By Proposition 3.1, $O(n^{2.2})$ of these triples are similar to (p_0, p_1, p_2).

3. Determining the transformations f_{ijm} requires $O(n^{2.2})$ time.

4. Sort S by lexicographic order in $\Theta(n \log n)$ time. Then, for every $f_{ijm}(p_q)$, determining via a binary search of S which, if any, member of S that $f_{ijm}(p_q)$ equals, requires, altogether, $O(kn^{2.2} \log n)$ time.

5. If desired, duplicate (set) matches of P in S may be eliminated via the algorithm of Proposition 2.3 in $O(kn^{2.2} \log n)$ time.

Thus, the general case requires $O(n^3 + kn^{2.2} \log n)$ time. ■

Note that if $k = \Omega(\frac{n^{0.8}}{\log n})$, then the running time is $O(kn^{2.2} \log n)$, which is within a factor of $\log n$ of our output bound.

6 Analysis: coarse grained multicomputer

6.1 Preliminaries

The *Coarse Grained Multicomputer*, or $CGM(n,p)$, consists of p processors, each of which has $\Omega(\frac{n}{p})$ local memory (*i.e.*, $\Omega(\frac{n}{p})$ memory cells of $\Theta(\log n)$ bits apiece). The term *Coarse Grained* implies that the number of processors is "considerably less" than n; we assume, as is typical, $p^2 \leq n$. See [13] for more about this model of parallel computation.

We use the symbol $T_{sort}(n,p)$ to refer to the time required by a $CGM(n,p)$ to sort $\Theta(n)$ data. We will use the following results.

Lemma 6.1 [8] For positive integers k, n, p, we have

$$k \cdot T_{sort}(n,p) = O(T_{sort}(kn,p))$$

on a $CGM(kn,p)$. ■

Proposition 6.2 [8] Let $X = \{x_1, \ldots, x_n\}$ be a set of data values distributed $\Theta(\frac{n}{p})$ per processor of a $CGM(n,p)$ and let $\circ$ be an associative binary operation on X that may be computed in $\Theta(1)$ time. Then a semigroup operation that computes $x_1 \circ x_2 \circ \ldots \circ x_n$ can be computed in $\Theta(\frac{n}{p} + T_{sort}(p^2, p))$ time. At the end of the operation, the result of the computation is stored in every processor. ■

Proposition 6.3 [8] Let $X = \{x_1, \ldots, x_n\}$ be a set of data values distributed $\Theta(\frac{n}{p})$ per processor of a $CGM(n,p)$ and let $\circ$ be an associative binary operation on X that may be computed in $\Theta(1)$ time. Then a parallel prefix operation that computes the set of prefix values $\{x_1, x_1 \circ x_2, \ldots, x_1 \circ x_2 \circ \ldots \circ x_n\}$ can be computed in $\Theta(\frac{n}{p} + T_{sort}(p^2, p))$ time. At the end of the operation, the prefix $x_1 \circ \ldots \circ x_i$ is in the same processor as x_i, $i \in \{1, \ldots, n\}$. ■

Proposition 6.4 [9] *Let X and Y be lists of data, each evenly distributed among the processors of a $CGM(m+n, p)$, where $|X| = m$ and $|Y| = n$. Then a parallel search, in which each member of X searches Y for a value or a range of values, may be performed in $\Theta(T_{sort}(m+n, p))$ time.* ■

Proposition 6.5 [9] *Let $X = \{x_1, \ldots, x_n\}$. Let m be a fixed integer, $1 < m < n$. Then the set of all combinations of members of X taken m at a time, $\{\{x_{i_1}, \ldots, x_{i_m}\} \mid 1 \leq i_1 < \ldots < i_m \leq n\}$, can be formed in*

$$\Theta(\frac{n^m}{p} + p \cdot T_{sort}(n,p)) = O(T_{sort}(n^m, p)) \text{ time}$$

on a $CGM(n^m, p)$. If $p^2 = O(\frac{n^{m-1}}{\log n})$ (which must be true for $m > 2$), the running time is $\Theta(\frac{n^m}{p})$. ■

The next result is a CGM version of Proposition 2.3.

Proposition 6.6 [9] *Let X be a well-ordered set such that for $\{x_0, x_1\} \subset X$, it can be decided in $\Theta(1)$ time if $x_0 < x_1$ or if $x_0 = x_1$. Let W be a list of n sets such that each member of W is a subset of X with cardinality $k \leq n$. Then duplicate members can be eliminated from the listing of W in $\Theta(T_{sort}(kn, p))$ time on a $CGM(kn, p)$.* ■

6.2 CGM Analysis: all congruences

In [9], an algorithm was given to solve the *congruent* version of the PSPM Problem in R^3. However, the resources claimed for this algorithm depend on the incorrect Lemma 3.2 of [6]. In this section, we give a correction and improvements.

First, we consider the special case of a collinear pattern set. We have the following.

Proposition 6.7 [9] Let P and S be finite subsets of R^3. Let $|P| = k \leq n = |S|$. Suppose there is a line $L \subset R^3$ such that $P \subset L$. Then every subset P' of S such that P' is congruent to P, can be identified using a $CGM(n^2 + kD_3(n), p)$ in

$$O(\frac{n^2}{p} + T_{sort}(kD_3(n), p)) = O(\frac{n^2}{p} + T_{sort}(kn^{1.5}[\frac{\lambda_6(n)}{n}]^{0.25}, p)) \text{ time. } \blacksquare$$

A somewhat different solution that in some circumstances is more efficient, is given in the next result.

Proposition 6.8 *Let P and S be finite subsets of R^3. Let $|P| = k \leq n = |S|$. Suppose there is a line $L \subset R^3$ such that $P \subset L$. Then every subset P' of S such that P' is congruent to P, can be identified using $O(T_{sort}(kn\delta_2(n), p) + p \cdot T_{sort}(n, p))$ time on a $CGM(np + kD_3(n), p)$. If $\delta_2(n) = O(D_3(n)/n)$, the running time of the algorithm simplifies as*

$$O(T_{sort}(kn^{1.5}[\frac{\lambda_6(n)}{n}]^{0.25}, p)).$$

Proof: We give the following algorithm.

1. Sort S into lexicographic order. This takes $\Theta(T_{sort}(n, p))$ time.
2. Notice every processor has sufficient memory to store its own copy of the complete lists P and S. Rotate the data of P and S through all processors via $p - 1$ sort operations so that each processor can accumulate both lists in their entirety. This takes
$$\Theta(p \cdot T_{sort}(k + n, p)) = \Theta(p \cdot T_{sort}(n, p)) \text{ time.}$$
3. Form the $\Theta(k^2)$ pairs of distinct members of P. Every processor can form $\Theta(\frac{k^2}{p})$ pairs in
$$\Theta(\frac{k^2}{p}) = O(\frac{kn}{p}) = O(T_{sort}(kn, p)) \text{ time.}$$
4. Compute $d_2(P)$, the minimum length of the line segments determined in the previous step, via a semigroup operation in
$$\Theta(\frac{k^2}{p} + T_{sort}(p^2, p)) = O(\frac{kn}{p} + T_{sort}(n, p)) = O(T_{sort}(kn, p))$$
time. At the end of this step, all processors know the value of $d_2(P)$ and the coordinates of a pair of members of P, without loss of generality (via a constant number of $\Theta(1)$-time swaps in each processor's copy of P), p_0 and p_1, such that $d(p_0, p_1) = d_2(P)$.
5. Recall each processor has a copy of the ordered list S. In parallel, each processor does the following for $\Theta(n/p)$ members of S. Perform a binary search so that each $s_i \in S$ knows the subrange $S^i \subset S$ such that $s_j \in S^i$ implies $|x(s_i) - x(s_j)| \leq d_2(P)$. This takes $O(\frac{n \log n}{p}) = O(T_{sort}(n, p))$ time. Note that $|S^i| = O(\delta_2(n))$.

6. Each $s_i \in S$ can scan S^i to determine all pairs (s_i, s_j) of members of S such that $d(s_i, s_j) = d_2(P)$. These scans take $O(\frac{n\delta_2(n)}{p})$ time.

   Let t be the number of pairs (s_i, s_j) such that $d(s_i, s_j) = d_2(P)$. Then

$$t = O(\min\{n\delta_2(n), D_3(n)\}).$$

7. Use a parallel prefix operation to distribute the pairs (s_i, s_j) of members of S, such that $d(s_i, s_j) = d_2(P)$, evenly among the processors. This takes

$$O(\frac{n\delta_2(n)}{p} + T_{sort}(p^2, p)) = (\text{since } p^2 \leq n)\ O(\frac{n\delta_2(n)}{p} + T_{sort}(n, p)) \text{ time.}$$

8. For every pair (s_i, s_j) such that $d(s_i, s_j) = d_2(P)$, conduct the at most $k-2$ searches necessary to determine if for all $m \geq 2$, there exists $s_{q(m)} \in S$ such that $(s_i, s_j, s_{q(m)})$ matches (p_0, p_1, p_m). Since each processor has its own copy of S, these operations take

$$O(\frac{kt \log n}{p}) = O(T_{sort}(kt, p)) \text{ time.}$$

9. Use a parallel prefix operation to determine from the previous step which subsets of S match P. This takes

$$O(\frac{kt}{p} + T_{sort}(p^2, p)) = O(T_{sort}(kt + p^2, p)) = O(T_{sort}(kt + n, p)) \text{ time.}$$

10. If desired, we may remove duplicate matches of P listed in the previous step via the algorithm of Proposition 6.6. This takes $O(T_{sort}(kt, p))$ time.

The time required by this case of our algorithm is

$$O(\frac{n\delta_2(n)}{p} + T_{sort}(kt + n, p) + p \cdot T_{sort}(n, p)) = O(T_{sort}(kn\delta_2(n), p) + p \cdot T_{sort}(n, p)).$$

Note that if $\delta_2(n) = O(D_3(n)/n)$, the running time of the algorithm simplifies as

$$O(T_{sort}(kD_3(n), p) + T_{sort}(n^{1.5}, p)) = O(T_{sort}(kn^{1.5}[\frac{\lambda_6(n)}{n}]^{0.25}, p)). \blacksquare$$

Theorem 6.9 *Let P and S be finite subsets of R^3. Let $|P| = k \leq n = |S|$. Then every subset P' of S such that P' is congruent to P can be identified on a $CGM(nD_3(n), p)$ in running times given as follows.*

- *If P is a collinear set:*

$$\text{in } O(\frac{n^2}{p} + T_{sort}(kD_3(n), p)) = O(\frac{n^2}{p} + T_{sort}(kn^{1.5}[\frac{\lambda_6(n)}{n}]^{0.25}, p)) \text{ time;}$$

- *If P is a collinear set and $\delta_2(n) = O(D_3(n)/n)$:*

$$\text{in } O(T_{sort}(kn^{1.5}[\frac{\lambda_6(n)}{n}]^{0.25}, p)) \text{ time;}$$

- $O(\frac{n^{2.5}[\frac{\lambda_6(n)}{n}]^{0.25}}{p} + T_{sort}(kn^{1.8}[\log^* n]^{O(1)}, p))$ *time, in general.*

Proof: We give analysis for the algorithm of Section 4.

1. Determining whether or not P is a collinear set (together with other useful chores) can be done as follows.

   - Sort X with respect to the lexicographic order of the s_i entries. This takes $\Theta(T_{sort}(n, p))$ time.
   - Notice that each processor has sufficient memory for copies of the complete lists P and S. Use $p-1$ sort steps to circulate the data of P and S among the processors so that each processor can accumulate these complete lists. This takes

     $$\Theta(p \cdot T_{sort}(k+n, p)) = O(T_{sort}(np, p)) = O(T_{sort}(n^{1.5}, p)) \text{ time.}$$

   - Determine in $\Theta(\frac{k}{p})$ time if each p_m, $m > 1$, is collinear with $\{p_0, p_1\}$.
   - Perform a semigroup (AND) operation in $\Theta(\frac{k}{p} + T_{sort}(p^2, p)) = O(T_{sort}(n, p))$ time to determine from the results of the previous step if P is a collinear set.

   Note we have assumed asymptotically more memory than in Proposition 6.7 and Proposition 6.8. Therefore, if P is a collinear set, proceed with the algorithm of Proposition 6.7 or the algorithm of Proposition 6.8. The asserted times for the corresponding cases follow from these Propositions. We therefore proceed assuming that P is not collinear.

   The discovery of non-collinearity may be followed by a semigroup (minimum) operation, requiring $\Theta(\frac{k}{p} + T_{sort}(p^2, p))$ time, to note an index r such that p_0, p_1, p_r are not collinear, broadcasting r to all processors in $O(p)$ time, and having every processor swap p_r with p_2 (so that p_0, p_1, p_2 are not collinear) in the processor's copy of P. In an additional $\Theta(1)$ time, each processor computes $d_3 = diam(\{p_0, p_1, p_2\})$.

2. We find each triple of members of P that matches (p_0, p_1, p_2) as follows.

   - Use local binary searches (*i.e.*, within a processor's copy of S) so that each $s_i \in S$ knows the range S^i of members of S that differ from s_i with respect to the x-coordinate by at most d_3. This takes $\Theta(\frac{n \log n}{p}) = O(T_{sort}(n, p))$ time. Note that $|S^i| = O(n)$.

- For each $s_i \in S$, scan the corresponding subrange $S^i \subset S$ to determine which $s_j \in S$ satisfy $d(s_i, s_j) = d_3$. This takes $O(\frac{n^2}{p})$ time. It follows from Proposition 2.2 that the number of pairs satisfying $d(s_i, s_j) = d_3$ is $O(D_3(n))$. We use a parallel prefix operation to distribute these pairs evenly among the processors; this takes $O(\frac{n^2}{p} + T_{sort}(p^2, p))$ time.
- For each pair (s_i, s_j) such that $d(s_i, s_j) = d_3$, scan the corresponding subrange $S^i \subset S$ for members s_q such that (s_i, s_j, s_q) matches (p_0, p_1, p_2). The time required is $O(\frac{nD_3(n)}{p})$, and the number of triples matching (p_0, p_1, p_2), $\Delta(n)$, satisfies equation (1).
- We now have all triples of points in S that match (p_0, p_1, p_2). We apply a parallel prefix operation to the set of triples examined above in order to distribute evenly among the processors those triples that match (p_0, p_1, p_2). This takes $O(\frac{nD_3(n)}{p} + T_{sort}(p^2, p))$ time.

3. Determining the transformations f_{ijm} takes $O(\frac{\Delta(n)}{p}) = O(T_{sort}(\Delta(n), p))$ time.
4. We can now find all subsets of S that are congruent to P, as follows. For each triple (s_i, s_j, s_m) that matches (p_0, p_1, p_2), determine if $f_{ijm}(\{p_3, \ldots, p_{k-1}\}) \subset S$ by searching S for each $f_{ijm}(p_q)$, and, if so, the corresponding members of S. Since every processor has its own copy of S, this takes
$$O(\frac{k\Delta(n)\log n}{p}) = O(T_{sort}(k\Delta(n), p)) \text{ time.}$$
5. If desired, duplicate matchings of P in the list produced by the previous step can be eliminated by the algorithm of Proposition 6.6 in $O(T_{sort}(k\Delta(n), p))$ time.

Our algorithm runs in

$$O(\frac{nD_3(n)}{p} + T_{sort}(k\Delta(n), p)) = O(\frac{n^{2.5}[\frac{\lambda_6(n)}{n}]^{0.25}}{p} + T_{sort}(kn^{1.8}[\log^* n]^{O(1)}, p)) \text{ time.}$$

■

Note if $k = \Omega(\frac{n^{0.7}}{\log n})$, the running time for the general case simplifies as $O(T_{sort}(kn^{1.8}[\log^* n]^{O(1)}, p))$, the time required to sort a volume of data equal to our output bound.

6.3 CGM Analysis: all similarities

In this section, we consider the *similar* version of the PSPM Problem on a CGM. We start by considering the special case of a collinear pattern set.

Proposition 6.10 *Suppose P is a collinear set. Then every subset P' of S such that P' is similar to P can be identified on a $CGM(kn^2, p)$ in $O(T_{sort}(kn^2, p))$ time.*

Proof: We give the following algorithm.

1. Sort S by lexicographic order. This takes $\Theta(T_{sort}(n,p))$ time.

2. Form all pairs of distinct members of S in $O(T_{sort}(n^2,p))$ time, by the algorithm of Proposition 6.5.

3. Rotate P throughout the processors so that every processor has a copy of P. The time required for this step is $O(p \cdot T_{sort}(k,p)) = O(T_{sort}(kp,p))$.

4. For each pair (s_i, s_j) of distinct points in S, determine a rigid transformation f_{ij} of R^3 that takes p_0 to s_i and p_1 to s_j. This takes $\Theta(\frac{n^2}{p})$ time.

5. For each transformation f_{ij} and for each $m \in \{2, \ldots, k-1\}$, determine if $f_{ij}(p_m) \in S$. This is done by conducting a parallel search. By Proposition 6.4, this takes $\Theta(T_{sort}(kn^2,p))$ time.

6. If for each m, $f_{ij}(p_m) \in S$, then $f_{ij}(P)$ is a subset of S that is similar to P; otherwise, $f_{ij}(P)$ is not a subset of S. In $\Theta(\frac{kn^2}{p} + T_{sort}(p^2,p)) = O(T_{sort}(kn^2,p))$ time, use a parallel prefix operation to determine from the previous step which subsets of S match P.

7. A subset of S that is similar to P may have been discovered by more than one f_{ij}. If desired, duplicates may be removed from the list of similarities by the algorithm of Proposition 6.6 in $O(T_{sort}(kn^2,p))$ time.

Our algorithm uses $O(T_{sort}(kn^2,p))$ time. ■

Theorem 6.11 *Let P and S be finite subsets of R^3. Let $|P| = k \leq n = |S|$. Then every subset P' of S such that P' is similar to P can be identified on a $CGM(n^3 + kn^{2.2}, p)$ in*

- *$O(T_{sort}(kn^2,p))$ time, if P is a collinear set;*
- *$O[\frac{n^3}{p} + T_{sort}(kn^{2.2},p)]$ time, in general.*

Proof: We show the algorithm of section 4 can be implemented in the times claimed.

1. We determine whether or not P is a collinear set as follows.
   - Broadcast p_0 and p_1 to all processors in $O(p)$ time.
   - Use a semigroup AND operation to determine if, for all $m \in \{2, 3, \ldots, k-1\}$, $\{p_0, p_1, p_m\}$ is a collinear set. This takes $\Theta(\frac{k}{p} + T_{sort}(p^2,p))$.

   If P is a collinear set, we proceed to solve the problem via the algorithm of Proposition 6.10 in $O(T_{sort}(kn^2,p))$ time. Otherwise, we continue as follows. We assume a point p_r was noted such that $\{p_0, p_1, p_r\}$ is not collinear. In $O(p)$ time, swap p_2 and p_r so that $\{p_0, p_1, p_2\}$ is not collinear.

2. Finding triples of members of S that are similar to the triple (p_0, p_1, p_2) is done as follows.

- Form all triples of members of S. By Proposition 6.5, this takes $\Theta(\frac{n^3}{p})$ time.
- In $O(p)$ time, broadcast p_2 to all processors. Now every processor has $\{p_0, p_1, p_2\}$.
- For each of the $\Theta(n^3)$ triples (s_i, s_j, s_m) of members of S, determine whether or not (s_i, s_j, s_m) is similar to (p_0, p_1, p_2). This takes $\Theta(\frac{n^3}{p})$ time.
- By Proposition 3.1, $O(n^{2.2})$ of the triples of members of S are similar to (p_0, p_1, p_2). Perform a parallel prefix operation to distribute these triples evenly among the processors in

  $$\Theta(\frac{n^3}{p} + T_{sort}(p^2, p)) = \Theta(\frac{n^3}{p}) \text{ time,}$$

  by Proposition 6.3.

3. Computing the rigid transformations f_{ijm} requires $O(\frac{n^{2.2}}{p})$ time.

4. Determining the sets V_{ijm} takes $O(\frac{kn^{2.2}}{p})$ time. In an additional $O(T_{sort}(kn^{2.2}, p))$ time, we can search for their members in S, then apply a parallel prefix operation to the results of the searches in

   $$O(\frac{kn^{2.2}}{p} + T_{sort}(p^2, p)) = O(T_{sort}(kn^{2.2}, p)) \text{ time}$$

   to determine which V_{ijm} satisfy $V_{ijm} \subset S$, and, for those that do, the corresponding members of S.

5. The same subset of S may correspond to more than one V_{ijm}. If desired, we may eliminate duplicates via the algorithm of Proposition 6.6 in $O(T_{sort}(kn^{2.2}, p))$ time.

Our algorithm therefore requires $O(\frac{n^3}{p} + T_{sort}(kn^{2.2}, p))$ time. ■

Note if $k = \Omega(\frac{n^{0.8}}{\log n})$, then the running time is $O(T_{sort}(kn^{2.2}, p))$, the time to sort a volume of data equivalent to our output bound.

7 Further remarks

We have given an algorithm to solve the Point Set Matching Problem in R^3. The algorithm runs in $O(n^{2.5}[\frac{\lambda_6(n)}{n}]^{0.25} + kn^{1.8}\log n[\log^* n]^{O(1)})$ sequential time for the all congruences variant of the problem. The previous fastest algorithm for this problem, that of [5], runs in $O(kn^{2.5}[\lambda_6(n)/n]^{0.5}\log n)$ sequential time. (The faster time asserted in [6] is dubious due to an error in Lemma 3.2 of [6]. However, in some cases our running time is faster than that claimed in [6]. Our implementation of this algorithm on a coarse-grained multicomputer similarly corrects the asserted result of [9].) Notice that for both the sequential (RAM) and CGM models of computation, we have studied interesting cases for which our running times are faster than in general; these include cases with running times asymptotically bounded by the sorting times for the output bounds. We have also obtained efficient solutions to the all similarities variant of the problem.

Our improvement of the running time of the algorithm of [5] is largely due to the observation of a smaller bound on the volume of output than was previously observed. There are gaps between our running time and our upper bound on the output, and between our upper bound for the output and the lower bound for the worst case output. Thus, further improvements in the running time of a solution to the PSPM Problem in R^3 may be possible.

References

[1] P. Agarwal, M. Sharir, and P. Shor, "Sharp upper and lower bounds on the length of general Davenport-Schinzel sequences," *Journal of Combinatorial Theory Series A* **52**, pp. 228-274, 1989.

[2] T. Akutsu, H. Tamaki, and T. Tokuyam, "Distributions of distances and triangles in a point set and algorithms for computing the largest common point sets," *Discrete and Computational Geometry* **20**, pp. 307-331, 1998.

[3] H. Alt and L.J. Guibas, "Discrete geometric shapes: matching, interpolation, and approximation," Report **B 96-11**, Institut für Informatik, Freie Universität Berlin, 1996.

[4] L. Boxer, "Finding congruent regions in parallel," *Parallel Computing* **18**, pp. 807-810, 1992.

[5] L. Boxer, "Point set pattern matching in 3-D," *Pattern Recognition Letters* **17**, pp. 1293-1297, 1996.

[6] L. Boxer, "Faster point set pattern matching in 3-D," *Pattern Recognition Letters* **19**, pp. 1235-1240, 1998.

[7] L. Boxer, "Even faster point set pattern matching in 3-D," *Proceedings SPIE Vision Geometry* 8 (1999), 168-178.

[8] L. Boxer, R. Miller, and A. Rau-Chaplin, "Scaleable parallel algorithms for lower envelopes with applications," *Journal of Parallel and Distributed Computing* **53**, pp. 91-118, 1998.

[9] L. Boxer, R. Miller, and A. Rau-Chaplin, "Scaleable parallel algorithms for geometric pattern recognition," *Journal of Parallel and Distributed Computing* 58 (1999), 466-486.

[10] P. Braß and C. Knauer, Testing the congruence of d-dimensional point sets, *Proceedings 16th ACM Symposium on Computational Geometry*, 2000, 310-314.

[11] D.E. Cardoze and L.J. Schulman, "Pattern matching for spatial point sets," *Proceedings Symposium on Foundations of Computer Science*, 1998, 156-165.

[12] K.L. Clarkson, H. Edelsbrunner, L.J. Guibas, M. Sharir, and E. Welzl, "Combinatorial complexity bounds for arrangements of curves and surfaces," *Discrete and Computational Geometry* **5**, pp. 99-160, 1990.

[13] F. Dehne, A. Fabri, and A. Rau-Chaplin "Scalable parallel geometric algorithms for multicomputers," *Proceedings 9th ACM Symposium on Computational Geometry*, 1993.

[14] P.J. de Rezende and D.T. Lee, "Point set pattern matching in d-dimensions," *Algorithmica* **13**, pp. 387-404, 1995.

[15] P. Erdös, "On a set of distances of n points," *American Mathematical Monthly* **53**, pp. 248-250, 1946.

[16] M.T. Goodrich and D. Kravetz, "Point set pattern matching," Johns Hopkins University Department of Computer Science Technical Report **JHU-92/11**, 1992.

[17] C-L Lei and H-T Liaw, "A parallel algorithm for finding congruent regions," *Computers and Graphics* **16**, pp. 289-294, 1992.

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

[19] S. Schirra, "Approximate decision algorithms for approximate congruence," *Information Processing Letters* **43**, pp. 29-34, 1992.

[20] Z.C. Shih, R.C.T. Lee, and S.N. Yang, "A parallel algorithm for finding congruent regions," *Parallel Computing* **13**, pp. 135-142, 1990.

[21] J. Spencer, E. Szeremedi, and W.T. Trotter, Jr., "Unit distances in the Euclidean plane," in *Graph Theory and Combinatorics*, pp. 293-303, Academic Press, London, 1984.