Smallest k-point enclosing rectangle and square of arbitrary orientation

Sandip Das [a,*], Partha P. Goswami [b], Subhas C. Nandy [a]

[a] *Indian Statistical Institute, Kolkata 700108, India*
[b] *Calcutta University, Kolkata 700029, India*

Received 12 August 2004; received in revised form 8 February 2005

Available online 8 April 2005

Communicated by F.Y.L. Chin

Abstract

Given a set of n points in 2D, the problem of identifying the smallest rectangle of arbitrary orientation, and containing exactly k ($\leqslant n$) points is studied in this paper. The worst case time and space complexities of the proposed algorithm are $O(n^2 \log n + nk(n-k)(n-k+\log k))$ and $O(n)$, respectively. The algorithm is then used to identify the smallest square of arbitrary orientation, and containing exactly k points in $O(n^2 \log n + kn(n-k)^2 \log n)$ time.

Keywords: Levels of arrangement; Plane sweep; Computational geometry; Optimization

1. Introduction

Given a set S of n points in 2D, and an integer k ($\leqslant n$), we consider the problem of identifying the smallest rectangle on the plane which encloses exactly k points of S. A restricted variation of this problem has already been investigated, where the desired rectangle is axis-parallel. The first result on this problem appeared in [1] with time and space complexities $O(k^2 n \log n)$ and $O(kn)$, respectively. The time complexity result is improved to $(k^2 n + n \log n)$ in [5,8]. The space complexities of [8,5] are $O(kn)$ and $O(n)$, respectively. All these algorithms are efficient when k is small. For large k (very close to n), an efficient algorithm is proposed in [14]. It runs in $O(n + k(n-k)^2)$ time and using $O(n)$ space. In d (> 2) dimensions, the algorithm proposed in [14] runs in $O(dn + dk \cdot (n-k)^{2(d-1)})$ time using $O(dn)$ space. In all these variations, the points are assumed to be in general position, i.e., no two points lie on the same horizontal or vertical line, and the desired rectangle is isothetic and closed (i.e., enclosed points may lie on the boundary of the rectangle). A similar problem is studied in [12], where n points are distributed on the plane, and the proposed algorithm identifies the smallest circle con-

* Corresponding author.
E-mail address: sandipdas@isical.ac.in (S. Das).

taining exactly k points in $O(n \log n + (n-k)^3 n^{\varepsilon})$ time for some $\varepsilon > 0$.

We consider the generalized version of this problem, where the desired rectangle may be of arbitrary orientation. Our proposed algorithm is simple and it runs in $O(n^2 \log n + kn(n-k)(n-k+\log k))$ time using $O(n)$ space. The proposed technique can also identify the smallest k-enclosing square of arbitrary orientation in $O(n^2 \log n + kn(n-k)^2 \log n)$ time and $O(n^2)$ space. The time complexity of identifying smallest k-enclosing square is comparable with that of smallest k-enclosing circle proposed in [12].

The motivation of studying this generalized version comes from pattern recognition and facility location, where essential features are represented as a point set, and the objective is to identify a precise cluster (region) containing desired number of features [3,4,10].

2. Smallest k-point enclosing rectangle

Let $S = \{p_1, p_2, \ldots, p_n\}$ be the set of given points. The objective is to identify the smallest area rectangle of arbitrary orientation which contains exactly k points of S. Without loss of generality, we describe our method under the general position assumptions for the points in S, i.e., (i) no two points lie on a vertical line, and (ii) no three points are collinear. But the assumptions can be relaxed with a minor modification of the algorithm. The following definition is motivated from the fact that many rectangles exist which can cover a given set of k points.

Definition 1. An exactly k point enclosing rectangle R is said to be a *k-rectangle* if there does not exist another rectangle R' having area less than that of R and containing the same set of k points.

Let a *k-rectangle* contain a designated subset S' ($\subset S$) of k points. In other words, this rectangle encloses the convex hull of S'. Thus, each side of a *k-rectangle* contains at least one member of S'. The following result implies that at least one edge of the *k-rectangle* contains two points of S'.

Result 1 [15]. The minimum area rectangle enclosing a convex polygon has a side collinear with one of the edges of the polygon.

Given this characterization of a *k-rectangle*, we can specify major steps of the algorithm for identifying the minimum area *k-rectangle*. We consider each pair of points $p_i, p_j \in S$, and execute two passes. Below, we explain *Pass-1* of the algorithm, which identifies all the *k-rectangle*s with p_i and p_j at bottom boundary. The *Pass-2* works in a similar manner to identify all the *k-rectangle*s with p_i and p_j at top boundary. In each *pass*, we maintain the smallest area *k-rectangle* found so far. Finally, the smallest area *k-rectangle* among the two *pass*es is reported.

Let L_{ij} denote the line joining p_i and p_j, and assume that p_i be to the left of p_j on L_{ij}. Let L_i and L_j be two lines perpendicular to L_{ij}, drawn at p_i and p_j, respectively. This splits the half-plane above L_{ij} into three parts, say LEFT, MID and RIGHT. Our algorithm for *Pass-1* sweeps a line L parallel to L_{ij} in the upward direction. The following observation lists all possible situations that may take place when a new point p_m is encountered by the sweep line L. Let S_L, S_R and S_M denote the set of points encountered by L in LEFT, RIGHT and MID, respectively, up to and including p_m. We assume that $p_i, p_j \in S_M$. We use $|A|$ to denote the cardinality of the set A.

Observation 1.

(a) *If $|S_L| + |S_M| + |S_R| < k$ then no k-rectangle exists with p_m at top boundary and p_i, p_j at bottom boundary.*

(b) *If $|S_M| = k$ and $p_m \in$ MID, then only one k-rectangle exists with the line segment $[p_i, p_j]$ as its bottom boundary, p_m at top boundary, left and right boundaries are defined by lines L_i and L_j, respectively. Here, one need not have to consider any point above p_m, and hence the sweep of L stops.*

(c) *If $|S_M| < k$ and $p_m \in$ MID, then $\min(k - |S_M|, |S_L|, |S_R|) + 1$ k-rectangles exist with p_m at top boundary and p_i, p_j at bottom boundary.*

(d) *If $|S_M| < k$ and $p_m \in$ LEFT, then if the rectangle with one side aligned with L_{ij} and the line segment $[p_m, p_j]$ as diagonal contains more than k points then no k-rectangle exists with p_m at top boundary and p_i, p_j at bottom boundary. Moreover, in the further sweep if another point p_ℓ appears whose projection on L_{ij} is to the left of that*

of p_m, then also no k-rectangle exists with p_l at top boundary and p_i, p_j at bottom boundary.

(e) *If $|S_M| < k$ and $p_m \in$ LEFT, then if the rectangle with one side aligned with L_{ij} and the line segment $[p_m, p_j]$ as diagonal contains r ($\leqslant k$) points then* $\min(k - r, |S_L \cup S_M| - r, |S_R|) + 1$ *k-rectangles exist with p_m at top boundary and p_i, p_j at bottom boundary.*

(f) *Results similar to* (d) *and* (e) *hold if $|S_M| < k$, $p_m \in$ RIGHT.*

Observation 1 leads to the following result.

Lemma 1. *Let $p_i, p_j, p_m \in S$ be three points. The number of k-rectangles with one edge passing through p_i, p_j, and its parallel edge passing through p_m, may be anything from 0 to $k-2$.*

Proof. Let us consider the line L_{ij} as the x-axis. Consider the set of points encountered by the sweep line L until it reaches p_m. Among these points, let $\chi \leqslant k$ be the number of points having x-coordinate between $\min(x(p_i), x(p_j), x(p_m))$ and $\max(x(p_i), x(p_j), x(p_m))$. Thus, in a valid *$k$-rectangle*, at most $(k - \chi)$ points may have x-coordinate less than $\min(x(p_i), x(p_j), x(p_m))$. This indicates that the number of possible *k-rectangles* with (p_i, p_j) at bottom boundary and p_m at its top boundary is $(k - \chi + 1)$. As the minimum value of χ is 3, the maximum number of possible *k-rectangles* is $k-2$. □

2.1. Algorithm

We use geometric duality [6] for systematically identifying the point-pairs in S. Here (i) a point $p = (a, b)$ in the primal plane is mapped to the line p^*: $y = ax - b$ in the dual plane, and (ii) a non-vertical line ℓ: $y = mx - c$ in the primal plane is mapped to the point $\ell^* = (m, c)$ in the dual plane. Immediate consequence of this definition is that a point p is below (resp. on, above) a line ℓ in the primal plane if and only if the line p^* is above (resp. on, below) the point ℓ^* in the dual plane.

Let $\mathcal{A}(H)$ denote the arrangement of the set of lines $H = \{p_i^*,\ i = 1, 2, \ldots, n\}$, where p_i^* is the dual (line) of the point $p_i \in S$. The vertices in $\mathcal{A}(H)$ are denoted by $\{v_{ij},\ i \neq j,\ i = 1, 2, \ldots, n,\ j = 1, 2, \ldots, n\}$, where v_{ij} is the dual of L_{ij}. We consider each vertex $v_{ij} \in \mathcal{A}(H)$, and execute *Pass-1* which computes the *k-rectangles* in the primal plane with p_i and p_j at its bottom boundary. Our algorithm does not store the entire arrangement in the memory. The vertices in $\mathcal{A}(H)$ are considered by sweeping a vertical line from left to right through the arrangement $\mathcal{A}(H)$ (see [7,11] for details of the approach).

During the sweep of the vertical line through $\mathcal{A}(H)$, the sweep line status is maintained as an array B of size n. It contains the lines of H in the order in which they intersect the sweep line in its current position, from bottom to top. Each element $B[\alpha]$ (representing a line p^*) is attached with an *id* of the corresponding point in S. The array B is initialized by the lines of H in increasing order of the ordinates of their intersections with the line $X = -\infty$. The sweep process is guided by an event-queue Q. It is maintained in the form of a min-heap. It stores the vertex v_{ij} if p_i^* and p_j^* are consecutive entries in the array B and intersect at v_{ij} to the right of the sweep line. During the sweep, the next event point v_{ij} is obtained from Q in $O(\log n)$ time. The updating of Q needs another $O(\log n)$ time using the method described in [7,11]. The processing of v_{ij} includes swapping two consecutive entries p_i^* and p_j^* in the array B, and considering all *k-rectangles* with p_i and p_j at its bottom boundary.

Lemma 2. *If $B[\alpha] = p_i^*$ and $B[\alpha + 1] = p_j^*$, then the points in the primal plane corresponding to all the lines stored at $B[\theta], \theta = 1, 2, \ldots, \alpha - 1$, are above the line L_{ij}, and the points corresponding to all the lines stored in $B[\theta]$, $\theta = \alpha + 2, \ldots, n$, are below the line L_{ij}.*

Proof. Follows from the definition of geometric duality. □

Thus, while processing a vertex $v_{ij} \in \mathcal{A}(H)$ (i.e., the pair of points p_i and p_j in the primal plane), the upward sweep for scanning the points above the line L_{ij} in increasing order of their distances from L_{ij} is equivalent to scanning the elements $B[\theta]$, $\theta = \alpha - 1, \alpha - 2, \ldots, 1$. In order to analyze the combinatorial complexity of *k-rectangles* on the plane, we need the following definition:

Definition 2 [6]. A point q in the dual plane is at *level* θ $(0 \leqslant \theta \leqslant n)$ if and only if there are exactly θ lines in H that lie strictly below q. The θ*th level* of $\mathcal{A}(H)$ is the closure of a set of points on the lines of H which are exactly at level θ in $\mathcal{A}(H)$.

It also needs to be mentioned that the array B actually contains the levels of $\mathcal{A}(H)$ at the current position of the sweep line.

Lemma 3. *While processing a vertex v_{ij} at level θ (say) of $\mathcal{A}(H)$,*

(i) *if $\theta < k$, then no k-rectangle exists with p_i and p_j at its bottom boundary, and*
(ii) *if $\theta \geqslant k$, then the number of k-rectangles with p_i and p_j at bottom boundary is at most $(k-2) \times (\theta - k + 1)$.*

Proof. Part (i) of the lemma (i.e., $\theta < k$) follows from Lemma 2 and Observation 1(a). In part (ii) (i.e., $\theta \geqslant k$), for each $B[\beta]$, $\beta = \theta - k, \theta - k - 1, \ldots, 0$, at most $k-2$ k-*rectangles* may exist with the corresponding point at top-boundary and p_i, p_j at bottom boundary (by Lemma 2). Hence, the result in this case follows. □

An integer variable χ_M is used to store the number of points in S_M that are encountered by the sweep line L up to the current instant of time. The points in S_L and S_R that are encountered during the upward sweep of L are stored in two link-lists T_L and T_R. At an instant of time, T_L stores at most $k - \chi_M$ points in S_L closest to L_i in increasing order of their distances from L_i; T_R also stores at most $k - \chi_M$ points in S_R closest to L_j in increasing order of their distances from L_j. Initially, both T_L and T_R are empty, $\chi_M = 2$, and the upward sweep of L starts. For the first $k-1$ encountered points, no k-*rectangle* is generated (by Lemma 3(i)). For each of them, if it is in MID then χ_M is incremented. If it is in LEFT or RIGHT it is accumulated in T_L or T_R, respectively, in unordered manner. When the kth point is encountered, the reporting of k-*rectangles* starts. We now arrange the elements in T_L in increasing order of their distances from L_i. This needs copying the elements in a temporary array, then sorting the array and finally copying the sorted array in T_L. The elements in T_R are also arranged in similar manner.

During the execution, we use a scalar variable *opt* which stores the area of the smallest k-*rectangle* that is identified up to the present instant of time. The four corners of the corresponding rectangle are also preserved in another four-tuple, called *opt_rectangle*.

Let p be the point faced by the sweep line, which corresponds to $B[\theta]$ (in the sweep line status array). $|T_L|$, $|T_R|$ and χ_M indicates the number of points in the respective sets prior to the insertion of p in its appropriate set. The following cases need to be considered.

$p \in$ LEFT: If $|T_L| + \chi_M = k$, then the point farthest from L_i is deleted from T_L. Note that, p is not inserted in T_L just now; it will be inserted in the next step. We execute the following steps to report the k-*rectangles*.

- Perform a linear scan to identify the $(k - |T_L| - \chi_M)$th element in T_R (from left).
- Next scan T_L from its leftmost element and T_R from the $(k - |S_L| - \chi_M)$th element in ordered manner until (i) the proper position of p in T_L is reached or (ii) the end of T_R is reached. At each move, a k-*rectangle* is reported. Its bottom side passes through p_i and p_j, top side passes through the point p; the left and right sides are bounded by the respective elements in T_L and T_R.
- The area of each of these rectangles is compared with that of the existing *opt*. If it is less than the existing value of *opt*, then *opt* is updated, and the corresponding rectangle is stored in *opt_rectangle*.
- In case (i), p is inserted in T_L at its proper position. In case (ii), the scanning in T_L proceeds further to insert p in T_L at its proper position.

$p \in$ RIGHT: Similar to the earlier case.

$p \in$ MID: We increment χ_M.

If $\chi_M = k$, then a k-*rectangle* is reported with bottom boundary equal to the line segment $[p_i, p_j]$ and p at top boundary, and the sweep of L stops (by Observation 1(b)).

Otherwise, if $|T_L| + \chi_M = k + 1$, then the point farthest from L_i is deleted from T_L. Similarly, if $|T_R| + \chi_M = k + 1$, then the point farthest from L_j is deleted from T_R.

Next, *k-rectangles* are reported as described for $p \in \text{LEFT}$. Here the question of inserting p in any set does not arise.

Remark 1. We can avoid the general position assumption (i), if any, by rotating the plane by a small amount. We can relax the general position assumption (ii) by allowing more than one (dual) lines pass through a vertex. While processing such a vertex, we consider each pair of lines passing through that vertex, and execute the line sweep of *Pass*-1. The sweep line L first encounters the points corresponding to the other (dual) lines passing through the same vertex, and then proceeds upward.

2.2. Complexity

Theorem 1. *Given a* 2D *plane containing n points, the number of k-rectangles on the plane is* $O(nk(n-k)^2)$.

Proof. By Observation 1(a), no *k-rectangle* exists for the vertices $v_{ij} \in \mathcal{A}(H)$ which are at level $\leqslant k$ in *Pass-1* of our algorithm. For each vertex of $\mathcal{A}(H)$ which is at level $\geqslant k$, at most $O(k(n-k))$ *k-rectangles* are reported (by Lemma 3). The number of vertices of $\mathcal{A}(H)$ which are at level $\geqslant k$ is $O(n(n-k+1))$ (see Corollary 5.17 of [13]). Thus, at most $O(nk(n-k)^2)$ *k-rectangles* can be reported in *Pass-1* of all the vertices in $\mathcal{A}(H)$. Similarly, in *Pass-2* of all the vertices in $\mathcal{A}(H)$ can also generate at most $O(nk(n-k)^2)$ *k-rectangles*. □

Theorem 2. *The time and space complexities of our algorithm are* $O(n^2 \log n + nk(n-k)(n-k+\log k))$ *and* $O(n)$, *respectively.*

Proof. The scanning of all the vertices of $\mathcal{A}(H)$ using a vertical sweep line needs $O(n^2 \log n)$ time. We now analyze the time complexity of *Pass-1* for processing a vertex v_{ij} at a level α $(\geqslant k)$.

During the upward sweep, the points corresponding to $B[\alpha-1], B[\alpha-2], \ldots, B[\alpha-k-1]$ are only accumulated in T_L or T_R (without maintaining the order). This needs $O(k)$ time. Prior to the processing of $B[\alpha-k]$, we arrange the accumulated elements in T_L and T_R in ordered manner. This needs sorting the elements in T_L and T_R separately, and hence it requires $O(k \log k)$ time.

Next, we start reporting the *k-rectangles* with p_i and p_j at bottom boundary. For each point p corresponding to $B[\alpha-k-i]$, $i = 1, 2, \ldots, (\alpha-k)$, we examine all possible *k-rectangles* with p at its top boundary, and finally, insert p in the respective list at its proper position. This needs at most $O(k)$ time. Thus, the total time required for processing v_{ij} is $O(k + k \log k + k(n-k))$ in the worst case.

As the combinatorial complexity of $(\geqslant k)$-levels is $O(n(n-k))$, the total time complexity of our algorithm is $O(nk(n-k)(n-k+\log k))$. The space complexity result follows from the fact that the total arrangement is not maintained in the memory during the execution of the algorithm. □

3. Smallest k point enclosing square

In this section, we show that our above algorithm can easily be used to design a simple algorithm for reporting the smallest arbitrarily oriented k point enclosing square. A k point enclosing square is called a *k-square* if its size is smallest among all squares containing the same set of points. Without loss of generality, we assume that the points appearing on the boundary of a square are inside it. We now classify the *k-squares* with respect to the number of points η present on its boundary. Obviously, no *k-square* is possible with $\eta = 0$ or 1. The only possibility with $\eta = 2$ is that, the two points appear at the two diagonally opposite corners of the corresponding *k-square* (see Fig. 1(a)). Below, we list the possible configurations of a *k-square* with $\eta = 3$ and 4. It also needs to be mentioned that we need not have to consider any configuration with $\eta > 4$, since a suitable selection of a subset among its boundary points leads to one of the configurations which we have dealt for $\eta \leqslant 4$.

Two instances of $\eta = 3$ may produce a valid *k-square*:

(i) Two points on one side of the corresponding *k-square* and one point on its parallel side (see Fig. 1(b)).
(ii) One point at a corner of the corresponding *k-square*, and the other two points lie on the two adjacent sides of the diagonally opposite corner (see Fig. 1(c)).

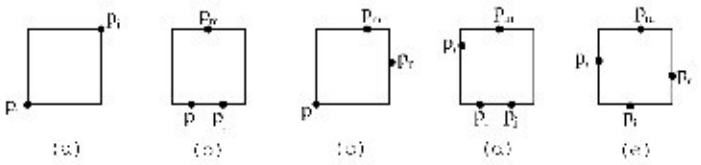

Fig. 1. Different possibilities of *k-squares*.

For other k point enclosing squares with $\eta = 3$, we can push them in the direction perpendicular to the empty side such that the point on its opposite side goes inside the square. Thus, we have a configuration with $\eta = 0$, 1 or 2. Thus a smaller *k-square* will always exist with the same set of k points.

For $\eta = 4$, the possible configurations for a *k-square* are as follows:

(i) One side of the said square contains two points of S, its parallel side and one adjacent side contain one point each (see Fig. 1(d)).
(ii) Each side of the said square contains exactly one point of S (see Fig. 1(e)).

For all other configurations of the k point enclosing squares with $\eta = 4$, a smaller square can always be obtained with the same set of points inside it.

Consider the type (i) *k-square* with $\eta = 3$. Its one side contains two points p_i and p_j, the corresponding parallel side contains a point p_m, but the other two mutually parallel sides do not contain any point. Let it contains a subset $S' \subset S$ of k points inside it. Thus, if we slide it horizontally, it will remain a *k-square* containing the same set S' of points and having the same area. We do minimum horizontal sliding of the *k-square* such that one of its vertical sides touches a point in S. If it is a member in S', it is a type (i) *k-square* with $\eta = 4$ (see Fig. 1(d)). Such *k-squares* are identified in Procedure A, stated below. But, if it is a member $p_r \in S \setminus S'$, then it gives birth to a square containing $k+1$ points. We first move the other vertical side to touch with the point in S'. Thus, it becomes a rectangle with $k+1$ points with one side passing through p_i and p_j, and each of the other three sides passing through exactly one point, namely p_ℓ, p_m and p_r, respectively. In Procedure B, we shall extract a *k-square* from such a configuration which is of type (ii) with $\eta = 4$ as shown in Fig. 1(e).

Lemma 4. *Given four points $p_i, p_m, p_\ell, p_r \in S$, the number of squares whose each of the four sides contains one of these four points is at most* 2.

Proof. Let the lines which form the desired square are L_i, L_m, L_ℓ and L_r, where L_a pass through the point p_a, where $a = i, m, \ell, r$ (see Fig. 1(e)). Also let the gradient of the lines L_i and L_m be μ and therefore the gradient of the lines L_ℓ and L_r be $-\frac{1}{\mu}$. Let A, B, C and D denote the points of intersection of (L_i and L_r), (L_r and L_m), (L_m and L_ℓ) and (L_ℓ and L_i), respectively. Equating the length of the line segments $[A, B]$, $[A, D]$, we get a quadratic equation in μ. If the roots of this equation are imaginary, no *k-square* is possible. Otherwise, at most two *k-squares* may exist. For each of them, we need to check whether

(i) p_i, p_r, p_m and p_ℓ appear on the sides DA, AB, BC, and CD, respectively, and
(ii) still it contains exactly k points. □

Remark 2. Each *k-square* must satisfy $\eta = 4$. Note that, the *k-rectangles* with $\eta = 2$ and case (ii) of $\eta = 3$ can be considered as degenerate cases of the *k-rectangles* with $\eta = 4$.

3.1. Algorithm

For $\eta = 2$, we consider each pair of points $p_i, p_j \in S$, and test whether the square with $[p_i, p_j]$ as diagonal contains exactly k points by using the method of simplex range searching [9]. If *yes*, we compare its area with the existing optimum. The *k-squares* with $\eta = 3$ and $\eta = 4$ are identified using the method for identifying *k-rectangles* with minor modification as described below.

We describe two procedures for identifying the instances with $\eta = 3$ and $\eta = 4$, respectively. As in the earlier problem, we consider the vertices of the arrangement $\mathcal{A}(H)$. At each vertex (representing a line L_{ij}), the upward (resp. downward) sweep of a line

L parallel to L_{ij} is also performed as earlier and using the same data structure. Procedure A identifies the smallest *k-square* with both p_i, p_j at its bottom boundary, and some other point p_m at its top boundary. Procedure B identifies the smallest *k-square* with either p_i or p_j at its bottom boundary, and each of the other three sides contains a point of S.

Procedure A. Let the length of the line segment $[p_i, p_j] = r$. Initially, we consider the $r \times r$ square with one side aligned with $[p_i, p_j]$. Using simplex range searching, we *count* the number of points inside that square. If it exceeds k, then this procedure terminates without computing any *k-square*. If it is equal to k then also the procedure terminates reporting the *k-square* whose two corners on the bottom boundary contain p_i and p_j. But, if the *count* is less than k, the following steps are executed.

Step A1. Our upward sweep proceeds, and the points in LEFT and RIGHT, encountered by the sweep line L, are accumulated in T_L and T_R. For the points in MID, χ_M is incremented. The sweep continues until it hits a point p_m whose distance from L_{ij} is greater than or equal to r.

Step A2. Now, the median find algorithm [2] is invoked for the points in T_L to collect only k points (if available) which are closest to L_i. They are also orderly stored in T_L. Similarly, among the points in T_R only k points are stored in increasing order of distances from L_j.

Step A3. The upward sweep continues and for each encountered point p_m, we test the existence of a *k-square* with p_i, p_j at bottom boundary and p_m at top boundary as mentioned below. Sweep terminates as soon as a *k-square* is found.

- Let the distance of p_m from L_{ij} is equal to h. We sequentially walk from left to right along T_L. For each element $\pi \in T_L$ (corresponding to the projection of a point $p_a \in S_L$ on L_{ij}) we choose a point ϕ on L_{ij} such that the length of the interval $[\pi, \phi] = h \geqslant [\pi, p_j]$. If the $h \times h$ square with p_i, p_j at its bottom boundary, p_m at top boundary, and p_a at its left boundary contains exactly k points, it is an instance of case (i) with $\eta = 4$. This can be observed from T_R during the walk.

Before describing Procedure B for identifying the *k-squares* of type (ii) with $\eta = 4$, we need the following important observation.

Observation 2. *Let R be the optimum k-*square *containing one designated point of S in each of its boundaries. If we rotate each edge by the same angle keeping the designated points on its boundaries, it becomes a rectangle. We continue rotation till one of its boundaries hit another point inside/outside the square. Thus, we have a k-*rectangle *or a $(k+1)$-*rectangle *whose one side contains two points of S and each of the other three sides contains exactly one point of S. In other words, this k-*rectangle *or $(k+1)$-*rectangle*, on rotation, produces the desired (optimum) k-*square.

Thus, for each pair of points (p_i, p_j), we identify all *k-rectangles* and *(k + 1)-rectangles* with (p_i, p_j) at one side. For each of these rectangle R, we execute following steps:

Procedure B.

Step B1. Let the other three sides of R contain p_r, p_m, p_ℓ, respectively. We consider two quadruples $\{p_i, p_r, p_m, p_\ell\}$ and $\{p_j, p_r, p_m, p_\ell\}$. For each quadruple, we compute the smallest valid square with those four points at its four boundaries respectively using Lemma 4.

Step B2. Next, we apply simplex range searching technique to each of these squares for testing whether it is a *k-square* or not. If *yes*, it is a candidate for the optimum *k-square*.

It needs to be mentioned that, all the rectangles generated in this process with p_i (resp. p_j) at bottom-left (resp. bottom-right) corner will generate the *k-squares* of type (ii) with $\eta = 3$ by applying required rotation.

3.2. Complexity

Theorem 3. *Our proposed algorithm correctly computes the smallest k-*square *in $O(n^2 \log n + nk(n-k)^2 \cdot \log n)$ time using $O(n^2)$ space.*

Proof. The correctness follows from Remark 2, and the fact that we have considered all the configurations with $\eta = 4$ including the degenerate cases. Though we

have considered the only case of $\eta = 2$, i.e., two points on the diagonally opposite corners of a *k-rectangle* separately, it can be generated by Procedure B with little modification.

In the time complexity analysis, we use the fact that the simplex range counting query can be performed in $O(\log n)$ time using $O(n^2)$ time and space [9].

For $\eta = 2$, we consider all possible pair of points $(p_i, p_j) \in S$, and perform simplex range searching to test whether the square with (p_i, p_j) as diagonal is a *k-square*. This needs $O(n^2 \log n)$ time.

Using almost a similar argument of proving Theorem 2, the worst case time required for executing Procedure A is $O(n^2 \log n + nk(n-k)(n-k+\log k))$.

In Procedure B, all the *k-rectangles* and $(k+1)$-*rectangles* are considered. Each of them gives birth to at most four squares at Steps B1 and B2. For each square, the simplex range counting query is to be invoked. As the total number of possible *k-rectangles* is $O(nk(n-k)^2)$ (see Theorem 1), the total time required for execution of Procedure B for all the vertices of $\mathcal{A}(H)$ is $O(nk(n-k)^2 \log n)$. □

References

[1] A. Aggarwal, H. Imai, N. Katoh, S. Suri, Finding k point with minimum diameter and related problems, J. Algorithms 12 (1991) 38–56.

[2] A.V. Aho, J.E. Hopcroft, J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA, 1974.

[3] H.C. Andrews, Introduction to Mathematical Techniques in Pattern Recognition, Wiley-Intersciences, New York, 1972.

[4] T. Asano, B. Bhattacharya, M. Keil, F. Yao, Clustering algorithms based on maximum and minimum spanning trees, in: Proc. 4th Annual Symp. on Computational Geometry, 1988, pp. 252–257.

[5] A. Datta, H.-P. Lenhof, C. Schwarz, M. Smid, Static and dynamic algorithms for k-point clustering problems, J. Algorithms 19 (1995) 474–503.

[6] H. Edelsbrunner, Algorithms in Computational Geometry, Springer, Berlin, 1987.

[7] H. Edelsbrunner, E. Welzl, Constructing belts in two-dimensional arrangements with applications, SIAM J. Comput. 15 (1986) 271–284.

[8] D. Eppstein, J. Erickson, Iterated nearest neighbors and finding minimal polytopes, Discrete Comput. Geom. 11 (1994) 321–350.

[9] P.P. Goswami, S. Das, S.C. Nandy, Triangular range counting query in 2D and its application in finding k nearest neighbors of a line segment, Comput. Geom. 29 (2004) 163–175.

[10] J.A. Hartigan, Clustering Algorithms, Wiley, New York, 1975.

[11] R. Janardan, F.P. Preparata, Widest-corridor problems, Nordic J. Comput. 1 (1994) 231–245.

[12] J. Matoušek, On geometric optimization with few violated constraints, Discrete Comput. Geom. 14 (1995) 365–384.

[13] M. Sharir, P.K. Agarwal, Davenport–Schinzel Sequences and their Geometric Applications, Cambridge University Press, New York, 1995.

[14] M. Segal, K. Kedem, Enclosing k points in the smallest axis parallel rectangle, Inform. Process. Lett. 65 (1998) 95–99.

[15] G. Toussaint, Solving geometric problems with the rotating calipers, in: Proc. IEEE MELECON'83, 1983, pp. A10.02/1–4.