Efficiently Enumerating Scaled Copies of Point Set Patterns

Aya Bernstine* and Yehonatan Mizrahi†

Abstract

Problems on repeated geometric patterns in finite point sets in Euclidean space are extensively studied in the literature of combinatorial and computational geometry. Such problems trace their inspiration back to Erdős' original work on this topic. In this paper, we investigate the problem of finding scaled copies of any pattern within a set of n points, that is, the algorithmic task of efficiently enumerating all such copies. We initially focus on one particularly simple pattern of axis-parallel squares, and present an algorithm with an $O(n\sqrt{n})$ running time and $O(n)$ space for this task, involving various bucket-based and sweep-line techniques. Our algorithm's running time is worst-case optimal, as it matches the known lower bound of $\Omega(n\sqrt{n})$ on the maximum number of axis-parallel squares determined by n points in the plane, thereby solving an open question for more than three decades of realizing that bound for this pattern. We extend our result to an algorithm that enumerates all copies, up to scaling, of any full-dimensional fixed set of points in d-dimensional Euclidean space, that runs in time $O(n^{1+1/d})$ and space $O(n)$, matching the more general lower bound due to Elekes and Erdős.

1 Introduction

The problems of geometric point pattern matching and the identification of repeated geometric patterns are fundamental computational problems with a myriad of applications, ranging from computer vision [12, 10], image and video compression [1], model-based object recognition [15], structural biology [11] and even computational chemistry [9]. Such problems were motivated in part by questions regarding the maximal number of occurrences of a given pattern determined by a set of points, a field historically inspired by Erdős' well-known Unit Distance Problem (1946) regarding the maximal number of unit distance pairs induced by such sets [8]. Our paper approaches the computational problems of identifying patterns using tools and techniques encountered in the framework of computational geometry, ensuring exact, provably correct and efficient solutions.

In this paper, we analyze the problem of identifying and listing all translated and scaled copies of *any* point set pattern in Euclidean space, termed homothetic copies of this set, where the scaling is applied identically in all axes. We begin with focusing on the problem of repeated patterns of squares having axis-parallel edges in the plane, where a square is defined by a subset of four points that constitute its vertices. As articulated in 1990 by van Kreveld and de Berg [13], the maximum possible number of axis-parallel squares determined by n points in the plane is $\Theta(n\sqrt{n})$ (attained, for example, in a regular $\sqrt{n}\times\sqrt{n}$ grid), and those can be enumerated in time $O(n\sqrt{n}\log n)$[1] and space $O(n)$ by an algorithm whose extension also treats the enumeration of all full-dimensional axis-parallel d-dimensional hypercubes in d-dimensional Euclidean space in time $O(n^{1+1/d}\log n)$. This exhibits a logarithmic-factor gap separating this computational result from the lower bound of a maximum of $\Theta(n^{1+1/d})$ possible hypercubes, raising the challenge of overcoming this gap as an open question. We remark that in [14], a later journal version of [13], an algorithm for the planar case that works in time $O(n\sqrt{n\log n})$ is presented. However, as the authors point out, its approach does not generalize to higher dimensions.

The combinatorial result from [13] was further extended by Elekes and Erdős [7], establishing a bound of $\Theta(n^{1+1/d})$ on the maximum number of copies of *any* full-dimensional pattern (i.e., a set of points that generates the vector space) in $\mathbb{Q}^d$. The computational aspect of it occurs in [4], providing an algorithm that works in time $O(n^{1+1/d}\log n)$, assuming that the pattern and d are constant, for the task of enumerating all such copies, exhibiting the same logarithmic-factor gap between the two results.

1.1 Our Results

Our main result of this paper is an efficient deterministic algorithm that enumerates all scaled copies of any fixed d-dimensional pattern, for any constant d. The treatment of general patterns appeared, e.g., in [4], but [13] were the first to raise the question of whether it is computationally feasible to realize the combinatorial

*School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel, aya.bernstine@mail.huji.ac.il

†School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel, yehonatan.mizrahi@mail.huji.ac.il

[1]The analysis given throughout this paper of time and space complexities is based on the relatively non-restrictive Pointer Machine model of computation [3], as mentioned later in this paper.

bound of $\Theta(n\sqrt{n})$ possible axis-parallel *squares*, thereby improving their algorithmic result. Our algorithm fully answers this question which was open for more than three decades. To this end, we use in our algorithm a reduction from arbitrary input points to points having "compressed" coordinates, that is, we relabel the coordinates, allowing the use of linear-time sorting methods. Second, we deploy a sweep-line scanning sub-procedure that marks points forming a square, instead of searching those in a set, avoiding the logarithmic cost of searching a point in a set. Third, we relabel the sum and the difference of the input coordinates, in addition to the relabeling of the coordinates themselves. We show why the last step is crucial for the algorithm to succeed in Section 2.

Theorem: *Given a planar set P of points of size n, all axis-parallel squares defined by points from P can be enumerated in time $O(n\sqrt{n})$ and $O(n)$ space.*

Our main result for general patterns relies on the ideas from the previous theorem. Specifically, we relabel some affine transformations of the input coordinates, a relabeling that creates a representation of the points for the purpose of sweep-line scanning them.

Theorem: *Given a fixed set Q of points of full dimension in the d-dimensional Euclidean space, and a set P of points of size n, all scaled copies of Q determined by subsets of P can be enumerated in time $O(n^{1+1/d})$ and $O(n)$ space.*

The running time in this theorem matches the corresponding lower bound of the same magnitude, and improves the best known running time of $O(n^{1+1/d}\log n)$ for the specific case of d-dimensional hypercubes [13], extended later for general arbitrary patterns [4]. Note that although the improvement suggested is by a logarithmic factor, the upshot is an asymptotically *worst-case optimal* algorithm[2] in terms of running time analysis, even for the most general case of arbitrary patterns. This can be compared with [6], where the authors studied the problem of enumerating all *rotated* copies of a given pattern, improving the running time of the trivial algorithm for this companion task by a logarithmic factor as well. An excellent survey that covers this variant of our problem can be found in [2].

Aside from the worst-case optimality of our results, the techniques deployed form a rather general scheme, and may therefore be potentially useful to treat other variants of the problem studied.

[2]For the task of outputting an explicit representation of all copies of the pattern, rather than some other representation of this set of copies, that later needs to be further parsed.

2 Axis-Parallel Squares

In this section, we present an efficient algorithm that reports all axis-parallel squares defined by a planar set of n points. A relatively efficient algorithm, devised by van Kreveld and de Berg [13], works as follows (Note that we refer, for any x_0, to the set of all points whose x coordinate is x_0, as the "column" corresponding to x_0. Moreover, we refer to columns with at most $\sqrt{n}$ points as "short columns").

Squares-Listing$(p_1,\ldots,p_n)$:

1. Build a balanced search tree T and an array A on the input, sorted by the x coordinate.
2. For every pair of points p and q in A residing in a short column, search in T whether they can be complemented to a square from the right or from the left. Report each square found unless the other two vertices defining it are on a short column to the left of p and q.
3. Delete all short columns from T and A, and convert each remaining point (x,y) to (y,x).
4. Apply step 2 on the remaining converted points.

It operates correctly with a running time of $O(n\sqrt{n}\log n)$ and $O(n)$ space, in essence, since the total number of searched points defined in each of the two iterations of step 2 is

$$O\left(\sum_i s_i^2\right) \le O\left(\sum_i s_i\sqrt{n}\right) = \\ = O\left(\sqrt{n}\cdot\sum_i s_i\right) \le O\left(n\sqrt{n}\right)$$

where s_i denotes the length of the i'th column scanned. Every pair is scanned during its course, since there are at most $\frac{n}{\sqrt{n}}$ original long columns (otherwise there are more than n points), so the length of each column in step 4 is at most $\frac{n}{\sqrt{n}} = \sqrt{n}$. We strive for an algorithm with a running time of $O(n\sqrt{n})$ and space $O(n)$. As shown in [13]:

Theorem 1 *(van Kreveld, de Berg) For a set P of n points in d-dimensional space, the maximal number of 2^d points that are subsets of P and that form the vertices of an axis-parallel hypercube is $\Theta(n^{1+1/d})$.*

This theorem induces a lower bound on the running time of the optimal relevant algorithm. Our result bridges the gap between this bound, and the previously best known upper bound.

2.1 Main Ideas Towards an Improvement

Assume that all input points have coordinates in $\{1, \ldots, n\}$. Instead of searching in a set for the *query points* that complement the pair $(x, y), (x, y + \delta)$ to a square, i.e., the points in the pair $(x+\delta, y), (x+\delta, y+\delta)$ and those in the pair $(x - \delta, y), (x - \delta, y + \delta)$, we apply the following procedure: We put all query points along with the original points in an array, apply radix sort on it, treating each point as a two-digit number in base n corresponding to its two coordinates, and then scan and mark all positive query points. That is, we mark each query point adjacent to an existing input point sharing the same coordinates, or to an already marked identical query point. The resulting marked query points define the existing squares.

However, we cannot generally assume that all coordinates are taken from $\{1, \ldots, n\}$. We address this issue by "shrinking" the coordinates of all input points by relabeling their coordinates to values in $\{1, \ldots, n\}$. The main caveat, though, is that arithmetical considerations regarding these labels are invalid, as the proportions are not necessarily preserved after relabeling.

So, we avoid using arithmetic considerations when defining the query points q_1, q_2 that complement the pair $p_1 = (x, y), p_2 = (x, y + \delta)$ to a square (from the right, assuming $\delta > 0$). Instead of using the invalid label $x + \delta$ as a coordinate, we make use of the diagonals by replacing each point (x, y) with $(x, y, x + y, x - y)$ and relabel each of those four coordinates for all points to values in $\{1, \ldots, n\}$. We call the points after this relabeling the *post-labeled points*. Then, the pair q_1, q_2 (with q_2 above q_1) is defined using *identical* labels as those of p_1, p_2. The query point q_1 is defined having the same horizontal y label as p_1 and the same diagonal $x + y$ label as p_2. The point q_2 is treated similarly, only with the second diagonal. Searching in this manner, we can use two out of the four coordinates for each point we search, leaving the other two as wildcards.

Another related observation is that the linear transformation that rotates a vector (x, y) in the plane by 45^o and stretches it by $\sqrt{2}$ yields the vector $(x + y, y - x)$, as illustrated in Figure 1. So, this process is in fact a labeling of the post-rotated points.

2.2 The Efficient Solution

The ideas from the previous subsection lead to our main theorem of this section. We will first describe our algorithm in full detail, and then analyze its correctness and its complexity.

Theorem 2 *Given a planar set P of points of size n, all axis-parallel squares defined by points from P can be enumerated in time $O(n\sqrt{n})$ and $O(n)$ space.*

Proof. The following algorithm is considered:

Amplified-Squares-Listing$(p_1, \ldots, p_n)$:

1. Change the representation of each point $p = (x, y)$ to the representation $(x, y, x + y, y - x)$. Map each x coordinate in the input to a value in $\{1, \ldots, n\}$ according to its ranking, using a sorting algorithm. Perform a similar procedure for the y coordinates, the $x + y$ coordinates and the $y - x$ coordinates. Apply this mapping on the input, to obtain the post-labeled points.

2. Build an array A on the input points, sorted by the x coordinate.

3. For each pair of post-labeled points $p_1 = (x, y_1, w_1, z_1)$ and $p_2 = (x, y_2, w_2, z_2)$ with $y_2 > y_1$, out of the first n pairs of points in A that reside in a short column – construct the query points $q_1 = (*, y_1, w_2, *), q_2 = (*, y_2, *, z_1)$ that complement p_1, p_2 to a square from the right, both pointed by the same pointer R, and the query points $q_1' = (*, y_1, *, z_2), q_2' = (*, y_2, w_1, *)$ that complement to a square from the left, both pointed by the same pointer L. The wildcards replace the unknown coordinates.

4. Place each query point defined by its y and w coordinates in an array B_1 along with all input points, and apply radix sort on B_1 based on those two coordinates. Perform a similar procedure for points of the form of q_2 and q_1' from step 3 in another array B_2.

5. Scan B_1 and mark each query point adjacent to an input point sharing the same coordinates, or to an already marked identical query point. Act similarly on B_2. Scan the list of pointers defined in step 3, and report each square found (a pointer with both points marked), unless the other two vertices defining it are on a short column and complement to a square from the left, to avoid reporting the same square more than once.

6. Perform steps 3-5 iteratively on each subsequent n pairs of points in A in a short column.

7. Delete all points that are on short columns from A. Convert each remaining point (x, y) to (y, x). Apply steps 1-6 on the remaining converted points.

Algorithm's Correctness: Most of the main ideas behind the algorithm's correctness were described in Subsection 2.1. Some other details: All squares having at least one edge on a short column are reported in step 5 of the algorithm, before applying step 7. The rest have both edges on long columns, and so they are reported in step 7.

Given a pair $p_1 = (x_1, y_1)$ and $p_2 = (x_1, y_2)$ with $y_2 > y_1$, the pair $q_1 = (x_2, y_1)$ and $q_2 = (x_2, y_2)$ that

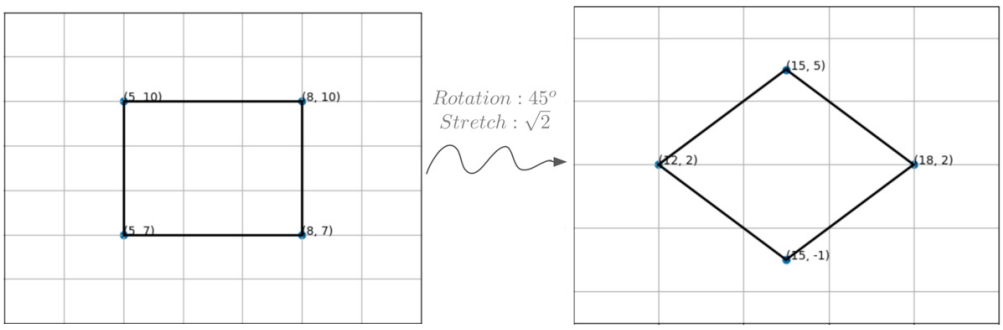

Figure 1: Illustrating the rotation by 45^o and the stretch by a factor of $\sqrt{2}$ applied on four points in the plane. Each point (x, y) was converted to the point $(x + y, y - x)$ as a result.

complements to a square from the right (i.e., $x_2 > x_1$) maintains that $x_2 + y_1 = x_1 + y_2$ since

$$x_2 = x_1 + |y_2 - y_1| \Rightarrow x_2 + y_1 = x_1 + (y_2 - y_1) + y_1 = x_1 + y_2$$

As for q_2, the latter equality also shows that $x_2 - y_2 = x_1 - y_1$ by subtracting $y_1 + y_2$ from both sides. These are exactly the query points defined by the algorithm, up to the labeling that maintains those properties. The analysis for the pair that complements to a square from the left is symmetric.

Algorithm's Complexity: As for the running time, the first two steps of the algorithm cost $O(n \log n)$ using some standard sorting algorithm, e.g., merging sort. Each time step 3 is performed, at most $2n$ query points are constructed in $O(n)$ time. Each time steps 4-5 are performed, two arrays, each of size at most n, are sorted and then scanned in a linear time. Marking the obtained squares in step 5, based on the marked queries, is also carried out in $O(n)$ time by scanning the constructed pointers from step 3. The total number of query points constructed after finishing step 6 is

$$O\left(\sum_i s_i^2\right) \leq O\left(\sum_i s_i \sqrt{n}\right) = \\ = O\left(\sqrt{n} \cdot \sum_i s_i\right) \leq O\left(n\sqrt{n}\right)$$

where s_i denotes the length of the i'th short column, i.e., the number of points in it. As mentioned, each batch of $O(n)$ queries is handled in $O(n)$ time, so the total running time analysis for steps 1-6 of this algorithm is $O(n\sqrt{n})$. The analysis for the converted points in step 7 is symmetric. It only remains to notice that the number of pairs, this time, is

$$O\left(\sum_i d_i^2\right) \leq O\left(\sum_i d_i \sqrt{n}\right) = \\ = O\left(\sqrt{n} \cdot \sum_i d_i\right) \leq O\left(n\sqrt{n}\right)$$

where d_i denotes the length of the i'th row out of the remaining rows, after deleting the short columns. We used the fact that there are at most $\frac{n}{\sqrt{n}}$ long columns, as otherwise there are more than n input points. Therefore, the length of each remaining row, after deletion, is at most $\frac{n}{\sqrt{n}} = \sqrt{n}$, and all points are treated.

As for the space complexity, note that each of the data structures defined in the above algorithm is of size $O(n)$, and that each step involving those structures does not cost more than $O(n)$ space. □

Note that both of these algorithms need not rely on any random-access operation, as no pointer arithmetic or tests on pointers other than equality tests need to be performed. Dereferencing of pointers, along with arithmetic operations on data and comparisons on data are performed, but those are allowed in the Pointer Machine model [3]. The only step which classically involves random access to array cells is the one in which radix sort is used, but even this can be be adjusted to work in the mentioned model ([5]). This statement is true also for the algorithms given in the following sections.

3 Axis-Parallel Hypercubes

This subsection discusses the particular case of axis-parallel hypercubes in d-dimensional Euclidean space. Although following immediately from Theorem 3 given in the following section, it presents some of the ideas behind it in a clearer manner, and can serve as a warm-up for that theorem. We provide an algorithm with a

running time complexity of $O(n^{1+1/d})$ and with a linear space complexity, addressing the open question of matching the lower bound from Theorem 1. Our algorithm builds on the techniques and the observations from Section 2, with some additions and adjustments so it complies with the properties of the d-dimensional space.

One observation that is true for the d-dimensional case, is that any two points with all but one equal coordinate, determine 2^{d-1} full-dimensional possible hypercubes. Denote such a pair of points by t and r. Each of these hypercubes is uniquely associated with a vector $e \in \{-1,1\}^{d-1}$, in which the j'th coordinate determines the direction of progress from t and r along the j'th axis, where j is any coordinate except the one in which they differ. In a similar fashion to the planar case, we would like to relabel the input coordinates, their sums and their differences, place them in an array, radix sort it and mark the correct vertices that complement to a hypercube in an efficient manner, while scanning this array. Moreover, we scan only pairs of points lying on short axis-parallel lines, similarly to the planar case, only that this time, by "short" we mean having not more than $n^{1/d}$ point on it.

Proposition: Given a set P of points of size n in d-dimensional Euclidean space, all axis-parallel full-dimensional hypercubes defined by points from P can be enumerated in time $O(n^{1+1/d})$ and $O(n)$ space.

Proof. The following algorithm establishes the proposition's statement:

Amplified-Hypercubes-Listing$(p_1, \ldots, p_n)$:

1. For each input point $p = (x_1, x_2, \ldots, x_d)$, add the following additional list of coordinates:

   $$((x_i - x_j), (x_i + x_j) \;\mid\; \forall 1 \le i < j \le d)$$

   Map each of those augmented coordinates, including the original ones, to a label in $\{1, \ldots, n\}$.

2. Build an array A on the input points, sorted by each of their coordinates based on the coordinates' order, except for the last *original* coordinate (i.e., x_d).

3. For each pair of points t, r that lie in the same short axis-parallel line, having the same coordinates except for the last, out of the first n pairs with this property, add $2^d - 2$ query points which define together a hypercube. Do this for all 2^{d-1} possible hypercubes in the following manner. First, any axis-parallel hypercube having t and r as its vertices, is defined using one additional vertex

   $$r' = (r_1 + e_1 \cdot \delta, \ldots, r_{d-1} + e_{d-1} \cdot \delta, t_d)$$

   where r_i is the i'th coordinate in r (and similarly for t), $\delta = r_d - t_d$ and $e \in \{-1,1\}^{d-1}$. The rest of the vertices in each such hypercube are defined similarly, except for replacing all subsets of the coordinates in the vector e by zeros, and using t instead of r.

   Now, define the coordinates that are to be searched – *not* in the aforementioned arithmetic manner, but using the labels from step 1 instead. That is, translate $r_i + e_i \cdot \delta$ to the label of $r_i + r_d$ if $e_i = 1$, and to that of $r_i - r_d$ otherwise. Fill in the unknown coordinates using wildcards, as those are uniquely defined anyway, given the others.

4. Place all query points defined by the same coordinates in an array along with all input points. Apply radix sort on each of those arrays, according to the known coordinates in it.

5. Scan each array from step 4, and mark each query point adjacent to an input point sharing the same coordinates, or to an already marked identical query point. Report all hypercubes that were found (by checking that all vertices are present for each hypercube), except for hypercubes that have two vertices on a short axis-parallel line of the same type, only with a smaller index.

6. Perform steps 3-5 on each subsequent n pairs of points in A on a short axis-parallel line of that type.

7. Delete all points that are on a short axis-parallel line of a currently analyzed type from A, and convert each remaining point $(x_1, x_2, \ldots, x_d)$ to $(x_d, x_1, \ldots, x_{d-1})$. Apply steps 1-6 on the remaining converted points. This step is carried out $d-1$ times.

Algorithm's Correctness: Almost all details regarding the analysis of the correctness of this algorithm already appeared in that of Amplified-Squares-Listing$(p_1, \ldots, p_n)$. As for the phase of searching by labels, note that if r'_j is some unknown coordinate, for which we only have an undesired arithmetic definition based on the coordinate r_j and on δ, then it holds that

$$r'_j = r_j + \delta = r_j + (r_d - t_d) \Longrightarrow r'_j + t_d = r_j + r_d$$

which exactly corresponds to the labels that the algorithm searches (the treatment of positive or negative values of δ is symmetric), and similarly for subtraction.

Algorithm's Complexity: The running time analysis is similar to the running time analysis of Amplified-Squares-Listing$(p_1, \ldots, p_n)$ with the following differences. The relabeling in step 1 and the sorting of A in step 2; the definitions of 2^{d-1} hypercubes, each consisting of additional $2^d - 2$ points in step 3; applying

radix sort on $2^{O(d)}$ arrays in step 4; the linear scanning of those arrays in step 5 and applying step 7 for $d-1$ times – all of those involve multiplying the complexity of the planar case by at most a constant factor of $2^{O(d)}$. The only major difference concerns the total number of query points defined each time step 7 in the algorithm is invoked. In the treatment of the first $d-1$ axis-parallel lines, only short lines are considered, so the cost for each line is

$$O\left(\sum_i s_i^2\right) \leq O\left(\sum_i s_i \cdot n^{1/d}\right) = \\ = O\left(n^{1/d} \cdot \sum_i s_i\right) \leq O\left(n^{1+1/d}\right)$$

where s_i denotes the length of the i'th short axis-parallel line of that form. Regarding the treatment of the last axis-parallel line, we note that all remaining such lines are short. Assume that there exists a long remaining line. Then all points on it are on long axis-parallel lines, with respect to some axis, which induces more points that are on long lines with respect to another axis, yielding that there are more than $\left(n^{1/d}\right)^d$ input points in total, which is obviously a contradiction. Thus, all input points are treated with the mentioned running time, so the total running time analysis is indeed $O(n^{1+1/d})$. The analysis of the space complexity is similar, with a constant multiplicative factor of $2^{O(d)}$ compared to the analysis of the planar case. Note that although being exponential in d, as also occurs in the solution from [13], the running time can be regarded as polynomial in d and the size of the pattern, as presented in Section 4. □

4 The General Case

In this section, we describe an algorithm that enumerates all scaled copies of any fixed arbitrary full-dimensional pattern in d-dimensions. For general fixed patterns, where d is fixed, our algorithm works in time $O(n^{1+1/d})$ and $O(n)$ space. This answers the open question of realizing the lower bound of [7].

Theorem 3 *Given a fixed set Q of points of full dimension in the d-dimensional Euclidean space, and a set P of points of size n, all scaled copies of Q determined by subsets of P can be enumerated in time $O(n^{1+1/d})$ and $O(n)$ space.*

Proof. We first assume that no three points in Q are on the same line, and present an appropriate algorithm for this case. Then we describe how this algorithm can be adjusted to handle the more general case.

Amplified-Patterns-Listing$(p_1, \ldots, p_n)$:

1. Rotate the pattern points such that two of them, p and q, share afterwards all coordinates except the last one. Apply this rotation on the input points.
2. For each point $r \neq p, q$ in Q, compute $d-1$ hyperplanes of dimension $d-1$ that include p and r, and an additional hyperplane including q and r, altogether defining r uniquely. Apply each of the $d \cdot |Q|$ transformations corresponding to those hyperplanes on each input point, attach those values to the original points' list of coordinates, and label the resulting values – the augmented coordinates (original coordinates along with those corresponding to the transformations) using $\{1, \ldots, n\}$.
3. Build an array A on the input points, sorted by all original coordinates by their order.
4. Scan A. For each pair r and t on an axis-parallel line that corresponds to step 1 that also has at most $n^{1/d}$ points ("short" line), construct the rest of the $|Q|-2$ points that complement to a pattern using the labels obtained from step 2, until constructing n such sets of queries. Point each such set that corresponds to a single copy by the same pointer.
5. Place all query points that are defined by the same augmented coordinates in an array with all input points, forming several such arrays. Apply radix sort on each such array.
6. Scan each array from step 5, and mark each query point adjacent to an identical input point or an already marked query point. Scan the list of pointers defined in step 4, and report each copy found (a pointer with all points marked), only after applying on it the rotation which is inverse to that of step 1.
7. Perform steps 4-6 on each subsequent n pairs of points in A of the form of step 4.
8. Apply steps 2-7 for each pair among the pattern points that determines a line parallel to that through p and q, excluding enumeration of duplicate copies (similarly to the identification of duplicate squares, i.e., using an appropriate ordering). Delete all points on those "short" lines from step 4. Apply steps 1-7 on the remainder, for a different pair of points from the pattern, and perform this $d-1$ times.

As for the case where at least three points from Q are collinear: If $d \geq 2$, then in step 2 of the above algorithm, if the point r lies on the line that goes through p and q, it is not uniquely defined as the intersection of a line that goes through p and a hyperplane of dimension $d-1$ that goes through q. However, since Q is full-dimensional, there is another point r' which is not on

that line. So, instead, define r as the intersection of a line that goes through p and a hyperplane of dimension $d-1$ that goes through r', and label the additional extra coordinate that corresponds to r' in the augmented form. However, note that in step 4 of the above algorithm we scan input points that correspond to p and q, and not to r', yet for each such a pair (that corresponds to p and q) we still need to know the value of the corresponding extra coordinate of r' which is associated with them, and is unknown at that moment. This is bypassed by first defining the query points that corresponds to r' for each such a pair, then marking all of the positive r' query points that also exist in the input, and only then defining the query points that corresponds to r based on the r' coordinate fetched during the scanning process.

If $d = 1$, one can use a completely different approach than that described earlier. We sort the input points on the line, and then place $|Q|$ pointers on the $|Q|$ leftmost input points. If the point which is pointed by the third pointer is too close to the second one in terms of the proportions from Q, increment the third pointer. If it is too far, increment to second pointer. If they align correctly, increment the rest of the pointers until either a copy of the pattern is found, or until one of the pointers is too distant. Then continue and advance the second pointer, and proceed similarly. In this manner, for each leftmost point, the rest of the pointers only advance forward with a cost of $O((|Q|-1)\cdot n)$. Since this is performed n times, the desired running time of $O(n^2)$ is obtained.

Analysis: The main ideas behind the correctness of Amplified-Patterns-Listing$(p_1,\ldots,p_n)$ already appeared in Section 2. Aside from those ideas, note that step 2, in fact, defines each point as the intersection of a line and a $(d-1)$-dimensional hyperplane, and under the assumption that no three points are on the same line, the points are uniquely defined in that manner.

As for the running time, note that there is no remaining long line analyzed at the ultimate iteration. Otherwise, all points on it are on another long line defined by a linearly independent vector. This induces more points on a different long line, and so forth, yielding that there are more than $\left(n^{1/d}\right)^d = n$ input points, a contradiction. Other than that, we did not need the lines to be axis-parallel, but rather merely that the corresponding vectors form an independent set.

Compared to the squares or the hypercubes case, steps 1-3 cost $O(poly(d\cdot|Q|)\cdot n\log n)$ using a sorting algorithm; constructing $O(n)$ sets of queries in step 4 costs $O(poly(d\cdot|Q|)\cdot n)$ and constructing $O(|Q|)$ arrays of size $O(n)$ in step 5 has a similar running time; applying radix sort and then scanning and marking those arrays also cost $O(poly(d\cdot|Q|)\cdot n)$ (this is multiplied by $|Q|$ due to pairs among the pattern points which are parallel). As mentioned, only short lines are scanned during the algorithm's course, and since each batch of size n costs $O(poly(d\cdot|Q|)\cdot n)$, then each dimension costs $O(poly(d\cdot|Q|))$ multiplied by

$$O\left(\sum_i s_i^2\right) \le O\left(\sum_i s_i\cdot n^{1/d}\right) = \\ = O\left(n^{1/d}\cdot\sum_i s_i\right) \le O\left(n^{1+1/d}\right)$$

where s_i denotes the length of the i'th short line of that form. This is multiplied by d iterations, and results in a running time of $O(poly(d\cdot|Q|)\cdot n^{1+1/d})$, which is $O(n^{1+1/d})$ under our assumptions. Space complexity is linear due to similar arguments to those above, and to those presented in Section 2. □

5 Conclusion and Further Work

In this paper, we analyzed the problem of enumerating all scaled copies of a pattern in a set of n points in time $O(n^{1+1/d})$, answering open questions from [13] and [4] by realizing the lower bound due to Elekes and Erdős [7]. We relied on some existing ideas, amplified using bucket-based methods, sweep-line scanning and more. As far as we are aware of, the combinations of these techniques this way was not noted in the literature so far for similar tasks. One open question is whether these techniques can be adjusted for different pattern matching problems. Other questions include comparing the task of finding one copy of a pattern with the task of enumerating all copies of it ([13] show a separation between those for d-dimensional boxes), and similarly for the task of counting the number of copies instead of outputting them. In addition, the existence of an output-sensitive algorithm for our problem, and the existence of an efficient enumeration algorithm for patterns not of a constant size, form another two open questions for further research.

References

[1] M. Alzina, W. Szpankowski, and A. Grama. 2d-pattern matching image and video compression: theory, algorithms, and experiments. *IEEE Transactions on Image Processing*, 11(3):318–331, 2002.

[2] D. Avis, A. Hertz, and O. Marcotte. Graph theory and combinatorial optimization. *Springer Science & Business Media*, 2005.

[3] A. Ben-Amram. What is a "pointer machine"? *ACM SIGACT News*, 26(2):88-95, 1995.

[4] P. Braß. Combinatorial geometry problems in pattern recognition. *Discrete & Computational Geometry*, 28:495–510, 2002.

[5] A. Buchsbaum, H. Kaplan, A. Rogers, and J. Westbrook. Linear-time pointer-machine algorithms for least

common ancestors, mst verification, and dominators. *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 279–288, 1998.

[6] P. de Rezende and D. Lee. Point set pattern matching in d-dimensions. *Algorithmica*, 13:387–404, 1995.

[7] G. Elekes and P. Erdős. Similar configurations and pseudo grids. *Intuitive geometry*, 63:85–104, 1994.

[8] P. Erdős. On sets of distances of n points. *American Mathematical Monthly*, 53:248–250, 1946.

[9] P. Finn, L. Kavraki, J. Latombe, R. Motwani, C. Shelton, S. Venkatasubramanian, and A. Yao. Rapid: Randomized pharmacophore identification for drug design. *Computational Geometry: Theory and Applications*, 10:324–333, 1997.

[10] D. Mount, N. Netanyahu, and J. L. Moigne. Efficient algorithms for robust feature matching. *Pattern Recognition*, 32(1):17–38, 1999.

[11] R. Norel, D. Fischer, H. Wolfson, and R. Nussinov. Molecular surface-recognition by a computer vision-based technique. *Protein engineering*, 7:39–46, 1994.

[12] G. Schindler, P. Krishnamurthy, R. Lublinerman, Y. Liu, and F. Dellaert. Detecting and matching repeated patterns for automatic geo-tagging in urban environments. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–7, 2008.

[13] M. van Kreveld and M. de Berg. Finding squares and rectangles in sets of points. In *Graph-Theoretic Concepts in Computer Science*, 1990.

[14] M. van Kreveld and M. de Berg. Finding squares and rectangles in sets of points. In *BIT Numerical Mathematics*, 31(2):202–219, 1991.

[15] V. Zografos and B. Buxton. Affine invariant, model-based object recognition using robust metrics and bayesian statistics. In *Kamel M., Campilho A. (eds) Image Analysis and Recognition. ICIAR 2005. Lecture Notes in Computer Science*, volume 3656, 2005.